filehandling2

February 3, 2026

[ ]:
```
# Write a Python program, to process student data and print the output based on␣
↪the below requirements. (6 Marks)
# a. Retrieve StudentID, Name, Branch, Semester and CGPA fields from the␣
↪students txt file
# b. Update the value of StudentiD by inserting"" (hyphen) between S and␣
↪numeric part of the StudentID
# c. Split name of the student into FirstName and LastName
# d. Display the abbreviation of branches as follows
# If branch is
# "Computer Science" then display "CS"
# "Mechanical Engineering" then display "ME"
# "Electrical Engineering" then display "EE"
# Otherwise, display "others"
# e Print the header and processed data in the format given below
# Student ID, FirstName, LastName, Branch, Semester, COPA
# S-001,Alice, Wonderland, C#, 3,8.7
# S-002, Bob, Thebuilder, ME, 5,7.9
# S-003, Charlie, Chaplin, EE, 7,9.2
# S-004, Diana, Prince, others, 4,8.1

# given text file is in studentid,name,branch,semester,address,cgpa format
```

[19]:
```
# read file
with open("students.txt","r") as f:
    students=[]
    lines=f.readlines()

    for line in lines:
        if line.strip(): # line is not empty
            students.append(line.strip())

    # process data
    # a. retrive studentId,name,branch,semester,cgpa
    processed_students=[]
    for student in students:
        student_data=student.split(",")
        # print(student_data)
        # retrieve fields
        student_id=student_data[0]
        name=student_data[1]
        branch=student_data[2]
        semester=student_data[3]
        cgpa=student_data[5]

        # b.update studentId
        student_id=student_id.replace("S","S-")
        # c.split name
        name_parts=name.split(" ")
        first_name=name_parts[0]
        last_name=name_parts[1] if len(name_parts)>1 else ""

        # d.display abbreviation of branches
        if branch=="Computer Science":
            branch_abbr="CS"
        elif branch=="Mechanical Engineering":
            branch_abbr="ME"
        elif branch=="Electrical Engineering":
            branch_abbr="EE"
        else:
            branch_abbr="others"

        # print the header and processed data
        processed_students.
 ↪append(f"{student_id},{first_name},{last_name},{branch_abbr},{semester},{cgpa}")
    # print header
    print("Student ID, FirstName, LastName, Branch, Semester, CGPA")
    for p_student in processed_students:
        print(p_student.replace(",",", "))


    # print(students)

with open("processed_students.txt","w") as f:
    f.write("StudentID,FirstName,LastName,Branch,Semester,CGPA\n\n")
    f.write("\n\n".join(processed_students))
    f.write("\n")
```

```
Student ID, FirstName, LastName, Branch, Semester, CGPA
S-001, Alice, Wonderland, CS, 3, 8.7
S-002, Bob, Thebuilder, ME, 5, 7.9
S-003, Charlie, Chaplin, EE, 7, 9.2
S-004, Diana, Prince, others, 4, 8.1
S-005, Eve, Harrington, others, 6, 8.5
S-006, Frankenstein, Monster, others, 2, 7.5
S-007, Grace, Hopper, CS, 8, 9.5
```

```
S-008, Harry, Potter, others, 1, 7
S-009, Ivy, Queen, ME, 5, 8
S-010, Jack, Sparrow, others, 3, 7.8
```

[ ]:
```
# # another file handling question
# # Raw Data:
# S01, Rahul, Maths,78
# S02, Priya, Science,85
# S03, Arjun, English,65
# S04, Sneha, Maths, 92
# S05, Vikram, Science, 55
# Tasks:
# 1. Add 'STU-' before StuID.
# 2. Grade students: A if >= 80, B if 60-79, C if <60.
# 3. Mark subject category: Science subjects (Maths/Science) vs Non-Science.
# 4. Calculate Pass/Fail (Pass if >= 40).
# 5. Sort students by Marks descending.
# Expected Output:
# STU-S04, Sneha, Maths, 92, A, Science, Pass
# STU-S02, Priya, Science, 85, A, Science, Pass
# STU-S01, Rahul, Maths, 78, B, Science, Pass
# STU-S03, Arjun, English, 65, B, Non-Science, Pass
# STU-S05, Vikram, Science, 55.C, Science, Pass
```

