1.

1.1Create a simple calculator in Python.

def add(x, y):

    return x + y

def subtract(x, y):

    return x - y

def multiply(x, y):

    return x \* y

def divide(x, y):

    if y == 0:

        return "Error: Division by zero!"

    else:

        return x / y

def calculator():

    print("Welcome to Simple Calculator!")

    print("Select operation:")

    print("1. Add")

    print("2. Subtract")

    print("3. Multiply")

    print("4. Divide")

    choice = input("Enter choice (1/2/3/4): ")

    num1 = float(input("Enter first number: "))

    num2 = float(input("Enter second number: "))

    if choice == '1':

        print(num1, "+", num2, "=", add(num1, num2))

    elif choice == '2':

        print(num1, "-", num2, "=", subtract(num1, num2))

    elif choice == '3':

        print(num1, "\*", num2, "=", multiply(num1, num2))

    elif choice == '4':

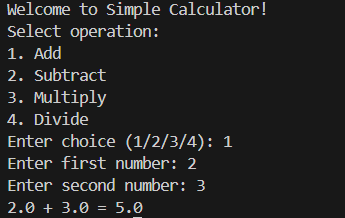
        print(num1, "/", num2, "=", divide(num1, num2))

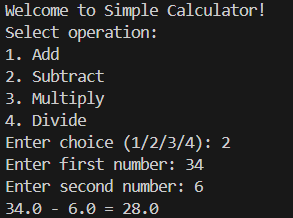
    else:

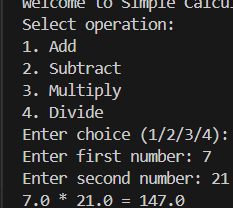
        print("Invalid input")

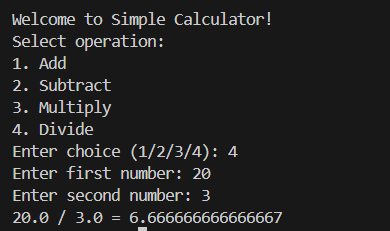
calculator()

**OUTPUT**

****

****





1.2

An electric power distribution company charges domestic customers as

follows: Consumption unit Rate of charge:

1.2.1. 0-200 Rs. 0.50 per unit

1.2.2. 201-400 Rs. 0.65 per unit in excess of 200

1.2.3. 401-600 Rs 0.80 per unit excess of 400

1.2.4. 601 and above Rs 1.00per unit excess of 600

1.2.5. If the bill exceeds Rs. 400, then a surcharge of 15% will be charged,

and the minimum bill should be Rs. 100/-

Create a Python program based on the scenario mentioned above.

def calculate\_electricity\_bill(units):

    if units <= 0:

        return "Invalid input. Units cannot be zero or negative."

    total\_bill = 0

    if units <= 200:

        total\_bill = units \* 0.50

    elif units <= 400:

        total\_bill = 200 \* 0.50 + (units - 200) \* 0.65

    elif units <= 600:

        total\_bill = 200 \* 0.50 + 200 \* 0.65 + (units - 400) \* 0.80

    else:

        total\_bill = 200 \* 0.50 + 200 \* 0.65 + 200 \* 0.80 + (units - 600) \* 1.00

    if total\_bill > 400:

        surcharge = total\_bill \* 0.15

        total\_bill += surcharge

    if total\_bill < 100:

        total\_bill = 100

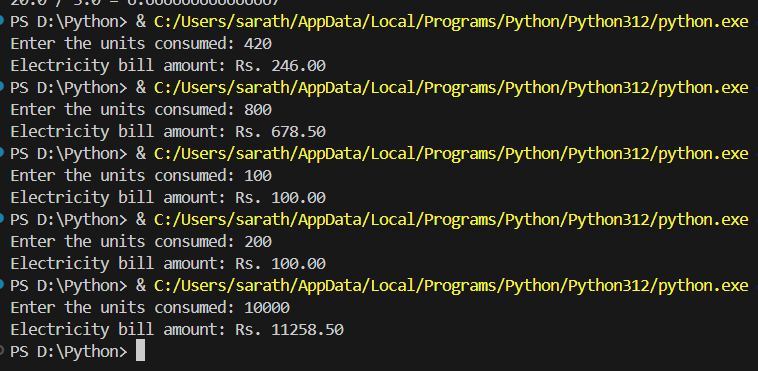
    return total\_bill

units\_consumed = int(input("Enter the units consumed: "))

bill\_amount = calculate\_electricity\_bill(units\_consumed)

print(f"Electricity bill amount: Rs. {bill\_amount:.2f}")

