**PRACTICAL-1**

# AIM : A program that models a bank account, with classes for the account, the customer, and the bank

**Problem Statement:**

Create a program that models a bank account system. The system should consist of classes representing the bank account, the customer, and the bank itself.

**Program Description:**

The program will simulate basic banking operations such as creating a new account, depositing and withdrawing money, checking the account balance, and managing customer information. It will use object-oriented programming principles with classes to organize and structure the data and behavior.

**Algorithm:**

1. Define a **Customer** class with attributes like customer ID, name, address, and contact details.
2. Create a **BankAccount** class with attributes such as account number, account holder (linked to a **Customer**), balance, and account type (e.g., savings or checking).
3. Implement methods in the **BankAccount** class for deposit, withdrawal, and checking the account balance.
4. Develop a **Bank** class to manage a collection of accounts, allowing the creation of new accounts, retrieval of account details, and overall management of the bank.

**Source Code:**

import random

class Customer:

def \_\_init\_\_(self, name, address, contact\_number):

self.name = name

self.address = address

self.contact\_number = contact\_number

self.accounts = []

def create\_account(self, account\_type, initial\_balance):

account\_number = Bank.generate\_account\_number()

account = BankAccount(account\_type, initial\_balance, self, account\_number)

self.accounts.append(account)

return account

def display\_customer\_info(self):

print(f"Customer Name: {self.name}")

print(f"Address: {self.address}")

print(f"Contact Number: {self.contact\_number}")

print("Accounts:")

for account in self.accounts:

print(f" - {account}")

class BankAccount:

def \_\_init\_\_(self, account\_type, balance, owner, account\_number):

self.account\_type = account\_type

self.balance = balance

self.owner = owner

self.account\_number = account\_number

def deposit(self, amount):

self.balance += amount

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

def withdraw(self, amount):

if amount <= self.balance:

self.balance -= amount

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

else:

print("Insufficient funds!")

def \_\_str\_\_(self):

return f"{self.account\_type} Account - Account Number: {self.account\_number}, Balance: INR {self.balance}"

class Bank:

def \_\_init\_\_(self, name):

self.name = name

self.customers = []

def add\_customer(self, customer):

self.customers.append(customer)

@staticmethod

def generate\_account\_number():

return ''.join(random.choice('0123456789') for \_ in range(8))

def display\_bank\_info(self):

print(f"Bank Name: {self.name}")

print("Customers:")

for customer in self.customers:

customer.display\_customer\_info()

print()

def find\_account\_by\_number(self, account\_number):

for customer in self.customers:

for account in customer.accounts:

if account.account\_number == account\_number:

return account

return None

# Example usage

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

# Create a bank

my\_bank = Bank("My Bank")

customer\_list=[]

while True:

print("1. New Customer 2. Existing Customer 3. Find Customers info 4.Exit")

try:

choice = int(input())

if choice==1:

print("Customer Registration: \n")

# Create a customer

name=input("Enter Customer Name:")

address=input('Enter Customer Address: ')

contact\_number=input("Enter Customer Contact Number: ")

customer\_obj = Customer(name, address, contact\_number)

customer\_list.append(customer\_obj)

my\_bank.add\_customer(customer\_obj)

while True:

acc\_type = int(input("Enter 1. To create Saving account 2. To Create Cheking account 3. Exit\n"))

if acc\_type == 1:

new\_account = customer\_obj.create\_account("Savings", 1000)

print(f"Savings account created with account number: {new\_account.account\_number}\n")

break

elif acc\_type == 2:

new\_account = customer\_obj.create\_account("Current", 1000)

print(f"Current account created with account number: {new\_account.account\_number}\n")

break

elif acc\_type == 3:

break

else:

print("Invalid option....Try again")

if choice==2:

# User input for transactions

account\_number\_input = input("Enter your account number: ")

account\_to\_transact = my\_bank.find\_account\_by\_number(account\_number\_input)

if account\_to\_transact:

print(f"\nWelcome, {account\_to\_transact.owner.name}!")

print(account\_to\_transact)

while True:

print("1. Enter 1 to deposit\n2. Enter 2 to Withdrawl\n3. Enter 3 to Check the Balance\n4. Exit")

option=int(input("Enter your Option:\n"))

if option==1:

print("Welcome to Deposit Section\n")

