



Module Title

Fundamentals of Data Science

Assessment Weightage & Type

Weekly Assignment 5 and 6 - Coursework & Regular

Year

2025

Student Name: NIRVIK K.C.

UWE ID: 25024649

Assignment Due Date: July 7, 2025

Assignment Submission Date: July 7, 2025



Bi-weekly assignment

Module Details

Module Code	UFCFK1-15-0
Module Title	Fundamentals of Data Science
Module Tutors	Saurav Gautam
Year	2024-2025
Component/Element Number	PSA/Bi-weekly assignment/Regular
Weighting	10%

Dates

Submission Date	07-July-2025
Submission Place	Backboard
Submission Time	23:59
Submission Notes	Submit Gitlab URL

Assignment 2

This assignment consists of the programming questions related to the topics of week 5 and week 6. The main topics of questions are: Python Basics, Operators, and Conditional Statements.

All the students are required to follow the format of the program as specified in the guideline below.

1. All the programs should have initial **doc string** comment (‘’ description of program’’) mentioning what your program will do.
2. Try to maintain single/multi-line comments in the place where needed to make the program understandable.
3. Maintain proper indention and newline spaces to increase the readability of the program.
4. The deliverable are 2 type of files (a single word file and multiple python program files):
 - a) Separate python program files with **.py** extension (e.g. program_name.py). Provide a relevant name to your program file on the basis of functionality of the program.
 - b) A word file describing the working of all the programs according to their number. The details required in this is the description of program, screenshot of the testing (input given and output obtained in the execution environment such as IDLE or Command prompt or terminal whichever you prefer.). It is preferred that you work with multiple inputs and outputs.

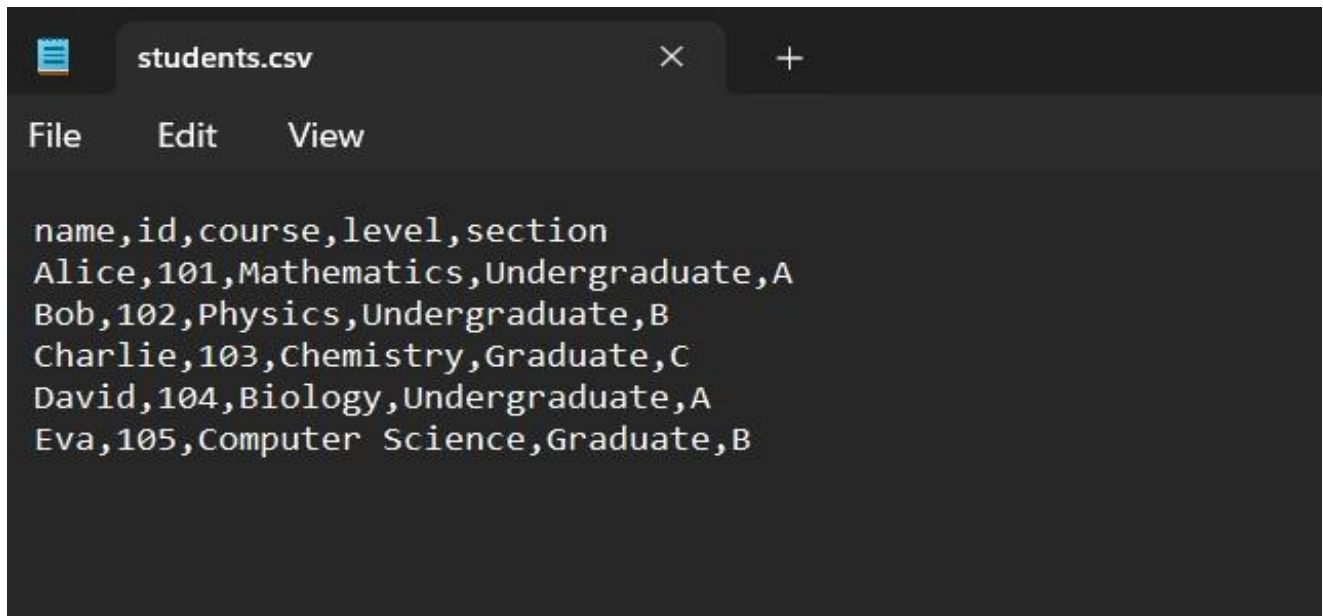
Questions

1. Write a program to read a file “students.csv” and print its value. The fields to read are name, id, course, level and section.

Answer:

The given python program below reads a CSV file named “students.csv” and prints the values of the specified fields such as name, id, course, level, and section.

Sample Data for csv.file:

A screenshot of a text editor window titled 'students.csv'. The window has a dark background and a menu bar with 'File', 'Edit', and 'View'. The text inside the editor is a CSV file with the following content:

```
name,id,course,level,section
Alice,101,Mathematics,Undergraduate,A
Bob,102,Physics,Undergraduate,B
Charlie,103,Chemistry,Graduate,C
David,104,Biology,Undergraduate,A
Eva,105,Computer Science,Graduate,B
```

Following code for input :

```

*Read a students.csv file.py - F:/BSc (Hons) Computer Science - Artificial Intelligence Semest...
File Edit Format Run Options Window Help
import csv

def read_students_file(filename):
    try:
        with open(filename, mode='r', newline='') as file:
            reader = csv.DictReader(file)
            for row in reader:
                name = row['name']
                student_id = row['id']
                course = row['course']
                level = row['level']
                section = row['section']
                print(f"Name: {name}, ID: {student_id}, Course: {course}, Level:
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Specify the CSV file name
csv_file = 'students.csv'

# Call the function to read and print the student data
read_students_file(csv_file)

```

Output obtained in execution :

```

IDLE Shell 3.13.3
File Edit Shell Debug Options Window Help
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr 8 2025, 14:47:33) [MSC v.1943 64 b
it (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester
II/FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Read
a students.csv file.py
Name: Alice, ID: 101, Course: Mathematics, Level: Undergraduate, Section: A
Name: Bob, ID: 102, Course: Physics, Level: Undergraduate, Section: B
Name: Charlie, ID: 103, Course: Chemistry, Level: Graduate, Section: C
Name: David, ID: 104, Course: Biology, Level: Undergraduate, Section: A
Name: Eva, ID: 105, Course: Computer Science, Level: Graduate, Section: B
>>>

```

Python Program File: Saved the above code in a file named “Read a students.csv file.py.”

Explanation of code:

Importing the CSV Module:

The csv module is imported in the program, which gives functionality to read and write CSV files.

