```python
# Create a Python program that accepts two numbers from the user and
performs basic arithmetic
# operations (addition, subtraction, multiplication, division).
Display the results with appropriate
# labels.
# 1. Use input() to read numbers.
# 2. Use variables and expressions.
# 3. Handle division carefully (no division by zero).

a=int(input("Enter an integer A : "))
b=int(input("Enter an integer B: "))

op = input("Enter the operation (+, -, *, /): ")

if op == '+':
    print(f"{a} + {b} = {a + b}")
elif op == '-':
    print(f"{a} - {b} = {a - b}")
elif op == '*':
    print(f"{a} * {b} = {a * b}")
elif op == '/':
    if b != 0:
        print(f"{a} / {b} = {a / b}")
    else:
        print("Error: Division by zero is not allowed.")
else:
    print("Invalid operation. Please enter one of +, -, *, /.")
```

```
Enter an integer A : 3
Enter an integer B: 0
Enter the operation (+, -, *, /): /
Error: Division by zero is not allowed.
```

```python
# Write a Python program that prints a famous quote, ensuring the
quote and the author's name
# are properly formatted with the use of quotation marks and escape
characters.
# Albert Einstein once said, "Imagination is more important than
knowledge."
# 1. Use proper use of quotes and escape sequences (\").
# 2. Use string variables to store the quote and author separately.


# The \" escape sequence is used to include double quotes inside the
string.
# The f-string (formatted string literal) helps combine the variables
neatly.

author = "Albert Einstein"
quote = "Imagination is more important than knowledge."
```

```python
print(f'{author} once said, \"{quote}\"')
```

```
Albert Einstein once said, "Imagination is more important than
knowledge."
```

```python
# Ask the user for a number and determine whether it is even or odd.
Display an appropriate
# message.
# 1. Use if-else decision-making statements.
# 2. Use modulus operator (%) for the check.
# 3. Print clear output

num = int(input("Enter a number: "))
if num % 2 == 0:
    print(f"The number {num} is even.")
else:
    print(f"The number {num} is odd.")
```

```
Enter a number: 7568
The number 7568 is even.
```

```python
# Create a Python program that accepts a student's score and prints
their grade based on the
# following:
# 90 and above: A
# 80–89: B
# 70–79: C
# 60–69: D
# Below 60: F
# Use if-elif-else statements.
# Validate that the score entered is between 0 and 100.

score = int(input("Enter the student's score (0–100): "))

if 0 <= score <= 100:
    if score >= 90:
        grade = 'A'
    elif score >= 80:
        grade = 'B'
    elif score >= 70:
        grade = 'C'
    elif score >= 60:
        grade = 'D'
    else:
        grade = 'F'

    print(f"The student's grade is: {grade}")
else:
    print("Invalid score! Please enter a value between 0 and 100.")
```

```
Enter the student's score (0–100): 3234
Invalid score! Please enter a value between 0 and 100.
```

```python
# Write a Python program that accepts a string from the user and prints the reversed version of
# the string.
# 1. Use slicing ([::-1]).
# 2. Also display the original string length using len().

text = input("Enter a string: ")
reversed_text = text[::-1]
print(f"Original string length: {len(text)}")
print(f"Reversed string: {reversed_text}")
```

```
Enter a string: cdsfvv
Original string length: 6
Reversed string: vvfsdc
```

```python
# Ask the user to input a string and check whether the string is a palindrome (reads the same
# forwards and backwards).
# 1. Ignore case (use .lower()).
# 2. Use decision-making (if-else) based on comparison.

text = input("Enter a string: ")
normalized = text.lower()
if normalized == normalized[::-1]:
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

```
Enter a string: CAac
The string is a palindrome.
```

```python
# Write a Python program that counts the number of words in a user-entered sentence.
# 1. Use the split() method.
# 2. Print the total word count.

sentence = input("Enter a sentence: ")
words = sentence.split()
print(f"Total number of words: {len(words)}")
```

```
Enter a sentence: fdgd vdfsgdfsg gdsg fd
Total number of words: 4
```

```python
# Create a program that asks the user to input a sentence and a character, then finds and prints:
# 1. How many times that character appears (count() method).
# 2. The first position where the character occurs (find() method).
```

```python
sentence = input("Enter a sentence: ")
char = input("Enter a character to search for: ")

count = sentence.count(char)
position = sentence.find(char)

print(f"The character '{char}' appears {count} time(s).")
if position != -1:
    print(f"The first occurrence is at position {position}.")
else:
    print("The character was not found in the sentence.")
```

```
Enter a sentence: dwed
Enter a character to search for: dwed
The character 'dwed' appears 1 time(s).
The first occurrence is at position 0.
```