OUTPUT



1.3. Print the pyramid of numbers using for loops.

def pyramid(rows):

    for i in range(1, rows + 1):

        for j in range(1, rows - i + 1):

            print(" ", end=" ")

        for j in range(1, i + 1):

            print(j, end=" ")

        for j in range(i - 1, 0, -1):

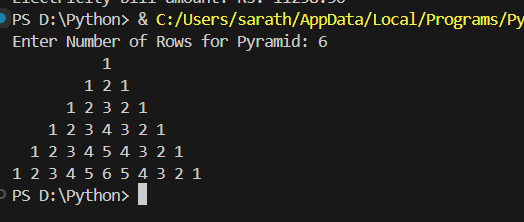
            print(j, end=" ")

        print()

rows = int(input("Enter Number of Rows for Pyramid: "))

pyramid(rows)

**OUTPUT**

****

**1.4. Write a program to find the number and sum of all integers greater than 100**

**and less than 200 that are divisible by 7.**

count = 0

total\_sum = 0

for num in range(101, 200):

    if num % 7 == 0:

        count += 1

        total\_sum += num

print("The integers greater than 100 and less than 200 that are divisible by 7 are:")

for num in range(101, 200):

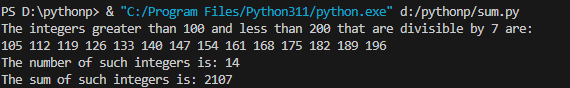
    if num % 7 == 0:

        print(num, end=" ")

print(f"\nThe number of such integers is: {count}")

print(f"The sum of such integers is: {total\_sum}")

**OUTPUT**



**1.5. Write a recursive function to calculate the sum of numbers from 0 to 10**

def sum\_numbers(n):

    if n == 0:

        return 0

    else:

        return n + sum\_numbers(n - 1)

result = sum\_numbers(10)

print(f"The sum of numbers from 0 to 10 is: {result}")

**OUTPUT**

****

1.6. Write a Python program to reverse the digits of a given number and add them

to the original. If the sum is not a palindrome, repeat this procedure.

def reverse\_digits(number):

    str\_number = str(number)

    reversed\_number = int(str\_number[::-1])

    return reversed\_number

def is\_palindrome(number):

    str\_number = str(number)

    return str\_number == str\_number[::-1]

def reverse\_and\_add\_until\_palindrome(number):

    original\_number = number

    while True:

        reversed\_number = reverse\_digits(original\_number)

        sum\_result = original\_number + reversed\_number

        if is\_palindrome(sum\_result):

            return sum\_result, reversed\_number

        original\_number = sum\_result

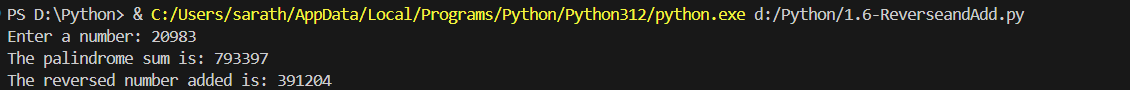
number = int(input("Enter a number: "))

palindrome\_sum, reversed\_number = reverse\_and\_add\_until\_palindrome(number)

print(f"The palindrome sum is: {palindrome\_sum}")

print(f"The reversed number added is: {reversed\_number}")