Functionality Definition:

The ‘read_students_file(filename)’ is a python function that reads a CSV file as an argument.

Opening the File:

The program uses a ‘with’ statement to open the file in read mode. The file is now properly closed after its suite finishes.

Reading the CSV File:

The ‘csv.DictReader(file)’ reads the CSV file and converts the information into a dictionary with the column headers as the keys.

Printing the Data:

The program iterates through each row in the CSV file and prints the specified fields such as name, id, course, level, and section.

Error Handling:

The program includes error handling to check ‘FileNotFoundError’ if the file exists or not and a general exception handler for any other errors.

Output of the Program:

A Data of students.csv file is displayed which contains fields like name, id, course, level, and section.

Conclusion of the Program:

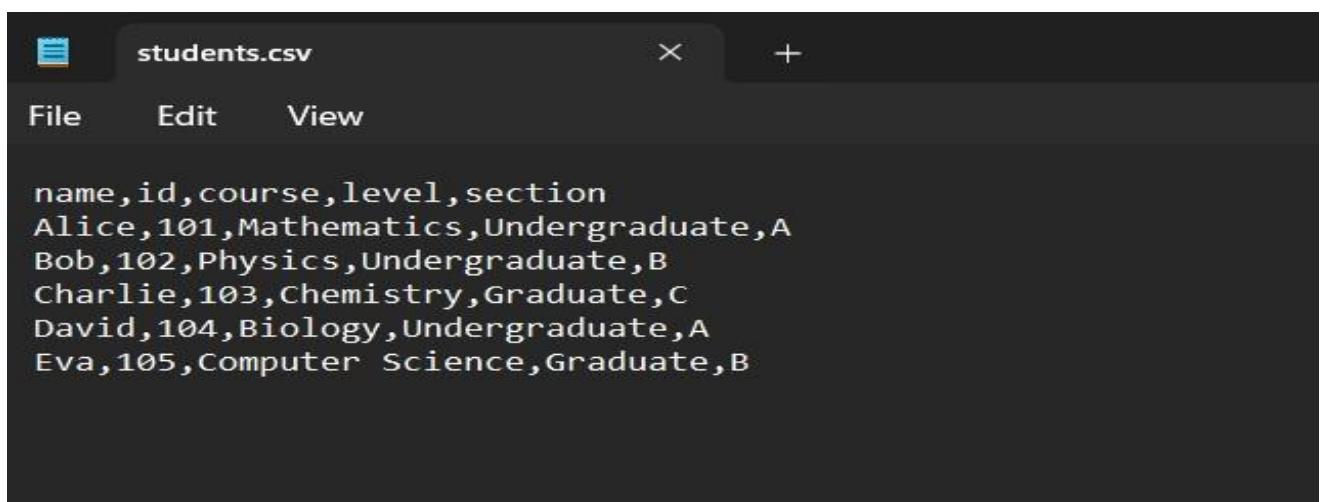
The given program is an example of how to read data related to some fields from a CSV file and print the fields as program iterates through row.

2. Write a program to take the user details as the input from the user and write it to the existing file “students.csv”. The new record should be added to the end of the file. The fields to take input are name, id, course, level and section.

Answer:

The given python program below takes user details as input from the user and appends it to the existing file “students.csv”. The new record is added at the end of the file.

Existing CSV File:

A screenshot of a text editor window titled 'students.csv'. The window has a menu bar with 'File', 'Edit', and 'View'. The text inside the editor is as follows:

```
name,id,course,level,section
Alice,101,Mathematics,Undergraduate,A
Bob,102,Physics,Undergraduate,B
Charlie,103,Chemistry,Graduate,C
David,104,Biology,Undergraduate,A
Eva,105,Computer Science,Graduate,B
```

Following code for input :

```

*Write new records to the existing students.csv file.py - F:/BSc (Hons) Computer Science - Ar...
File Edit Format Run Options Window Help
import csv

def add_student_record():

    # Take input from user

    name = input("Enter student name: ").strip()
    student_id = input("Enter student ID: ").strip()
    course = input("Enter course: ").strip()
    level = input("Enter level (Undergraduate/Graduate): ").strip()
    section = input("Enter section: ").strip().upper()

    # Prepare the new record
    new_record = {
        'name': name,
        'id': student_id,
        'course': course,
        'level': level,
        'section': section
    }

    # Append to the CSV file
    try:
        with open('students.csv', mode='a', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=['name', 'id', 'course', 'l

            # Check if file is empty to write header
            if file.tell() == 0:
                writer.writeheader()

            writer.writerow(new_record)
            print("Student record added successfully!")
    except Exception as e:
        print(f"An error occurred: {e}")

# Run the function
add_student_record()

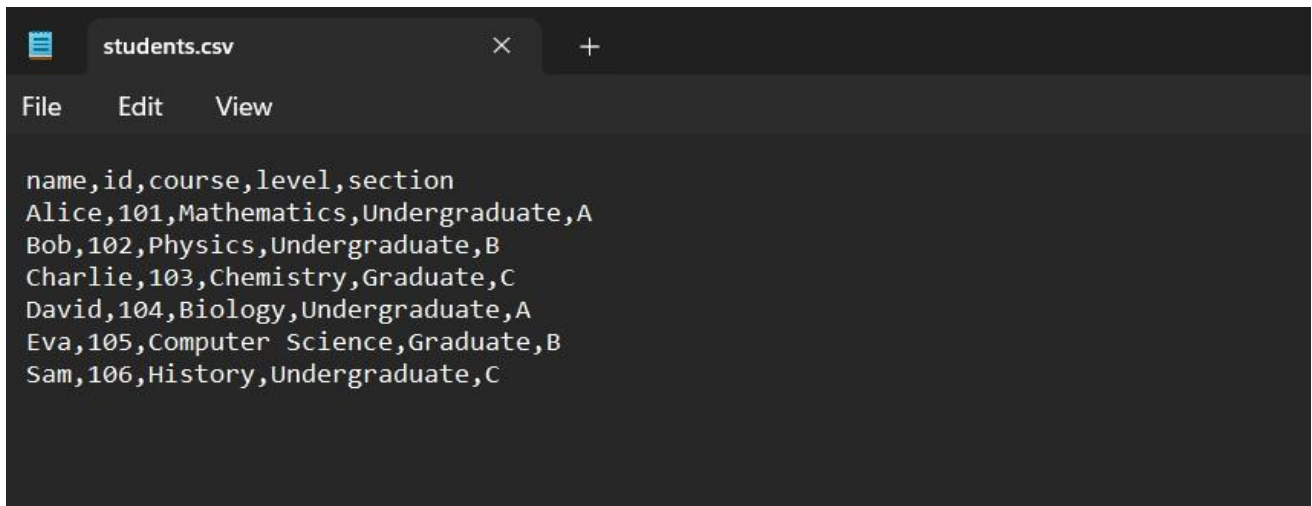
```