```python
# Create a list of 5 numbers entered by the user. Perform the
following:
# 1. Display the list.
# 2. Add a new number to the list.
# 3. Sort the list in ascending order.
# 4. Remove a number entered by the user.
# Use append(), sort(), and remove() methods.

numbers = []
for i in range(5):
    num = int(input(f"Enter number {i + 1}: "))
    numbers.append(num)

print("Original list:", numbers)

new_number = int(input("Enter a new number to add to the list: "))
numbers.append(new_number)
print("List after adding the new number:", numbers)

numbers.sort()
print("Sorted list:", numbers)

remove_number = int(input("Enter a number to remove from the list: "))
if remove_number in numbers:
    numbers.remove(remove_number)
    print("List after removal:", numbers)
else:
    print("The number is not in the list.")
```

```
Enter number 1: 43
Enter number 2: 33
Enter number 3: 22
Enter number 4: 11
Enter number 5: 334
```

```
Original list: [43, 33, 22, 11, 334]
Enter a new number to add to the list: 4325
List after adding the new number: [43, 33, 22, 11, 334, 4325]
Sorted list: [11, 22, 33, 43, 334, 4325]
Enter a number to remove from the list: 343
The number is not in the list.
```

```python
# Create a tuple with 5 user-entered elements and perform:
# 1. Find the maximum and minimum values.
# 2. Find the sum of all elements.
# 3. Print the second and fourth elements separately.
# Use built-in functions like max(), min(), and sum()


elements = []
for i in range(5):
    num = int(input(f"Enter element {i + 1}: "))
    elements.append(num)

data = tuple(elements)

print("Tuple:", data)
print("Maximum value:", max(data))
print("Minimum value:", min(data))
print("Sum of all elements:", sum(data))
print("Second element:", data[1])
print("Fourth element:", data[3])
```

```
Enter element 1: 4324
Enter element 2: 4234
Enter element 3: 343
Enter element 4: 22
Enter element 5: 2
Tuple: (4324, 4234, 343, 22, 2)
Maximum value: 4324
Minimum value: 2
Sum of all elements: 8925
Second element: 4234
Fourth element: 22
```

```python
# Ask the user for 3 keys and their corresponding values to create a
dictionary. Then:
# 1. Display all keys and values separately.
# 2. Access and print the value for a specific key entered by the
user.
# Use keys(), values(), and indexing methods.

my_dict = {}
for i in range(3):
    key = input(f"Enter key {i + 1}: ")
    value = input(f"Enter value for key '{key}': ")
    my_dict[key] = value

print("Keys:", my_dict.keys())
print("Values:", my_dict.values())

specific_key = input("Enter a key to get its value: ")
if specific_key in my_dict:
    print(f"The value for '{specific_key}' is:
{my_dict[specific_key]}")
else:
    print(f"The key '{specific_key}' does not exist in the
dictionary.")
```

```
Enter key 1: 32
Enter value for key '32': fewef
Enter key 2: 3244
Enter value for key '3244': dsf
Enter key 3: 3214
Enter value for key '3214': vds
Keys: dict_keys(['32', '3244', '3214'])
Values: dict_values(['fewef', 'dsf', 'vds'])
Enter a key to get its value: 3244
The value for '3244' is: dsf
```

```
# Create a dictionary to store the student’s name, roll number, and
marks. Allow the user to:
# 1. Update marks.
# 2. Add a new field like 'grade'.
# 3. Delete the roll number field.
# Use dictionary update(), del, and standard dictionary methods.

student_info = {
    "name": input("Enter student's name: "),
    "roll_number": input("Enter student's roll number: "),
    "marks": float(input("Enter student's marks: "))
}

new_marks = float(input("Enter new marks: "))
student_info.update({"marks": new_marks})

grade = input("Enter student's grade: ")
student_info["grade"] = grade

del student_info["roll_number"]
print("Updated student information:", student_info)
```

```
Enter student's name: sia
Enter student's roll number: 33
Enter student's marks: 324
Enter new marks: 5436
Enter student's grade: g
Updated student information: {'name': 'sia', 'marks': 5436.0, 'grade':
'g'}

# Create a 1D array of 5 integers entered by the user. Perform the
following:
# 1. Display the array elements.
# 2. Find and display the sum of all elements.
# 3. Find the maximum element.
# Use array module or lists.
# Use simple loops (for).

import array

arr = array.array('i', [int(input(f"Enter integer {i + 1}: ")) for i
in range(5)])

print("Array elements:", end=" ")
for num in arr:
    print(num, end=" ")

sum_elements = sum(arr)
max_element = max(arr)

print(f"\nSum of all elements: {sum_elements}")
print(f"Maximum element: {max_element}")