**OUTPUT:**

****

1.7. Write a menu-driven program that performs the following operations on

strings

1.7.1. Check if the String is a Substring of Another String

1.7.2. Count Occurrences of Character

1.7.3. Replace a substring with another substring

1.7.4. Convert to Capital Letters

def check\_substring():

    string1 = input("Enter the main string: ")

    string2 = input("Enter the substring to check: ")

    if string2 in string1:

        print(f"'{string2}' is a substring of '{string1}'")

    else:

        print(f"'{string2}' is not a substring of '{string1}'")

def count\_occurrences():

    string = input("Enter the string: ")

    char = input("Enter the character to count: ")

    count = string.count(char)

    print(f"Occurrences of '{char}' in '{string}': {count}")

def replace\_substring():

    string = input("Enter the main string: ")

    old\_substring = input("Enter the substring to replace: ")

    new\_substring = input("Enter the new substring: ")

    new\_string = string.replace(old\_substring, new\_substring)

    print(f"Modified string: {new\_string}")

def convert\_to\_capital():

    string = input("Enter the string to convert to capital letters: ")

    capitalized\_string = string.upper()

    print(f"String in capital letters: {capitalized\_string}")

def main():

    while True:

        print("\nOperations on Strings:")

        print("1. Check if the String is a Substring of Another String")

        print("2. Count Occurrences of Character")

        print("3. Replace a substring with another substring")

        print("4. Convert to Capital Letters")

        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '1':

            check\_substring()

        elif choice == '2':

            count\_occurrences()

        elif choice == '3':

            replace\_substring()

        elif choice == '4':

            convert\_to\_capital()

        elif choice == '5':

            print("Exiting the program...")

            break

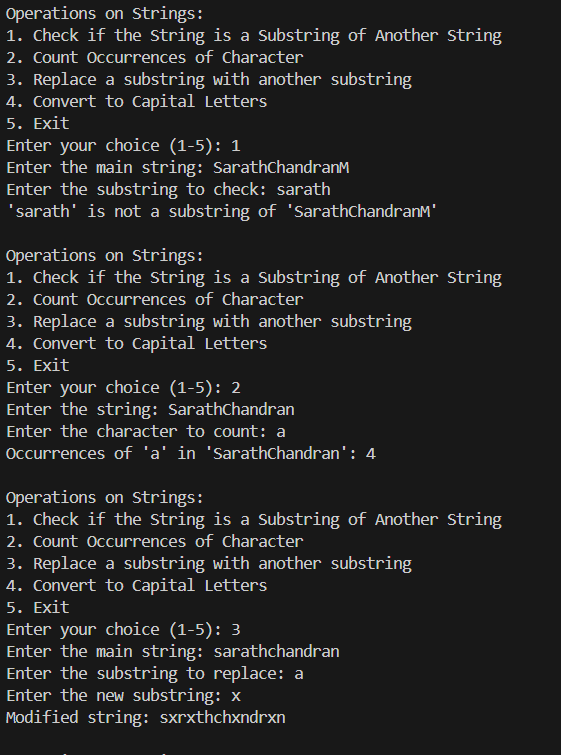
        else:

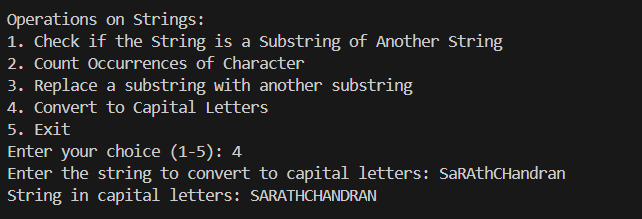
            print("Invalid choice. Please enter a number from 1 to 5.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**OUTPUT:**

****

****

1.8 Write a function to find the factorial of a number but also store the factorials

calculated in a dictionary.

def factorial\_with\_storage(n):

    if n < 0:

        return None

    factorial\_dict = {}

    def factorial(num):

        if num in factorial\_dict:

            return factorial\_dict[num]

        elif num == 0 or num == 1:

            factorial\_dict[num] = 1

            return 1

        else:

            result = num \* factorial(num - 1)

            factorial\_dict[num] = result

            return result

    result = factorial(n)

    print(f"The factorial of {n} is: {result}")

    print("Factorials stored in dictionary:")

    for num, fact in factorial\_dict.items():