Output obtained in execution :

```

IDLE Shell 3.13.3
File Edit Shell Debug Options Window Help
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr 8 2025, 14:47:33) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/
FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Write new re
cords to the existing students.csv file.py
Enter student name: Sam
Enter student ID: 106
Enter course: History
Enter level (Undergraduate/Graduate): Undergraduate
Enter section: C
Student record added successfully!
>>>

```


New Record added at the of the File:A screenshot of a text editor window titled 'students.csv'. The window has a menu bar with 'File', 'Edit', and 'View'. The text content is a CSV file with the following data:

```
name,id,course,level,section
Alice,101,Mathematics,Undergraduate,A
Bob,102,Physics,Undergraduate,B
Charlie,103,Chemistry,Graduate,C
David,104,Biology,Undergraduate,A
Eva,105,Computer Science,Graduate,B
Sam,106,History,Undergraduate,C
```

Python Program File: Saved the above code in a file named “Write new records to the existing student.csv file.py.”

Explanation of code:

Importing the CSV Module:

The program imports the CSV module, which provided functionality to read and write the CSV files.

Function Definition:

The ‘read_students_file(filename)’ python function reads a CSV file as an argument.

Opening the File:

The program uses a ‘with’ statement to open the file in read mode. This ensures that the file is properly closed after its suite finishes.

Using 'newline='': This argument is used to prevent extra blank lines from being added in some environments when writing to the file.

Creating a DictWriter Object:

Object 'DictWriter' named 'writer' is created, which will be used to write dictionaries to the CSV file. The 'fieldnames' parameter specifies the order of the columns.

Writing the Header:

The program checks if the file is empty using file.tell(), it returns the current position of the file. If the file is empty (position is at 0), it writes the header row using writer.writeheader(). This ensures that the header is written only once when the file is created.

Writing the New Record:

The writer.writerow(new_record) method is called to append the new record (the dictionary) to the CSV file.

Success Message:

If the record is added successfully, it prints a confirmation message to the user.

Error Handling:

The program includes a try-except block to catch any exceptions that may occur during the file operations. Displaying an error message with the corresponding exception details.

Function Call:

The add_student_record() function is called to execute the program.

Output of the Program:

New record is added at the end of file with similar details as the input from the user. The changes can be observed in the existing csv file as well.

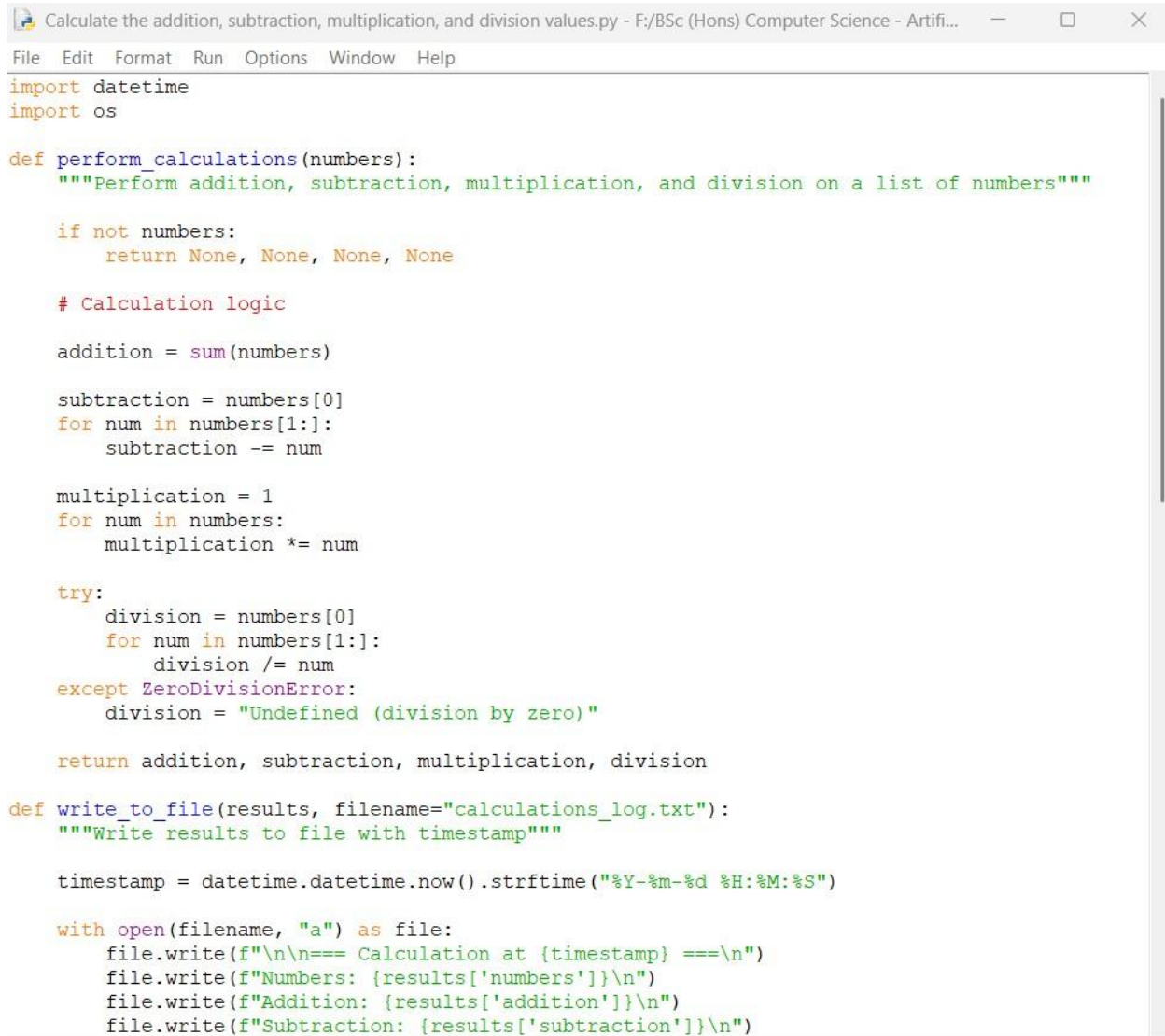
Conclusion of the Program:

The given program is an example of how to append new details as input from the user to an existing csv file with the record will display at the end. It helps to ensure data integrity and user-friendly interaction.