# Deposit

deposit\_amount = int(input("\nEnter the amount to deposit: INR "))

account\_to\_transact.deposit(deposit\_amount)

elif option==2:

print("Welcome to withdrawl section:\n")

# Withdrawal

withdrawal\_amount = int(input("\nEnter the amount to withdraw: INR "))

account\_to\_transact.withdraw(withdrawal\_amount)

elif option==3:

# Display updated account information

print("\nUpdated Account Information:")

print(account\_to\_transact)

elif option==4:

break

else:

print("Invalid Option")

else:

print("Account not found.")

if choice==3:

my\_bank.display\_bank\_info()

elif choice==4:

break

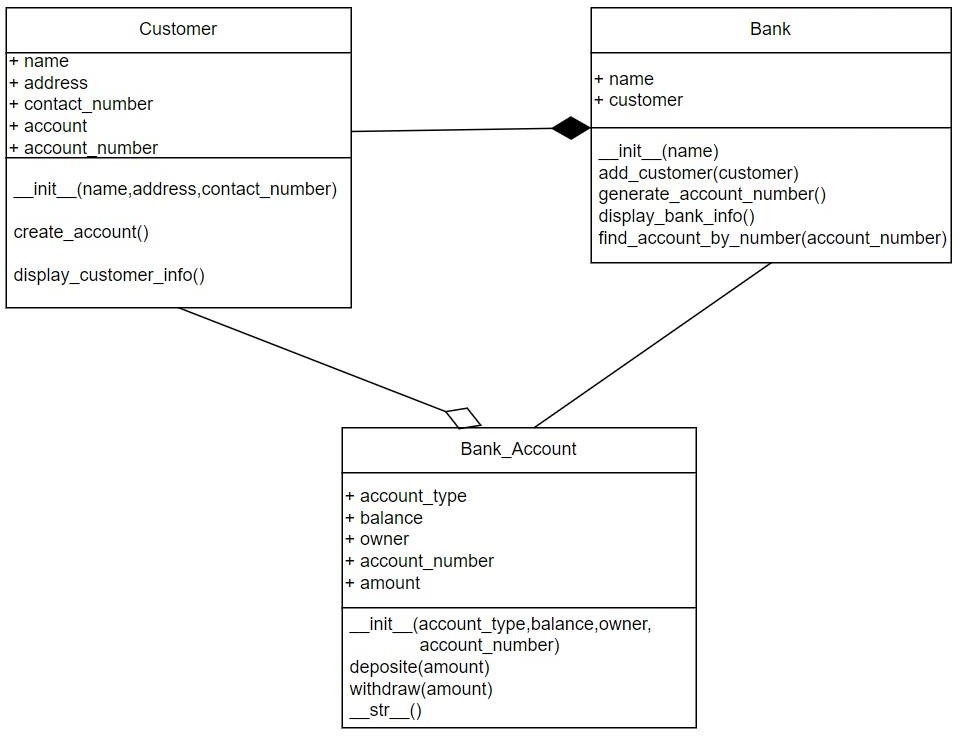
else:

pass

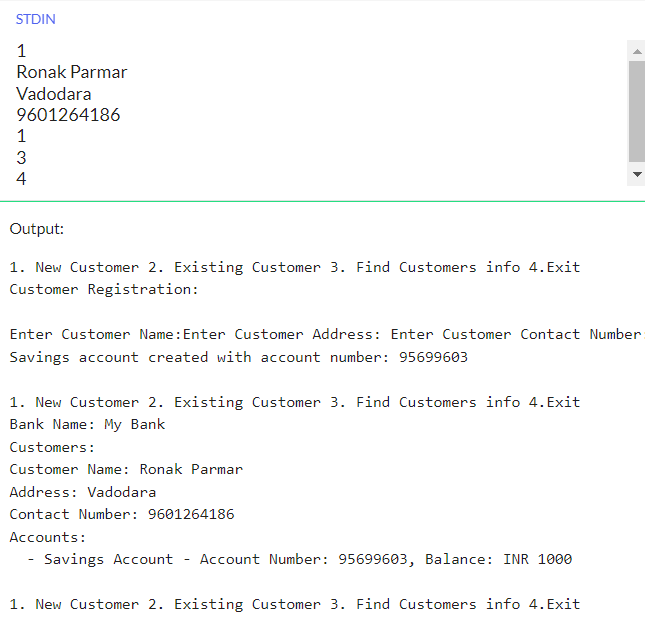
except ValueError:

print("Invalid input. Please enter a valid option.")