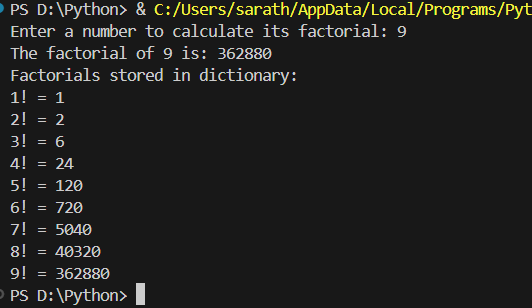
        print(f"{num}! = {fact}")

    return result

number = int(input("Enter a number to calculate its factorial: "))

factorial\_result = factorial\_with\_storage(number)

**OUTPUT:**

****

1.9. Perform various set operations

1.9.1. Set Union

1.9.2. Set Intersection

1.9.3. Set Difference

def set\_operations():

    elements\_A = input("Enter elements of set A separated by spaces: ").split()

    set\_A = set(elements\_A)

    elements\_B = input("Enter elements of set B separated by spaces: ").split()

    set\_B = set(elements\_B)

    union\_set = set\_A.union(set\_B)

    intersection\_set = set\_A.intersection(set\_B)

    difference\_AB = set\_A.difference(set\_B)

    difference\_BA = set\_B.difference(set\_A)

    print(f"\nSet A: {set\_A}")

    print(f"Set B: {set\_B}")

    print(f"\n1. Set Union (A union B): {union\_set}")

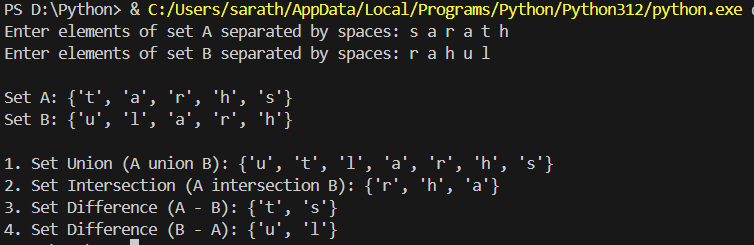
    print(f"2. Set Intersection (A intersection B): {intersection\_set}")

    print(f"3. Set Difference (A - B): {difference\_AB}")

    print(f"4. Set Difference (B - A): {difference\_BA}")

set\_operations()

**OUTPUT:**

****

1.10. Create a dictionary to store the name, roll\_no, and total\_mark of N students.

Now print the details of the student with the highest total\_mark.

# Number of students

N = int(input("Enter the number of students: "))

# Dictionary to store student details

students = {}

# Input student details

for \_ in range(N):

    name = input("Enter student's name: ")

    roll\_no = input("Enter student's roll number: ")

    total\_mark = float(input("Enter student's total mark: "))

    students[roll\_no] = {'name': name, 'total\_mark': total\_mark}

# Find the student with the highest total mark

highest\_mark\_student = max(students.values(), key=lambda x: x['total\_mark'])

# Print details of the student with the highest total mark

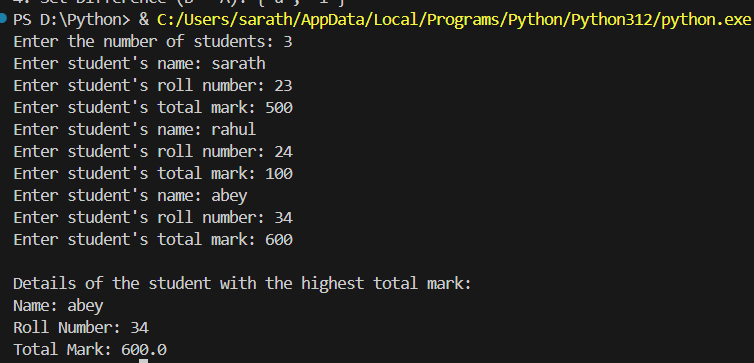
print("\nDetails of the student with the highest total mark:")

print(f"Name: {highest\_mark\_student['name']}")

print(f"Roll Number: {roll\_no}")

print(f"Total Mark: {highest\_mark\_student['total\_mark']}")