3. Write a program input any list of numbers from the user, calculate the addition, subtraction, multiplication and division values and write into a file with the current date and time. The program should allow the user to repeat the operation until they want. When the user choose the option to exit the program, the user should be displayed with the details in the current file in a proper format.

Answer:

The given python program below takes a list of numbers from the user, performs arithmetic calculations (addition, subtraction, multiplication, and division). It writes the result to a file with the current data and time. The operation can be repeated multiple times by the user until they choose to exit it, then the contents of file are displayed in a formatted way.

Following code for input:


```

Calculate the addition, subtraction, multiplication, and division values.py - F:/BSc (Hons) Computer Science - Arti...
File Edit Format Run Options Window Help
import datetime
import os

def perform_calculations(numbers):
    """Perform addition, subtraction, multiplication, and division on a list of numbers"""

    if not numbers:
        return None, None, None, None

    # Calculation logic

    addition = sum(numbers)

    subtraction = numbers[0]
    for num in numbers[1:]:
        subtraction -= num

    multiplication = 1
    for num in numbers:
        multiplication *= num

    try:
        division = numbers[0]
        for num in numbers[1:]:
            division /= num
    except ZeroDivisionError:
        division = "Undefined (division by zero)"

    return addition, subtraction, multiplication, division

def write_to_file(results, filename="calculations_log.txt"):
    """Write results to file with timestamp"""

    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    with open(filename, "a") as file:
        file.write(f"\n\n== Calculation at {timestamp} ==\n")
        file.write(f"Numbers: {results['numbers']}\n")
        file.write(f"Addition: {results['addition']}\n")
        file.write(f"Subtraction: {results['subtraction']}\n")

```

Continue:

```

def display_file_contents(filename="calculations_log.txt"):
    """Display file contents in a formatted way"""

    if not os.path.exists(filename):
        print("No calculations have been performed yet.")
        return

    print("\n=== Calculation History ===")
    with open(filename, "r") as file:
        print(file.read())

def main():
    print("Number Operations Calculator")
    print("Enter numbers separated by spaces")
    print("Enter 'q' to quit and view results\n")

    while True:
        try:
            user_input = input("Enter numbers: ")
            if user_input.lower() == 'q':
                break

            numbers = [float(num) for num in user_input.split()]

            addition, subtraction, multiplication, division = perform_calculations(numbers)

            if None in (addition, subtraction, multiplication, division):
                print("Error: Please enter at least one number")
                continue

            results = {
                "numbers": numbers,
                "addition": addition,
                "subtraction": subtraction,
                "multiplication": multiplication,
                "division": division
            }

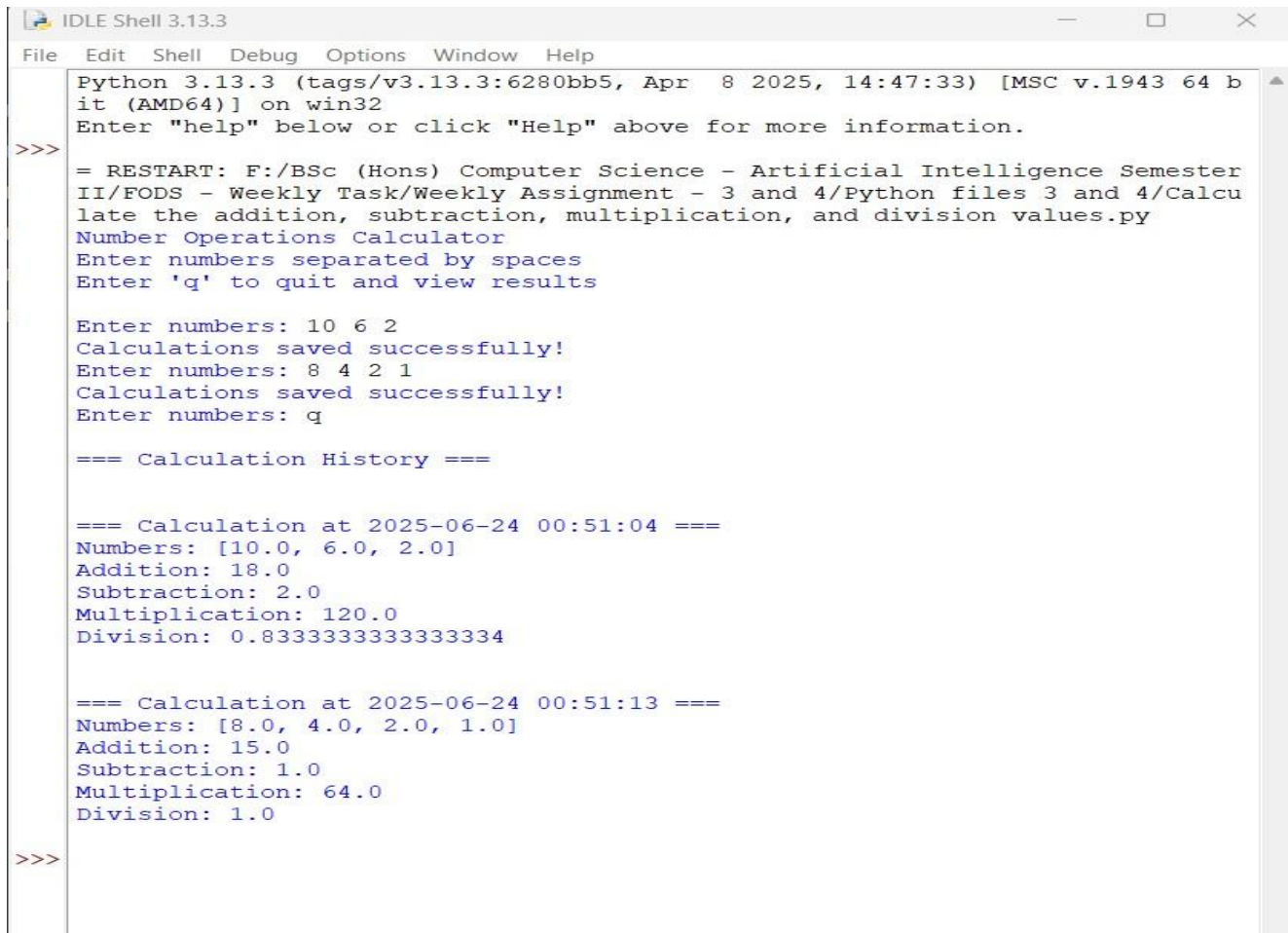
            write_to_file(results)
            print("Calculations saved successfully!")

        except ValueError:
            print("Error: Please enter valid numbers separated by spaces")
        except Exception as e:
            print(f"An unexpected error occurred: {e}")

    display_file_contents()

if __name__ == "__main__":
    main()