Enter integer 1: 423
Enter integer 2: 4324
Enter integer 3: 2
Enter integer 4: 1
Enter integer 5: 1
Array elements: 423 4324 2 1 1
Sum of all elements: 4751
Maximum element: 4324

# Create a 2D array (matrix) of size 3x3 where the user inputs the
elements. Perform the
# following:
# 1. Display the matrix.
# 2. Calculate the sum of each row and each column separately.
# Use nested lists.
# Use nested loops.

matrix = []
for i in range(3):
    row = []
    for j in range(3):
        row.append(int(input(f"Enter element at position ({i+1},
{j+1}): ")))
    matrix.append(row)

print("\nMatrix:")
for row in matrix:
    print(row)

row_sums = []
for i in range(3):
    row_sum = 0
    for j in range(3):
        row_sum += matrix[i][j]
    row_sums.append(row_sum)

col_sums = []
for j in range(3):
    col_sum = 0
    for i in range(3):
        col_sum += matrix[i][j]
    col_sums.append(col_sum)

print("\nRow sums:", row_sums)
print("Column sums:", col_sums)
```

```
Enter element at position (1, 1): 3
Enter element at position (1, 2): 3
Enter element at position (1, 3): 3
Enter element at position (2, 1): 3
Enter element at position (2, 2): 3
Enter element at position (2, 3): 3
Enter element at position (3, 1): 3
Enter element at position (3, 2): 3
Enter element at position (3, 3): 3

Matrix:
[3, 3, 3]
[3, 3, 3]
[3, 3, 3]

Row sums: [9, 9, 9]
Column sums: [9, 9, 9]
```

```
# Write a Python program with a function calculate_average(numbers)
that accepts a list of
# numbers and returns the average.
# 1. Define and call user-defined function.
# 2. Use input from the user

def calculate_average(numbers):
```

```python
    return sum(numbers) / len(numbers)

numbers = list(map(int, input("Enter numbers separated by space: 
").split()))

average = calculate_average(numbers)
print(f"The average of the numbers is: {average}")
```

```
Enter numbers separated by space: 43 43 53 21
The average of the numbers is: 40.0
```

```python
# Create a recursive function to print the Fibonacci series up to n 
terms.
# 1. Define a recursive function fibonacci(n).
# 2. Use if-else for recursion base condition

def fibonacci(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        series = fibonacci(n - 1)
        series.append(series[-1] + series[-2])
        return series

n = int(input("Enter the number of terms in the Fibonacci series: "))

fib_series = fibonacci(n)
print(f"The Fibonacci series up to {n} terms is: {fib_series}")
```

```
Enter the number of terms in the Fibonacci series: 5
The Fibonacci series up to 5 terms is: [0, 1, 1, 2, 3]
```

```python
# Write a Python program to input a list of numbers from the user and 
use map() with a lambda
# function to create a new list containing the squares of the numbers.
# 1. Use map() and lambda.
# 2. Print the original and squared lists.

numbers = list(map(int, input("Enter numbers separated by space: 
").split()))

squared_numbers = list(map(lambda x: x ** 2, numbers))

print(f"Original list: {numbers}")
print(f"Squared list: {squared_numbers}")
```

```
Enter numbers separated by space: 43 4 4 4 2
Original list: [43, 4, 4, 4, 2]
Squared list: [1849, 16, 16, 16, 4]
```

```python
# Create a list of numbers. Use the filter() function and a lambda
function to extract only the
# even numbers into a new list.
# 1. Use filter() and lambda.
# 2. Print both the original and filtered lists

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

even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(f"Original list: {numbers}")
print(f"Filtered even numbers: {even_numbers}")
```

```
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered even numbers: [2, 4, 6, 8, 10]
```

```python
# Ask the user for a list of numbers. Use the reduce() function along
with a lambda function to
# compute the sum of all numbers.
# 1. Import reduce from functools.
# 2. Use lambda inside reduce().

from functools import reduce
numbers = list(map(int, input("Enter numbers separated by space:
").split()))
total_sum = reduce(lambda x, y: x + y, numbers)
print(f"The sum of all numbers is: {total_sum}")
```

```
Enter numbers separated by space: 4 4 4 4
The sum of all numbers is: 16
```

```python
# Write a program to find the maximum number in a list using reduce()
and a lambda function
# (without using built-in max()).
# 1. Use reduce() and lambda.
# 2. Print the maximum number found.

from functools import reduce
numbers = list(map(int, input("Enter numbers separated by space:
").split()))
max_number = reduce(lambda x, y: x if x > y else y, numbers)
print(f"The maximum number in the list is: {max_number}")
```