**OUTPUT:**



1.11. Write a Python program to copy the contents of a file into another file, line by

line.

def copy\_file(source\_file, destination\_file):

    try:

        with open(source\_file, 'r') as src\_file:

            with open(destination\_file, 'w') as dest\_file:

                for line in src\_file:

                    dest\_file.write(line)

        print(f"Contents from '{source\_file}' copied to '{destination\_file}' successfully.")

    except FileNotFoundError:

        print(f"Error: One or both files not found.")

    except IOError as e:

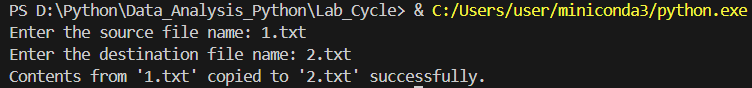
        print(f"Error: {e}")

source\_file = input("Enter the source file name: ")

destination\_file = input("Enter the destination file name: ")

copy\_file(source\_file, destination\_file)

**OUTPUT:**

****

1.12. Use the OS module to perform

1.12.1. Create a directory

1.12.2. Directory Listing

1.12.3. Search for “.py” files

1.12.4. Remove a particular file

import os

def create\_directory(directory\_name):

    try:

        os.mkdir(directory\_name)

        print(f"Directory '{directory\_name}' created successfully.")

    except FileExistsError:

        print(f"Error: Directory '{directory\_name}' already exists.")

    except OSError as e:

        print(f"Error creating directory '{directory\_name}': {e}")

def directory\_listing(directory\_path):

    try:

        files = os.listdir(directory\_path)

        print(f"\nListing files and directories in '{directory\_path}':")

        for file in files:

            print(file)

    except FileNotFoundError:

        print(f"Error: Directory '{directory\_path}' not found.")

    except OSError as e:

        print(f"Error listing directory '{directory\_path}': {e}")

def search\_py\_files(directory\_path):

    try:

        print(f"\nSearching for .py files in '{directory\_path}':")

        for root, dirs, files in os.walk(directory\_path):

            for file in files:

                if file.endswith(".py"):

                    print(os.path.join(root, file))

    except FileNotFoundError:

        print(f"Error: Directory '{directory\_path}' not found.")

    except OSError as e:

        print(f"Error searching files in '{directory\_path}': {e}")

def remove\_file(file\_path):

    try:

        os.remove(file\_path)

        print(f"File '{file\_path}' removed successfully.")

    except FileNotFoundError:

        print(f"Error: File '{file\_path}' not found.")

    except OSError as e:

        print(f"Error removing file '{file\_path}': {e}")

#

if \_\_name\_\_ == "\_\_main\_\_":

    create\_directory("test\_directory")

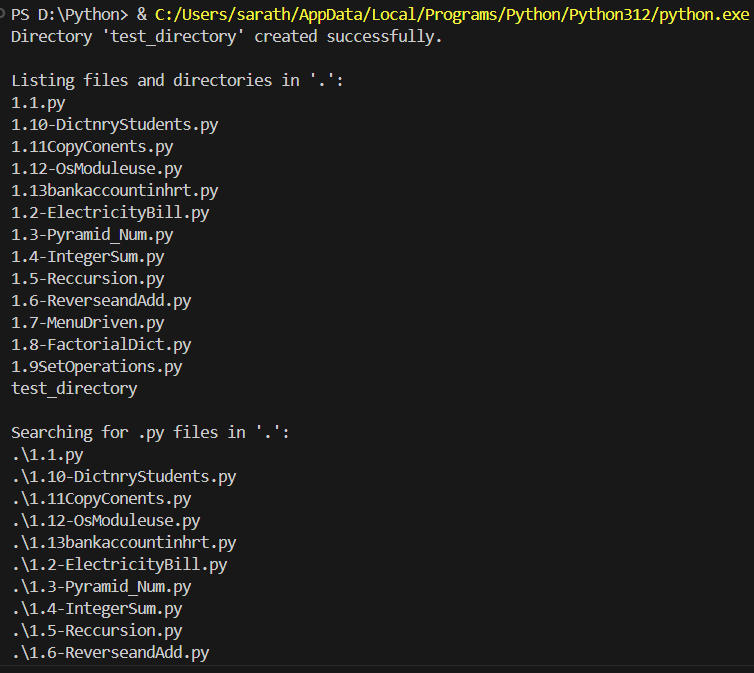
    directory\_listing(".")