```

Output obtained in execution:

```
IDLE Shell 3.13.3
File Edit Shell Debug Options Window Help
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr 8 2025, 14:47:33) [MSC v.1943 64 b
it (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester
II/FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Calcu
late the addition, subtraction, multiplication, and division values.py
Number Operations Calculator
Enter numbers separated by spaces
Enter 'q' to quit and view results

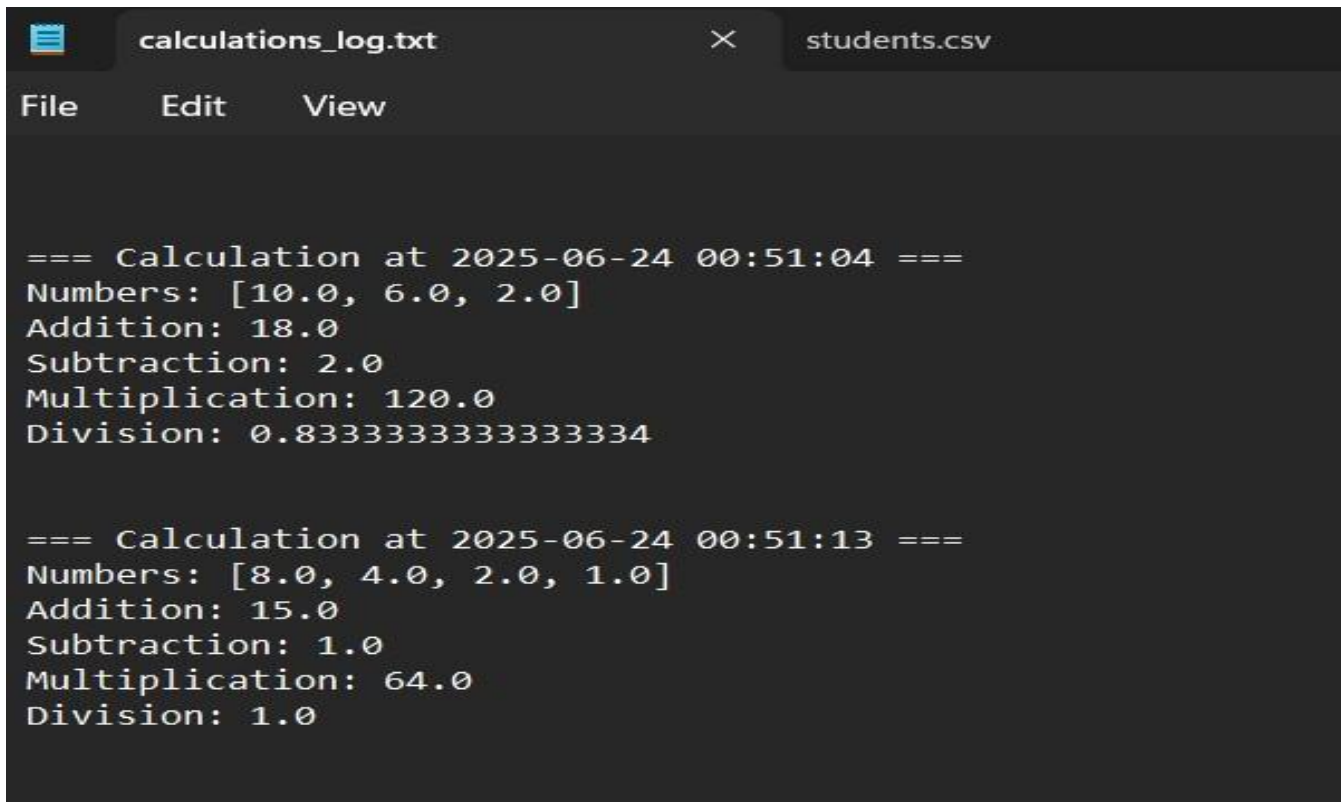
Enter numbers: 10 6 2
Calculations saved successfully!
Enter numbers: 8 4 2 1
Calculations saved successfully!
Enter numbers: q

=== Calculation History ===

=== Calculation at 2025-06-24 00:51:04 ===
Numbers: [10.0, 6.0, 2.0]
Addition: 18.0
Subtraction: 2.0
Multiplication: 120.0
Division: 0.8333333333333334

=== Calculation at 2025-06-24 00:51:13 ===
Numbers: [8.0, 4.0, 2.0, 1.0]
Addition: 15.0
Subtraction: 1.0
Multiplication: 64.0
Division: 1.0
>>>
```

Creates calculation_log.txt:



```
=== Calculation at 2025-06-24 00:51:04 ===  
Numbers: [10.0, 6.0, 2.0]  
Addition: 18.0  
Subtraction: 2.0  
Multiplication: 120.0  
Division: 0.8333333333333334  
  
=== Calculation at 2025-06-24 00:51:13 ===  
Numbers: [8.0, 4.0, 2.0, 1.0]  
Addition: 15.0  
Subtraction: 1.0  
Multiplication: 64.0  
Division: 1.0
```

Python Program File: Saved the above code in a file named “Calculation the addition, subtraction, multiplication, and division values.py.”

Explanation of code:

Importing Modules:

The module ‘datetime’ is imported to work with the date and time, allowing us to calculate and observe timestamp.

The ‘os’ module is imported to check for the existence of the file and perform operations in it.

Function Definition:

The ‘perform_calculations(numbers)’ function takes a list of numbers as input from the user and performs four arithmetic operations.

Empty List Check:

If the list of numbers is empty, it returns ‘None’ for all operations.

Addition Calculation:

The sum(numbers) function calculates the all of numbers in the list.

Subtraction Calculation:

The first number is assigned to ‘subtraction’, and then each number is subtracted from that number in a loop.

Multiplication Calculation:

The variable ‘multiplication’ is initialized to 1, and each number in the list is multiplied together in a loop.