[29]:
```
with open("students1.txt", "r") as f:
    students = []

    for line in f:
        data = [x.strip() for x in line.split(",")]

        # 1. Add 'STU-' before StudentID
        data[0] = "STU-" + data[0]

        # marks
        marks = int(data[3])

        # 2. Grade
        if marks >= 80:
            grade = "A"
        elif marks >= 60:
            grade = "B"
        else:
            grade = "C"
        data.append(grade)

        # 3. Subject category
        if data[2] in ["Maths", "Science"]:
```

```python
            subject_category = "Science"
        else:
            subject_category = "Non-Science"
        data.append(subject_category)

        # 4. Pass / Fail
        result = "Pass" if marks >= 40 else "Fail"
        data.append(result)

        students.append(data)

# sort by marks descending
students.sort(key=lambda x: int(x[3]), reverse=True)

# print output
print("StudentID, Name, Subject, Marks, Grade, Subject Category, Result")
for student in students:
    print(", ".join(student))
```

```
StudentID, Name, Subject, Marks, Grade, Subject Category, Result
STU-S04, Sneha, Maths, 92, A, Science, Pass
STU-S02, Priya, Science, 85, A, Science, Pass
STU-S01, Rahul, Maths, 78, B, Science, Pass
STU-S03, Arjun, English, 65, B, Non-Science, Pass
STU-S05, Vikram, Science, 55, C, Science, Pass
```

[ ]:
```python
# Dataset 1: Movie Ratings
# Raw Data:
# M01, Inception, Christopher Nolan, 2010, Sci-Fi,8.8
# M02, Titanic, James Cameron, 1997, Romance, 7.9
# M03, The Dark Knight, Christopher Nolan, 2008, Action, 9.0
# M04, Avatar, James Cameron, 2009, Sci-Fi, 7.8
# M05, Interstellar, Christopher Nolan, 2014, Sci-Fi,8.6
# Tasks:
# 1. Add prefix 'MOV-' before ID.
# 2. Extract decade from Year (e.g., 2010 - 2010s).
# 3. Classify as 'Hit' if IMDB >= 8.5 else 'Average'.
# 4. Count number of movies per Director.
# 5. Sort by IMDB descending.
# Expected Output:
# MOV-M03, The Dark Knight, Christopher Nolan, 2000s, Action, 9.0, Hit, 2
# MOV-M01, Inception, Christopher Nolan, 2010s, Sci-Fi, 8.8, Hit, 2
# MOV-M05, Interstellar, Christopher Nolan, 2010s, Sci-Fi,8.6,Hit, 2
# MOV-M02, Titanic, James Cameron, 1990s, Romance, 7.9.Average, 2
# MOV-M04, Avatar, James Cameron, 2000s, Sci-Fi, 7.8, Average, 2
```

[34]:
```python
# M01, Inception, Christopher Nolan, 2010, Sci-Fi,8.8
# M02, Titanic, James Cameron, 1997, Romance, 7.9
# M03, The Dark Knight, Christopher Nolan, 2008, Action, 9.0
# M04, Avatar, James Cameron, 2009, Sci-Fi, 7.8
# M05, Interstellar, Christopher Nolan, 2014, Sci-Fi,8.6

with open("movies.txt","r") as f:
    movies=[]
    lines=f.readlines()
    director_count={}

    for line in lines:
        data=line.strip().split(",")

        # 1. Add prefix 'MOV-' before ID.
        movie_id="MOV-"+data[0]

        title=data[1]
        director=data[2]

        # extract decade from year
        year=int(data[3])
        decade=(year//10)*10
        decade_str=str(decade)+"s"
        # classify as Hit or Average
        imdb=float(data[5])
        classification="Hit" if imdb>=8.5 else "Average"

        # count number of movies per director
        # dic={}
        # for movie in movies:
        #     director=movie[2]
        #     if director in dic:
        #         dic[director]+=1
        #     else:
        #         dic[director]=1
        # director_count=dic.get(data[2],0)+1 # include current movie
        # director_count=sum(1 for movie in movies if movie[2]==data[2])+1
        director_count[director]=director_count.get(director,0)+1

        movies.
 ↪append([movie_id,data[1],data[2],decade_str,data[4],data[5],classification,director_count])
    # sort by imdb descending
    movies.sort(key=lambda x:float(x[5]),reverse=True)
    # print output
    for movie in movies:
        print(",".join(map(str,movie)))
```

```
MOV-M03, The Dark Knight, Christopher Nolan,2000s, Action, 9.0,Hit,{'
Christopher Nolan': 3, ' James Cameron': 2}
MOV-M01, Inception, Christopher Nolan,2010s, Sci-Fi,8.8,Hit,{' Christopher
Nolan': 3, ' James Cameron': 2}
MOV-M05, Interstellar, Christopher Nolan,2010s, Sci-Fi,8.6,Hit,{' Christopher
Nolan': 3, ' James Cameron': 2}
MOV-M02, Titanic, James Cameron,1990s, Romance, 7.9,Average,{' Christopher
Nolan': 3, ' James Cameron': 2}
MOV-M04, Avatar, James Cameron,2000s, Sci-Fi, 7.8,Average,{' Christopher Nolan':
3, ' James Cameron': 2}
```