```
Enter numbers separated by space: 3 4 3 224 2
The maximum number in the list is: 224
```

```python
# Write a Python class Book with attributes title, author, and price.
# 1. Create a constructor to initialize these values.
# 2. Write a method display_details() to display book information.
# Create at least two book objects and display their details.

class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price

    def display_details(self):
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Price: ${self.price}")

book1 = Book("To Kill a Mockingbird", "Harper Lee", 15.99)
book2 = Book("1984", "George Orwell", 12.99)

print("Book 1 Details:")
book1.display_details()
print("\nBook 2 Details:")
book2.display_details()
```

```
Book 1 Details:
Title: To Kill a Mockingbird
Author: Harper Lee
Price: $15.99

Book 2 Details:
Title: 1984
Author: George Orwell
Price: $12.99
```

```python
# Write a class Rectangle with attributes length and breadth.
# 1. Write a constructor to initialize dimensions.
# 2. Write methods to calculate and return the area and perimeter of
the rectangle.
# Create an object and display area and perimeter

class Rectangle:
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length * self.breadth

    def perimeter(self):
        return 2 * (self.length + self.breadth)

rect = Rectangle(5, 3)
```

```
print(f"Area of rectangle: {rect.area()}")
print(f"Perimeter of rectangle: {rect.perimeter()}")
```

```
Area of rectangle: 15
Perimeter of rectangle: 16
```

```
# Create a class BankAccount with attributes account_holder,
account_number, and balance.
# 1. Use a constructor to initialize them.
# 2. Write methods deposit(amount) and withdraw(amount) to update the
balance.
# 3. Also add a method display_balance() to show current balance.
# Create one object and perform deposit and withdrawal operations

class BankAccount:
    def __init__(self, account_holder, account_number, balance=0):
        self.account_holder = account_holder
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: ${amount}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn: ${amount}")
        else:
            print("Insufficient funds")

    def display_balance(self):
        print(f"Current balance: ${self.balance}")

account = BankAccount("John Doe", "123456789", 500)

account.display_balance()
account.deposit(200)
account.withdraw(100)
account.display_balance()
```

```
Current balance: $500
Deposited: $200
Withdrawn: $100
Current balance: $600
```

```
# Create a class Employee with attributes name, employee_id, and
salary.
# 1. Write a constructor to initialize these.
# 2. Add a method show_details() to display employee details.
# 3. Add a method increment_salary() to increase the salary by a given
```

```
percentage.
# Create at least two employee objects and apply salary increment.

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

    def show_details(self):
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Salary: ${self.salary}")

    def increment_salary(self, percentage):
        self.salary += self.salary * (percentage / 100)
        print(f"Salary after {percentage}% increment: ${self.salary}")

employee1 = Employee("Alice", "E001", 50000)
employee2 = Employee("Bob", "E002", 60000)

print("Employee 1 Details:")
employee1.show_details()
employee1.increment_salary(10)

print("\nEmployee 2 Details:")
employee2.show_details()
employee2.increment_salary(15)

Employee 1 Details:
Name: Alice
Employee ID: E001
Salary: $50000
Salary after 10% increment: $55000.0

Employee 2 Details:
Name: Bob
Employee ID: E002
Salary: $60000
Salary after 15% increment: $69000.0

# Create a base class Vehicle with attributes brand and year.
# Create derived classes Car and Bike which add specific attributes
like model for Car and type
# for Bike.
# Write methods to display all details.
# Use constructors and method overriding.

class Vehicle:
    def __init__(self, brand, year):
        self.brand = brand
        self.year = year

    def display_details(self):
        print(f"Brand: {self.brand}")
        print(f"Year: {self.year}")

class Car(Vehicle):
    def __init__(self, brand, year, model):
        super().__init__(brand, year)
        self.model = model

    def display_details(self):
        super().display_details()
        print(f"Model: {self.model}")

class Bike(Vehicle):
    def __init__(self, brand, year, type_):
        super().__init__(brand, year)
        self.type_ = type_

    def display_details(self):
        super().display_details()
        print(f"Type: {self.type_}")

car = Car("Toyota", 2021, "Corolla")
bike = Bike("Yamaha", 2020, "Sport")

print("Car Details:")
car.display_details()

print("\nBike Details:")
bike.display_details()
```

```
Car Details:
Brand: Toyota
Year: 2021
Model: Corolla

Bike Details:
Brand: Yamaha
Year: 2020
Type: Sport
```

```
# Create a base class Shape with a method area().
# Create derived classes Rectangle and Circle that override the area()
method to calculate area
# appropriately.
# Use method overriding (polymorphism)

import math
```

```
class Shape:
    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length * self.breadth