Division Calculation:

The first number is assigned to ‘division’, and each number is then divided into it in a loop. A ‘try-except’ block is used to handle division be zero, setting ‘division’ to “Undefined” if a division by zero occurs.

Return Values:

The function returns the results of the four operations as a tuple.

Function Definition:

The ‘write_to_file(results, filename)’ function takes the results of calculations and a filename (calculation_log.txt) as input.

Timestamp Creation:

The current data and time are formatted as a string for logging.
“

Opening the File:

With `open(filename, “a”) as file:”`

`File.write(f“\n\n=== Calculation at {timestamp} ===\n”)`

The file is opened in append mode(“a”), which allows new data to be added without overwriting existing content.

Writing Results:

The function writes the timestamp, the list of numbers, and the results of the calculations into a file.

Function Definition:

The ‘`display_file_contents(filename)`’ function used in the program checks if the log file exist.

Function Existence Check:

It prints a message indicating that no calculations have been performed if the resulting file does not exist.

Reading the File:

It opens the file in read mode(“r”) and prints its contents to the console if the file does exist.

Main Function Definition:

The `main()` function serves as the entry point of the program.

User Instructions:

It prints instructions for the user to follow on how to print the input numbers and how to quit the program.

Input Loop:

The program enters an infinite loop, prompting the user to enter the required numbers.

Quit Option:

The user can input 'q', the loop breaks and the program prepares for exit.

Processing Input:

The input from the user is split into individual strings, which are converted to floats value and stored in the numbers list.

Performing Calculations:

The 'perform_calculation(number)' function is called with the list of numbers, and the results are rearranged into corresponding variables.

Error Handling:

If any of the results are 'None', it indicates that no valid numbers were provided by the user and an error message is printed.

Storing Results:

A dictionary named 'results' is created to store the numbers and the calculations.

Writing to the File:

The 'write_to_file(results)' function is called to log the results, and print a success message.

Exception Handling:

The program includes exception handling for error values 'valueError' in case of invalid input and exception handler for any other unexpected errors by the user.

Displaying Results:

When the user chooses to quit, the `display_file_contents()` function is called to show the contents of the log file.

Entry Point:

The `if __name__ == "__main__":` line checks if the script is being run directly and calls the `'main()'` function to start the program. The script should not be imported as a module.

Output of the Program:

It displays a formatted and a history of calculations when the user exits.

Conclusion of the Program:

The given program is an example of how users can perform calculations and maintain a log of their operations. It helps to track arithmetic results over time.

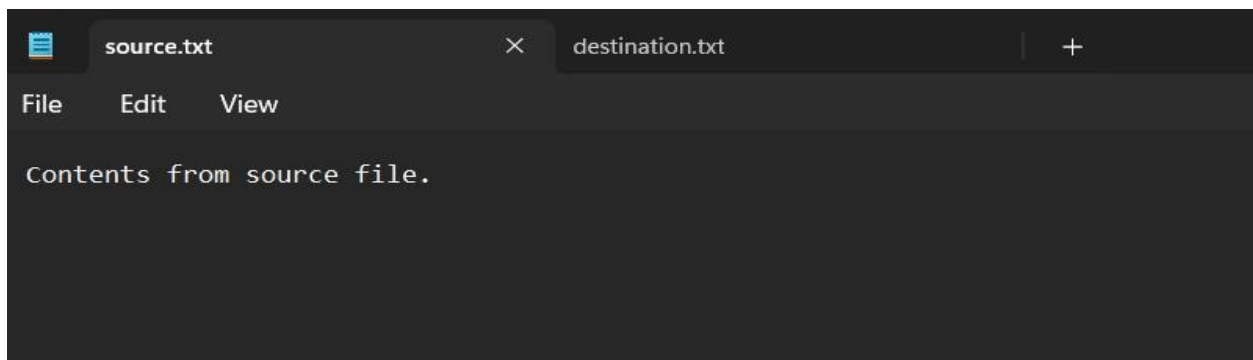
4. Write a program to take read a file and write it into another file. The input and output file names should be taken input from the user. Use the concept of `try/except` to handle the exception. Provide the proper error if the file does not exist while reading and also if the file for output exists.

Answer:

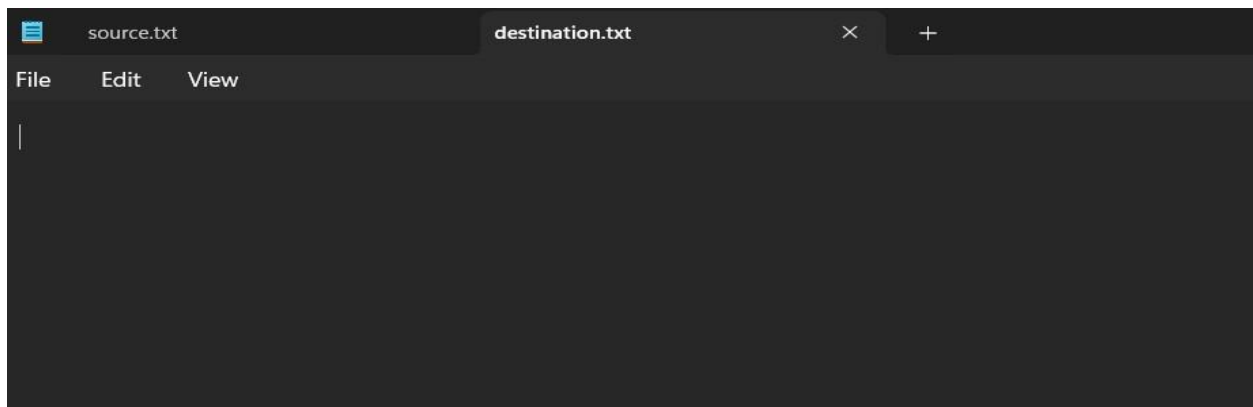
The given python program that reads from a file and writes its content to another file. It includes exception handling if the file does not exist while reading.