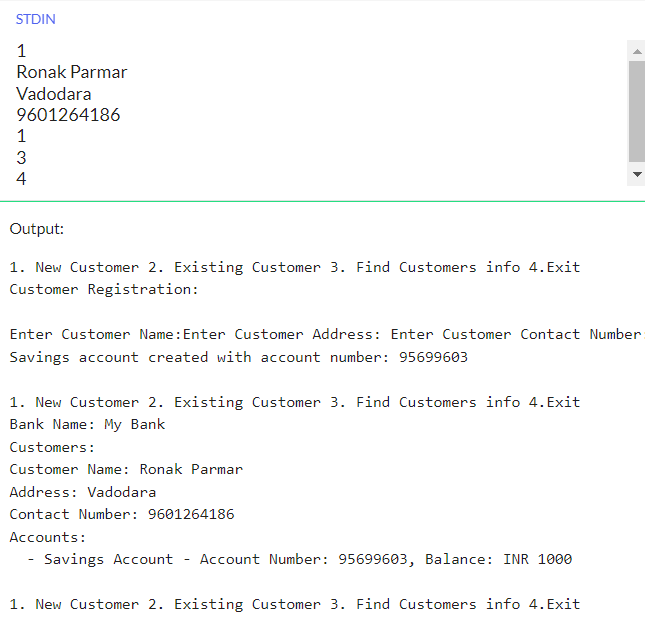
continue

**Class Diagram:**

**Expected Output:**

****

**Actual Output:**

****

**Result:**

The result will be a program that allows users to create bank accounts, perform transactions, and manage customer and account information.

**PRACTICAL - 2**

# AIM: A program that simulates a school management system, with classes for the students, the teachers, and the courses.

**Problem Statement:**

Develop a program that simulates a school management system, modeling classes for students, teachers, and courses. The system should allow for the creation of students and teachers, enrollment in courses, grading, and tracking of academic information.

**Program Description:**

The program will use object-oriented programming to represent students, teachers, and courses. Students can enroll in courses, teachers can assign grades, and the system will maintain academic records for each student.

**Algorithm:**

1. Define a **Student** class with attributes like student ID, name, address, contact details, and a method to enroll in courses.
2. Create a **Teacher** class with attributes such as teacher ID, name, subject expertise, and a method to assign grades to students.
3. Implement a **Course** class with attributes like course code, course name, and a list of enrolled students.
4. Develop a **SchoolManagementSystem** class to manage students, teachers, and courses, allowing the creation of new students and teachers, enrollment in courses, and grading.

**Source Code:**

class Student:

def \_\_init\_\_(self, student\_id, name, grade):

self.student\_id = student\_id

self.name = name

self.grade = grade

def display\_info(self):

print(f"\nStudent ID: {self.student\_id}, Name: {self.name}, Grade: {self.grade}")

class Teacher:

def \_\_init\_\_(self, teacher\_id, name, subject):

self.teacher\_id = teacher\_id

self.name = name

self.subject = subject

def display\_info(self):

print(f"\nTeacher ID: {self.teacher\_id}, Name: {self.name}, Subject: {self.subject}")

class Course:

def \_\_init\_\_(self, course\_code, course\_name, teacher, students):

self.course\_code = course\_code

self.course\_name = course\_name

self.teacher = teacher

self.students = students

def display\_info(self):

print(f"\nCourse Code: {self.course\_code}, Course Name: {self.course\_name}")

print("\nTeacher:")

self.teacher.display\_info()

print("\nStudents:")

for student in self.students:

student.display\_info()

def main():

students = []

teachers = []

courses = []

print("""1.Student\_form/details

2.Teacher\_form/details

3.Course\_form/details""")

cho = int(input("\nEnter your choice: "))

if cho == 1:

num\_students = int(input("\nEnter the number of students: "))

for i in range(num\_students):

student\_id = input(f"\nEnter student {i + 1} ID: ")

name = input(f"\nEnter student {i + 1} name: ")

grade = input(f"\nEnter student {i + 1} grade: ")

students.append(Student(student\_id, name, grade))

print("\nRegistration successful.")

elif cho == 2:

num\_teachers = int(input("\nEnter the number of teachers: "))

for i in range(num\_teachers):

teacher\_id = input(f"\nEnter teacher {i + 1} ID: ")

name = input(f"\nEnter teacher {i + 1} name: ")

subject = input(f"\nEnter teacher {i + 1} subject: ")

teachers.append(Teacher(teacher\_id, name, subject))

print("\nRegistration successful.")