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

rect = Rectangle(5, 3)
circle = Circle(4)

print(f"Area of Rectangle: {rect.area()}")
print(f"Area of Circle: {circle.area()}")
```

```
Area of Rectangle: 15
Area of Circle: 50.26548245743669
```

```
# Create a class Employee with attributes like name and salary.
# Create two subclasses Manager and Developer that inherit from
Employee and have their
# own extra method show_role() that displays their roles.
# Demonstrate polymorphism by calling the same method (show_role())
from different objects

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

class Manager(Employee):
    def __init__(self, name, salary, team_size):
        super().__init__(name, salary)
        self.team_size = team_size

    def show_role(self):
        print(f"{self.name} is a Manager with a team of
{self.team_size} people.")

class Developer(Employee):
    def __init__(self, name, salary, programming_language):
        super().__init__(name, salary)
        self.programming_language = programming_language

    def show_role(self):
        print(f"{self.name} is a Developer skilled in
{self.programming_language}.")

manager = Manager("Alice", 80000, 10)
developer = Developer("Bob", 70000, "Python")

manager.show_role()
developer.show_role()
```

```
Alice is a Manager with a team of 10 people.
Bob is a Developer skilled in Python.
```

```
# Create a base class BankAccount with basic attributes like
account_holder, balance.
# Create two subclasses SavingsAccount and CurrentAccount with methods
specific to their
# type (e.g., interest calculation for savings, overdraft limit for
current).
# Use inheritance and additional methods in subclasses

class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: ${amount}")

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"Withdrawn: ${amount}")
        else:
            print("Insufficient funds")

    def display_balance(self):
        print(f"Current balance: ${self.balance}")

class SavingsAccount(BankAccount):
    def __init__(self, account_holder, balance, interest_rate):
        super().__init__(account_holder, balance)
        self.interest_rate = interest_rate

    def calculate_interest(self):
        interest = self.balance * (self.interest_rate / 100)
        print(f"Interest: ${interest}")
        self.balance += interest
        print(f"New balance after interest: ${self.balance}")

class CurrentAccount(BankAccount):
    def __init__(self, account_holder, balance, overdraft_limit):
        super().__init__(account_holder, balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if self.balance + self.overdraft_limit >= amount:
            self.balance -= amount
            print(f"Withdrawn: ${amount}")
        else:
            print("Exceeded overdraft limit")

    def display_overdraft_limit(self):
        print(f"Overdraft limit: ${self.overdraft_limit}")

savings = SavingsAccount("Alice", 5000, 5)
current = CurrentAccount("Bob", 2000, 1000)

savings.deposit(1000)
savings.calculate_interest()
savings.display_balance()

print()

current.deposit(500)
current.withdraw(2500)
current.display_balance()
current.display_overdraft_limit()

Deposited: $1000
Interest: $300.0
New balance after interest: $6300.0
Current balance: $6300.0

Deposited: $500
Withdrawn: $2500
Current balance: $0
Overdraft limit: $1000

# Write a Python program that:
# 1. Accepts a string input from the user.
# 2. Writes the string into a text file.
# 3. Then reads and displays the contents of the file.
# 4. Handle exceptions like FileNotFoundError and IOError.
# Use open(), write(), and read() functions.
# Include try-except-finally blocks for error handling
```

```
try:
    user_input = input("Enter a string to save in a file: ")

    with open("user_input.txt", "w") as file:
        file.write(user_input)

    with open("user_input.txt", "r") as file:
        content = file.read()
        print("\nContents of the file:")
        print(content)

except FileNotFoundError:
    print("Error: The file was not found.")
except IOError:
    print("Error: There was an issue with reading or writing the
file.")
finally:
    print("\nProgram finished.")
```

```
Enter a string to save in a file: vrfgl;fmgh

Contents of the file:
vrfgl;fmgh

Program finished.
```

```
# Write a Python program that:
# 1. Opens the file in read mode.
# 2. Counts and prints the total number of words in the file.
# 3. Handle any file errors using exception handling.
# Use file reading methods (read() or readlines()).
# Use split() to count words.
# Use try-except to manage errors properly.