25024649

Source.txt File:



Destination.txt File:



Following code for input:

```

Read a file and write it into another file.py - F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/FODS - Week...
File Edit Format Run Options Window Help
def copy_file():
    """
    Reads content from a source file and writes to a destination file.
    Handles various file operation exceptions gracefully.
    """

    try:
        # Get input and output file names from user
        input_file = input("Enter the name of the input file: ")
        output_file = input("Enter the name of the output file: ")

        # Check if output file already exists
        if os.path.exists(output_file):
            overwrite = input(f"File '{output_file}' already exists. Overwrite? (y/n): ").lower()
            if overwrite != 'y':
                print("Operation cancelled by user.")
                return

        # Read from input file
        with open(input_file, 'r') as source:
            content = source.read()

        # Write to output file
        with open(output_file, 'w') as destination:
            destination.write(content)

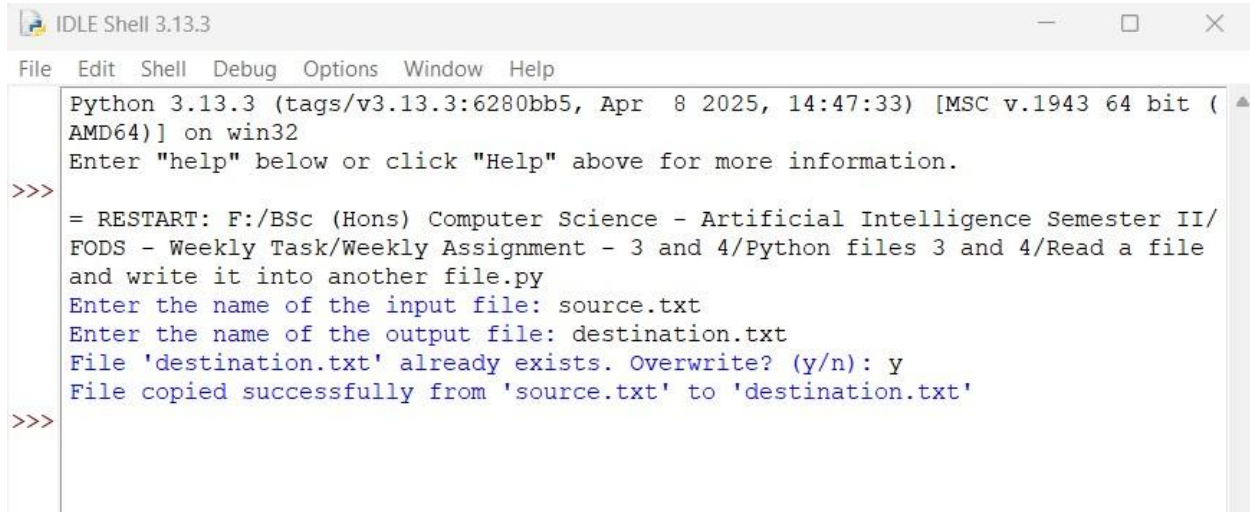
        print(f"File copied successfully from '{input_file}' to '{output_file}'")

    except FileNotFoundError:
        print(f"Error: The input file '{input_file}' does not exist.")
    except PermissionError:
        print("Error: You don't have permission to access one of these files.")
    except IOError as e:
        print(f"An I/O error occurred: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

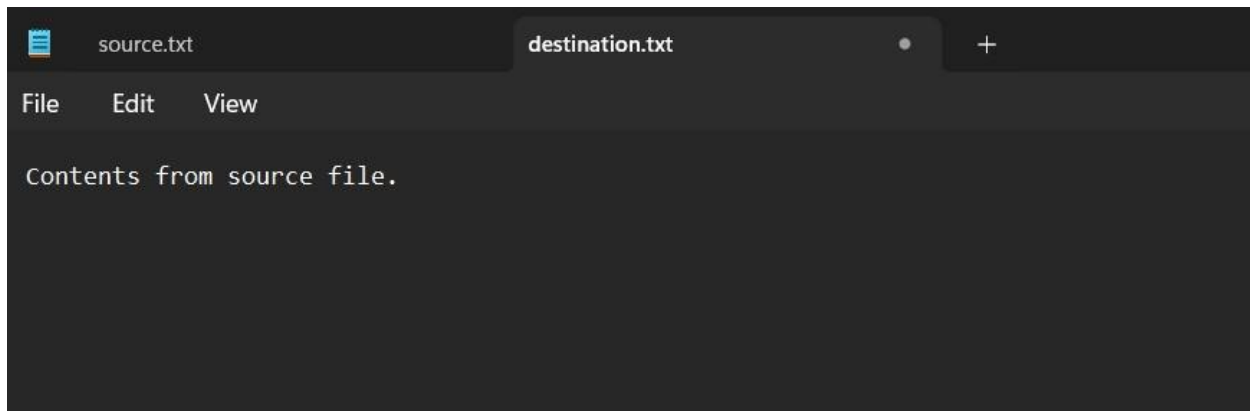
# Import needed for os.path.exists
import os

# Run the program
if __name__ == "__main__":
    copy_file()

```

Output obtained in execution:

```
IDLE Shell 3.13.3
File Edit Shell Debug Options Window Help
Python 3.13.3 (tags/v3.13.3:6280bb5, Apr 8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>> = RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/
FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Read a file
and write it into another file.py
Enter the name of the input file: source.txt
Enter the name of the output file: destination.txt
File 'destination.txt' already exists. Overwrite? (y/n): y
File copied successfully from 'source.txt' to 'destination.txt'
>>>
```

File copied successfully from source.txt to destination.txt file:

```
source.txt destination.txt
File Edit View
Contents from source file.
```

Python Program File: Saved the above code in a file named “Read from a file and write it into another file.py.”

Explanation of code:

The `'copy_file()'` function is for copying content from one file to another which is described in the docstring.

Try Block:

The program begins a `'try'` block to catch any exceptions that may occur during the file operations.

User Input:

It prompts the user to enter the names of the input and output files using the `'input()'` function.

Output File Check:

The program checks if the output file already exists or not using `'os.path.exists()'`.

Overwrite Confirmation:

If the file exists then it prompts the user, asking for confirmation to overwrite it. The response printed is converted to lowercase.

Cancel Operation:

If the user does not confirm (i.e., enters anything other `'y'`), the program prints a cancellation message and exits the function using `'return'`.

Reading the Input File:

The program opens the input file in read mode (`'r'`) using a `'with'` statement, with `open(input_file, 'r')`. This ensures that the file is closed properly after reading it.

Reading Content:

The entire content of the file is read and stored in the variable `'content'`.

Writing to the Output File:

The program opens the output file in write mode('w') using another 'with' statement, with open(output_file, 'w').

Writing Content:

The content read from the input file is written to the output file.

Success Message:

A confirmation message is printed if the file operations are successful which informs the user that the file has been copied.

FileNotFoundError Handling:

If the input file does not exist, a 'FileNotFoundError' is raised, and the program prints an error messages which indicates that the input file could not be found.