elif cho == 3:

num\_courses = int(input("\nEnter the number of courses: "))

for i in range(num\_courses):

course\_code = input(f"\nEnter course {i + 1} code: ")

course\_name = input(f"\nEnter course {i + 1} name: ")

teacher\_index = int(input("\nEnter the index of the teacher for this course: "))

teacher = teachers[teacher\_index]

student\_indices = input("\nEnter the indices of students for this course (comma-separated): ")

student\_indices = student\_indices.split(",")

students\_for\_course = [students[int(index)] for index in student\_indices]

courses.append(Course(course\_code, course\_name, teacher, students\_for\_course))

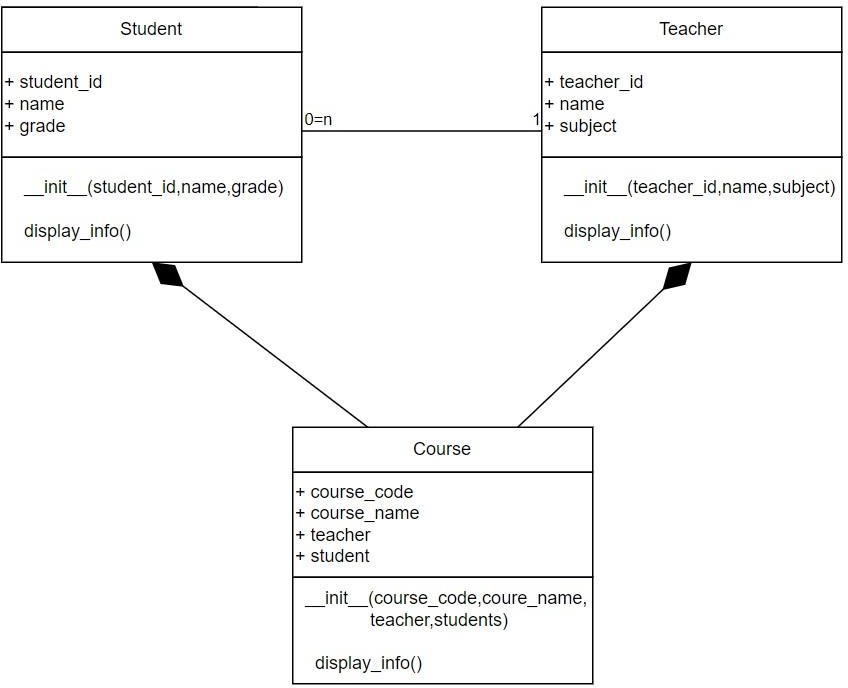
print("\nRegistration successful.")

else:

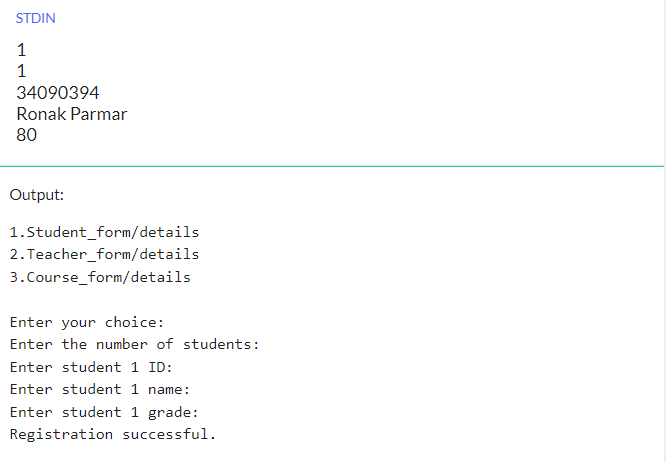
print("\nInvalid input")