# import os
# print("Current Working Directory:", os.getcwd())


try:
    with open("user_input.txt", "r") as file:
        content = file.read()

    words = content.split()
    word_count = len(words)

    print(f"Total number of words in the file: {word_count}")

except FileNotFoundError:
    print("Error: The file was not found.")
except IOError:
    print("Error: There was an issue with reading the file.")
finally:
    print("\nProgram finished.")
```

```
Total number of words in the file: 1

Program finished.
```

```
# Design a GUI application using Tkinter that:
# Accepts two numbers as input.
# Displays their sum when a button is clicked.
# Use Entry, Label, and Button widgets.
# Add basic input validation (e.g., both fields must be filled).

import tkinter as tk
from tkinter import messagebox

def calculate_sum():
    try:
        num1 = float(entry_num1.get())
        num2 = float(entry_num2.get())

        result = num1 + num2

        label_result.config(text=f"Sum: {result}")

    except ValueError:
        messagebox.showerror("Invalid Input", "Please enter valid
numbers in both fields.")

def validate_input():
    if entry_num1.get() == "" or entry_num2.get() == "":
        messagebox.showerror("Input Error", "Please fill both
fields.")
    else:
        calculate_sum()

window = tk.Tk()
window.title("Sum Calculator")

label_num1 = tk.Label(window, text="Enter first number:")
label_num2 = tk.Label(window, text="Enter second number:")
entry_num1 = tk.Entry(window)
entry_num2 = tk.Entry(window)
button_calculate = tk.Button(window, text="Calculate Sum",
command=validate_input)
label_result = tk.Label(window, text="Sum: ")

label_num1.grid(row=0, column=0, padx=10, pady=10)
entry_num1.grid(row=0, column=1, padx=10, pady=10)
label_num2.grid(row=1, column=0, padx=10, pady=10)
```

```python
entry_num2.grid(row=1, column=1, padx=10, pady=10)
button_calculate.grid(row=2, column=0, columnspan=2, pady=10)
label_result.grid(row=3, column=0, columnspan=2, pady=10)

window.mainloop()

# Create a basic login form GUI with:
# 1. Username and password entry fields.
# 2. A "Login" button that shows a message like “Login Successful” on
clicking.
# Use Label, Entry, and Button widgets.
# Display feedback using messagebox

import tkinter as tk
from tkinter import messagebox

def login():
    username = entry_username.get()
    password = entry_password.get()

    if username == "user" and password == "password":
        messagebox.showinfo("Login", "Login Successful!")
    else:
        messagebox.showerror("Login", "Invalid Username or Password!")

window = tk.Tk()
window.title("Login Form")

label_username = tk.Label(window, text="Username:")
label_username.grid(row=0, column=0, padx=10, pady=10)

entry_username = tk.Entry(window)
entry_username.grid(row=0, column=1, padx=10, pady=10)

label_password = tk.Label(window, text="Password:")
label_password.grid(row=1, column=0, padx=10, pady=10)

entry_password = tk.Entry(window, show="*")
entry_password.grid(row=1, column=1, padx=10, pady=10)

button_login = tk.Button(window, text="Login", command=login)
button_login.grid(row=2, column=0, columnspan=2, pady=20)

window.mainloop()

# Create a Python program to:
# 1. Connect to an SQLite database.
# 2. Create a students table (if it doesn’t exist).
# 3. Insert at least 3 student records (name, age, grade).
# 4. Fetch and display all student records.
# Use DDL (CREATE TABLE) and DML (INSERT, SELECT) commands.
# Use sqlite3 module in Python.

import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('school.db')
cursor = conn.cursor()

# Create the students table if it doesn't exist
cursor.execute('''
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER NOT NULL,
    grade TEXT NOT NULL
)
''')

# Insert student records (name, age, grade)
cursor.execute("INSERT INTO students (name, age, grade) VALUES
('Alice', 20, 'A')")
cursor.execute("INSERT INTO students (name, age, grade) VALUES ('Bob',
22, 'B')")
cursor.execute("INSERT INTO students (name, age, grade) VALUES
('Charlie', 21, 'A')")

# Commit the changes to the database
conn.commit()

# Fetch and display all student records
cursor.execute("SELECT * FROM students")
students = cursor.fetchall()

print("Student Records:")
for student in students:
    print(f"ID: {student[0]}, Name: {student[1]}, Age: {student[2]},
Grade: {student[3]}")

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

```
Student Records:
ID: 1, Name: Alice, Age: 20, Grade: A
ID: 2, Name: Bob, Age: 22, Grade: B
ID: 3, Name: Charlie, Age: 21, Grade: A
```