    search\_py\_files(".")

    file\_to\_remove = input("\nEnter the file path to remove: ")

    remove\_file(file\_to\_remove)

**OUTPUT:**

****

1.13. Create a simple banking application by using inheritance.

class Account:

    def \_\_init\_\_(self, account\_number, account\_holder, balance=0.0):

        self.account\_number = account\_number

        self.account\_holder = account\_holder

        self.balance = balance

    def deposit(self, amount):

        if amount > 0:

            self.balance += amount

            print(f"Deposited {amount}. New balance is {self.balance}.")

        else:

            print("Deposit amount should be greater than zero.")

    def withdraw(self, amount):

        if 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew {amount}. New balance is {self.balance}.")

        else:

            print("Insufficient funds or invalid withdrawal amount.")

    def display\_balance(self):

        print(f"Account Number: {self.account\_number}")

        print(f"Account Holder: {self.account\_holder}")

        print(f"Current Balance: {self.balance}")

class SavingsAccount(Account):

    def \_\_init\_\_(self, account\_number, account\_holder, balance=0.0, interest\_rate=0.05):

        super().\_\_init\_\_(account\_number, account\_holder, balance)

        self.interest\_rate = interest\_rate

    def apply\_interest(self):

        interest\_amount = self.balance \* self.interest\_rate

        self.balance += interest\_amount

        print(f"Interest applied. New balance is {self.balance}.")

class CurrentAccount(Account):

    def \_\_init\_\_(self, account\_number, account\_holder, balance=0.0, overdraft\_limit=1000.0):

        super().\_\_init\_\_(account\_number, account\_holder, balance)

        self.overdraft\_limit = overdraft\_limit

    def withdraw(self, amount):

        available\_balance = self.balance + self.overdraft\_limit

        if 0 < amount <= available\_balance:

            self.balance -= amount

            print(f"Withdrew {amount}. New balance is {self.balance}.")

        else:

            print("Insufficient funds or overdraw limit exceeded.")

if \_\_name\_\_ == "\_\_main\_\_":

    savings = SavingsAccount("SA001", "Sarath", balance=5000.0)

    savings.display\_balance()

    savings.deposit(1000.0)

    savings.apply\_interest()

    savings.withdraw(2000.0)

    savings.display\_balance()

    current = CurrentAccount("CA001", "Sharrisa", balance=3000.0)

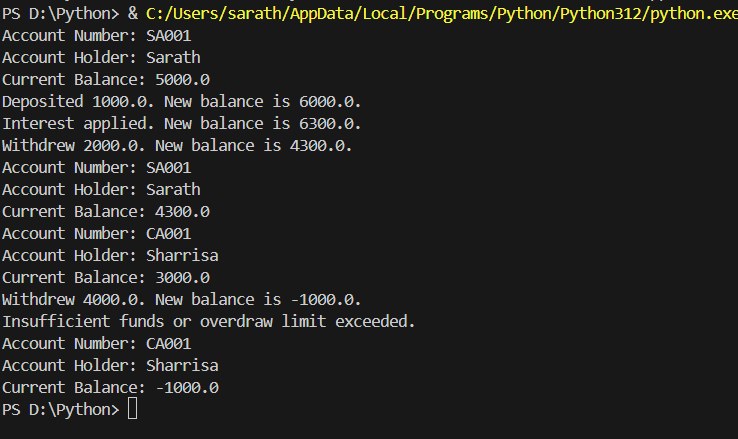
    current.display\_balance()

    current.withdraw(4000.0)

    current.withdraw(2000.0)

    current.display\_balance()

**OUTPUT:**

****