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

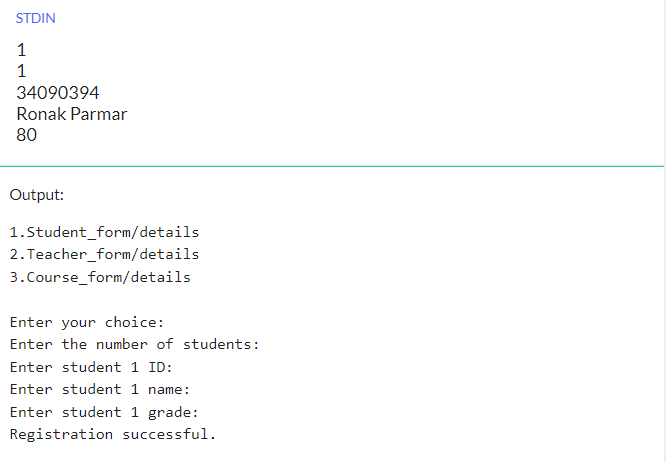
main()

**Class Diagram:**

**Expected Output:**



**Actual Output:**



**Result:**

The program successfully calculates the area and perimeter of a rectangle based on the user's choice. It provides a user-friendly interface and accurate results for the calculations, meeting the goal of creating a rectangle calculator.

**PRACTICAL - 3**

# AIM: A program that reads a text file and counts the number of words in it.

**Problem Statement:**

Develop a program that reads a text file and counts the number of words in it.

**Program Description:**

The program will take a text file as input, read its content, and count the number of words. A word is defined as any sequence of characters separated by whitespace.

**Algorithm:**

1. Open the text file in read mode.
2. Read the content of the file.
3. Tokenize the content based on whitespace to extract words.
4. Count the number of words.
5. Print the result.

**Source Code:**

def count(path):

try:

with open(path,'r') as file:

file\_content = file.read()

return f"data = {file\_content.split()}\nlength of the words: {len(file\_content.split())}"

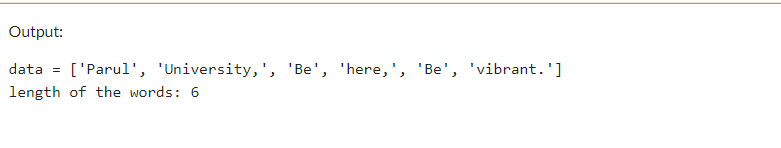
except FileNotFoundError:

return "Please Provide valid file path."

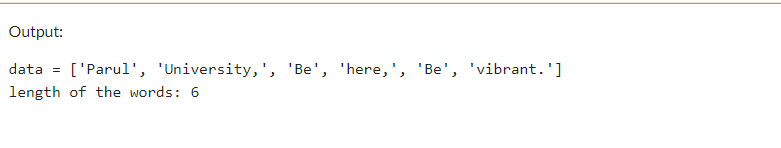
path ="example.txt"

print(count(path))

**Expected Output:**

****

**Actual Output:**

****

**Result:**

The result will be the count of words in the specified text file.

**PRACTICAL - 4**

# AIM: A program that reads a CSV file and calculates the average of the values in a specified column.

**Problem Statement:**

Develop a program that reads a CSV file, extracts data from a specified column, and calculates the average of the values in that column.

**Program Description:**

The program will take a CSV file as input, read its content, and allow the user to specify a column for which the average should be calculated. It will then perform the necessary calculations and display the average value.

**Algorithm:**

1. Accept the CSV file path and column name from the user.
2. Open and read the CSV file, extracting the specified column data.
3. Convert the column values to numerical format (assuming the values are numerical).
4. Calculate the average of the values in the specified column.
5. Display the result.

**Source Code:**

import csv

def calculate\_average(csv\_file, column\_name):

try:

with open(csv\_file, 'r') as file:

reader = csv.DictReader(file)

if column\_name not in reader.fieldnames:

print(f"Column '{column\_name}' not found in the CSV file.")

return None

total = 0

count = 0

for row in reader:

try:

value = float(row[column\_name])

total += value

count += 1

except ValueError:

print(f"Skipping row {reader.line\_num}: Invalid value in column '{column\_name}'.")

if count == 0:

print(f"No valid values found in column '{column\_name}'.")

return None

average = total / count

return average

except FileNotFoundError:

print(f"File '{csv\_file}' not found.")

return None

csv\_file\_path = 'file.csv'

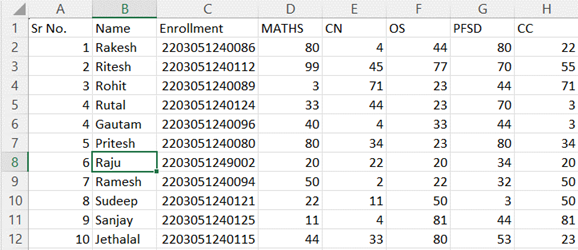
column\_to\_calculate = 'ENGLISH'

result = calculate\_average(csv\_file\_path, column\_to\_calculate)