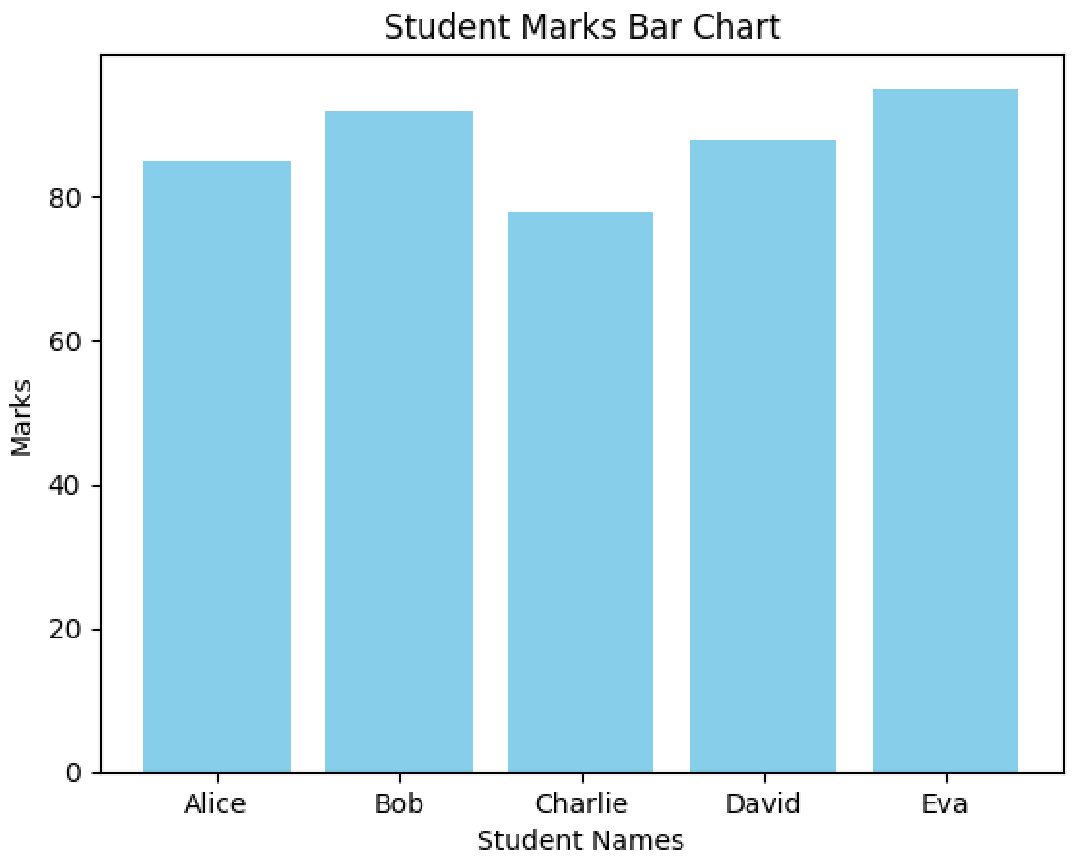
```python
# Write a Python program that:
# 1. Creates an employees table with fields: id, name, salary.
# 2. Inserts at least two employee records.
# 3. Updates the salary of one employee.
# 4. Deletes one employee record by name.
# 5. Displays all remaining records.
# Use DDL (CREATE TABLE) and DML (INSERT, UPDATE, DELETE, SELECT).
# Use sqlite3 with commit and close statements properly

import sqlite3

conn = sqlite3.connect('company.db')
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS employees (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    salary REAL NOT NULL
)
''')

cursor.execute("INSERT INTO employees (name, salary) VALUES ('Alice',
50000)")
cursor.execute("INSERT INTO employees (name, salary) VALUES ('Bob',
45000)")

conn.commit()

cursor.execute("update employees SET salary = 55000 WHERE name =
'Bob'")
conn.commit()

cursor.execute("DELETE FROM employees WHERE name = 'Alice'")
conn.commit()

cursor.execute("SELECT * FROM employees")
employees = cursor.fetchall()

for employee in employees:
    print(f"ID: {employee[0]}, Name: {employee[1]}, Salary:
{employee[2]}")

conn.close()
```

```
ID: 2, Name: Bob, Salary: 55000.0
```

```
# Write a program that:
# 1. Creates a Pandas DataFrame with names and marks of 5 students.
# 2. Uses NumPy to calculate the average, maximum, and minimum marks.
# 3. Uses Matplotlib to display a bar chart of student names vs.
marks.
# Use pandas.DataFrame, numpy.mean(), and matplotlib.pyplot.bar().

import pandas as pd
```

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

data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Marks': [85, 92, 78, 88, 95]
}

df = pd.DataFrame(data)

average_marks = np.mean(df['Marks'])
max_marks = np.max(df['Marks'])
min_marks = np.min(df['Marks'])

print(f"Average Marks: {average_marks}")
print(f"Maximum Marks: {max_marks}")
print(f"Minimum Marks: {min_marks}")

plt.bar(df['Name'], df['Marks'], color='skyblue')
plt.xlabel('Student Names')
plt.ylabel('Marks')
plt.title('Student Marks Bar Chart')
plt.show()
```

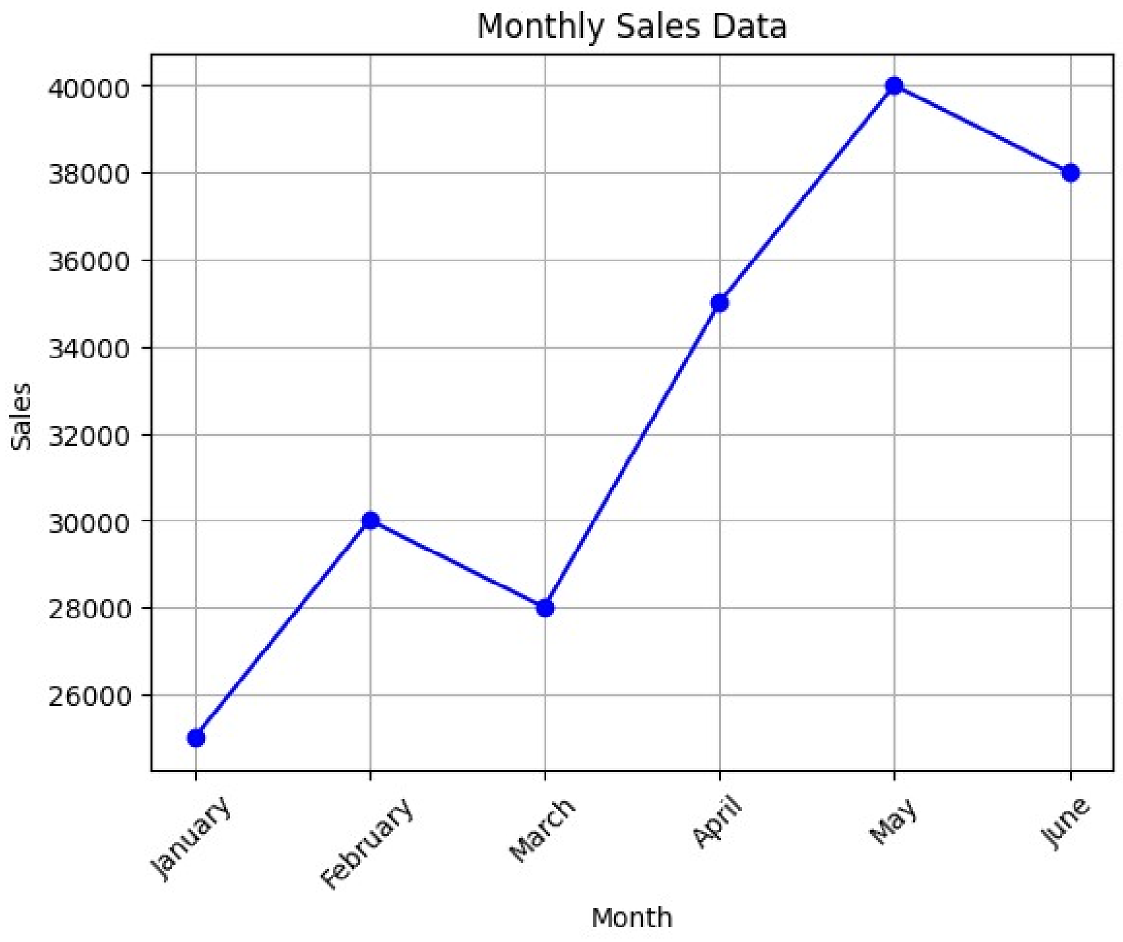
```
Average Marks: 87.6
Maximum Marks: 95
Minimum Marks: 78
```



```python
# Create a program that:
# 1. Loads monthly sales data into a Pandas DataFrame (e.g., 6
months).
# 2. Calculates the total and average sales using NumPy.
# 3. Plots a line graph of month vs. sales using Matplotlib.
# Use pandas, numpy, and matplotlib.pyplot.plot().

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = {
    'Month': ['January', 'February', 'March', 'April', 'May', 'June'],
    'Sales': [25000, 30000, 28000, 35000, 40000, 38000]
}

df = pd.DataFrame(data)

total_sales = np.sum(df['Sales'])
average_sales = np.mean(df['Sales'])

print(f"Total Sales: {total_sales}")
print(f"Average Sales: {average_sales}")
```

```
plt.plot(df['Month'], df['Sales'], marker='o', color='b')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.title('Monthly Sales Data')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
Total Sales: 196000
Average Sales: 32666.666666666668
```


Monthly Sales Data
Sales
January
February
March
April
May
June
Month