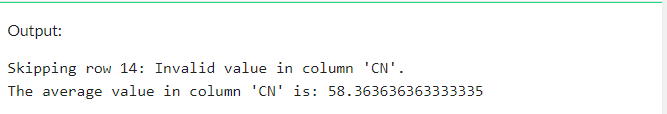
if result is not None:

print(f"The average value in column '{column\_to\_calculate}' is: {result}")

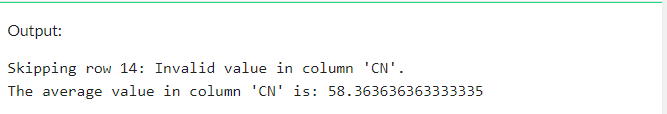
**file.csv:**



**Expected Output:**

****

**Actual Output:**

****

**Result:**

The result is the average value of the specified column in the provided CSV file. You can customize and expand this program based on your specific requirements and the structure of your CSV file.

**PRACTICAL - 5**

# AIM: A program that reads an Excel file and prints the data in a tabular format.

**Problem Statement:**

Develop a program that reads an Excel file and prints its data in a tabular format.

**Program Description:**

The program will take an Excel file as input, read its content, and display the data in a tabular format. It may use libraries like **openpyxl** or **pandas** to handle Excel file operations.

**Algorithm:**

1. Accept the Excel file path from the user.
2. Open and read the Excel file.
3. Extract the data from the sheets.
4. Display the data in a tabular format.

**Source Code:**

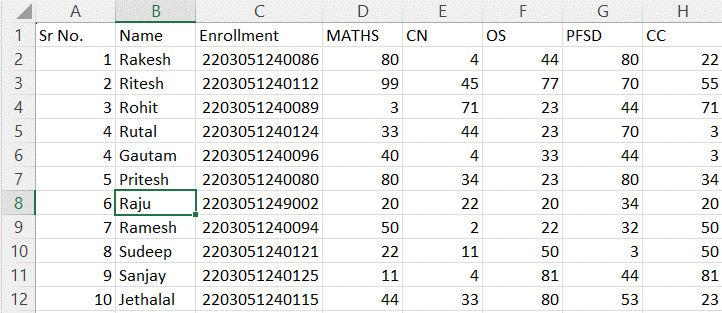
import pandas as pd

import openpyxl

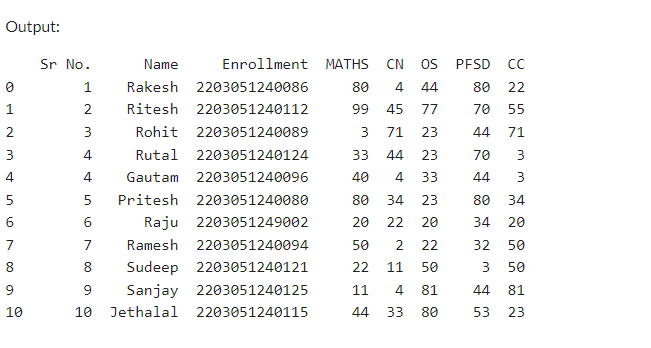
output = pd.read\_excel("delimited.xlsx")

print(output)

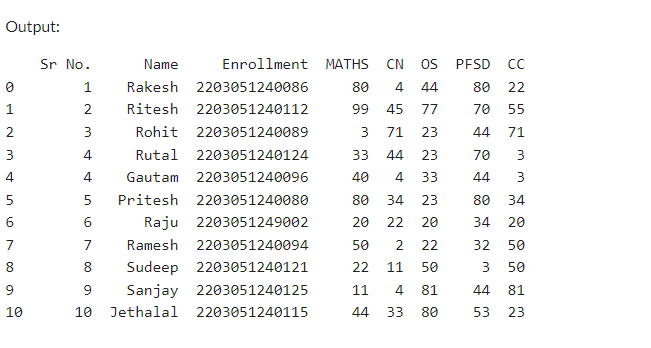
**delimited.xlsx:**

****

**Expected Output:**

****

**Actual Output:**

****

**Result:**

The result is the data from the Excel file displayed in a structured tabular format. You can customize and extend this program based on your specific requirements and the structure of your Excel file.