PermissionError Handling

If there are permission issues (e.g., the user does not have the permission to read the input file or write to the output file). A 'PermissionError' is raised and an error message is printed.

Input/output Error Handling:

Except IOError as e:

Print(f' An I/O error occurred: {e}')

This block catches any general I/O errors that may occur during the file operations. It prints a message along with the error details.

General Exception Handling:

Except Exception as e:

Print(f' An unexpected error occurred: {e}')

This block catches any other unexcepted exceptions that may not have been handled by the previous blocks. Again, it prints a message with the error details.

Importing the os Module:

The 'os' module is imported to use the 'os.path.exists' function, which checks for existence of files.

Entry Point:

The if `__name__ == "__main__"` line checks if the script is being run directly and calls the 'copy_file()' function to execute the program.

Output of the Program:

The program display contents of one file is copied to another file, ensuring the errors are handled carefully. When the output file already exist is shown in output. There are other cases like for normal file operation or input file does not exist.

Conclusion of the Program:

The given program is an example of how users can copy the contents of one file to another file while catching and handling potential issues.

5. Create a class Student with the attributes such as id, name, address, admission year, level, section. Instantiate the object of class to take input for all the attributes and display the output.

Answer:

The given python program below creates a 'student' class with specified attributes takes user input to instantiate an object and displays the resulting output.

Following code for input:

```

Create a class student with the attributes.py - F:/BSc (Hons) Computer Science - Artificial Int...
File Edit Format Run Options Window Help
class Student:
    def __init__(self, id, name, address, admission_year, level, section):
        """Initialize a Student object with provided attributes"""
        self.id = id
        self.name = name
        self.address = address
        self.admission_year = admission_year
        self.level = level
        self.section = section

    def display_info(self):
        """Display all student information in a formatted way"""
        print("\nStudent Information:")
        print(f"ID: {self.id}")
        print(f"Name: {self.name}")
        print(f"Address: {self.address}")
        print(f"Admission Year: {self.admission_year}")
        print(f"Level: {self.level}")
        print(f"Section: {self.section}")

def create_student():
    """Create and return a Student object with user input"""

    print("Enter Student Details:")
    id = input("ID: ")
    name = input("Name: ")
    address = input("Address: ")
    admission_year = input("Admission Year: ")
    level = input("Level (e.g., Undergraduate/Graduate): ")
    section = input("Section: ")

    return Student(id, name, address, admission_year, level, section)

# Main program

if __name__ == "__main__":

    # Create a student object with user input
    student = create_student()

    # Display the student information
    student.display_info()

```

Output obtained in execution:

```

Python 3.13.3 (tags/v3.13.3:6280bb5, Apr 8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/
FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Create a class student with the attributes.py
Enter Student Details:
ID: S00120
Name: John Smith
Address: XYZ College, Campus Town
Admission Year: 2025
Level (e.g., Undergraduate/Graduate): Undergraduate
Section: A

Student Information:
ID: S00120
Name: John Smith
Address: XYZ College, Campus Town
Admission Year: 2025
Level: Undergraduate
Section: A
>>>

```

Python Program File: Saved the above code in a file named “Create a class student with the attributes.py.”

Explanation of code:

Class Definition:

The ‘student’ class is defined with attributes: ‘id’, ‘name’, ‘address’, ‘admission_year’, ‘level’, and ‘section’. The ‘__init__ method’ serves as the constructor to initialize these attributes. The ‘display_info’ method prints details of all students in a formatted way.

Object Creation:

The 'create_student()' function prompts the user to input each attribute value which creates and returns a new student object with these values. Each input by the user in each field is clearly labeled for his/her understanding.

Main Program:

Instantiates a student object using the 'create_student()' function which calls the 'display_info()' method to show all the information collected of student.

Output of the Program:

The program displays fields which user can input himself/herself to fill the details of student information. Then, the information is displayed in a formatted way to the user. This shows user-friendly input/output handling in the program.

Conclusion of the Program:

The given program is an example of how to create a student class with various specified attributes, which is taken as input from the user to instantiate an object and the resulting output is displayed.

6. Write a program to create a class called recipe with the attributes such as id, name, ingredients, descriptions. Create another class called recipe book to manage the collection of recipes.

Answer:

The given python program below implements a 'Recipe' class and 'RecipeBook' class to manage collections of recipes.

Following code for input:

```

Create a class called recipe with the attributes.py - F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/FODS - Weekly...
File Edit Format Run Options Window Help
class Recipe:
    def __init__(self, recipe_id, name, ingredients, description):
        """Initialize a Recipe with all necessary attributes"""
        self.id = recipe_id
        self.name = name
        self.ingredients = ingredients
        self.description = description

    def display(self):
        """Display the recipe details in a formatted way"""
        print(f"\nRecipe ID: {self.id}")
        print(f"Name: {self.name}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")
        print(f"Description: {self.description}")

class RecipeBook:
    def __init__(self):
        """Initialize an empty recipe book"""
        self.recipes = []

    def add_recipe(self, recipe):
        """Add a recipe to the recipe book"""
        self.recipes.append(recipe)
        print(f"\nRecipe '{recipe.name}' added successfully!")

    def display_all(self):
        """Display all recipes in the book"""
        if not self.recipes:
            print("\nThe recipe book is empty!")
            return

        print("\n===== RECIPE BOOK =====")
        for recipe in self.recipes:
            recipe.display()
            print("-" * 30)

```

Continue:

```

def find_by_id(self, recipe_id):
    """Find and display a recipe by its ID"""
    for recipe in self.recipes:
        if recipe.id == recipe_id:
            print("\nFound recipe:")
            recipe.display()
            return recipe
    print(f"\nNo recipe found with ID: {recipe_id}")
    return None

def get_recipe_input():
    """Helper function to get recipe details from user"""
    recipe_id = input("Enter recipe ID: ")
    name = input("Enter recipe name: ")

    print("Enter ingredients (press enter after each, blank to finish):")
    ingredients = []
    while True:
        ingredient = input("- ")
        if not ingredient.strip():
            break
        ingredients.append(ingredient)

    description = input("Enter recipe description: ")
    return Recipe(recipe_id, name, ingredients, description)

def menu():
    print("\nRECIPE BOOK MENU")
    print("1. Add a new recipe")
    print("2. View all recipes")
    print("3. Find recipe by ID")
    print("4. Exit")

if __name__ == "__main__":
    recipe_book = RecipeBook()

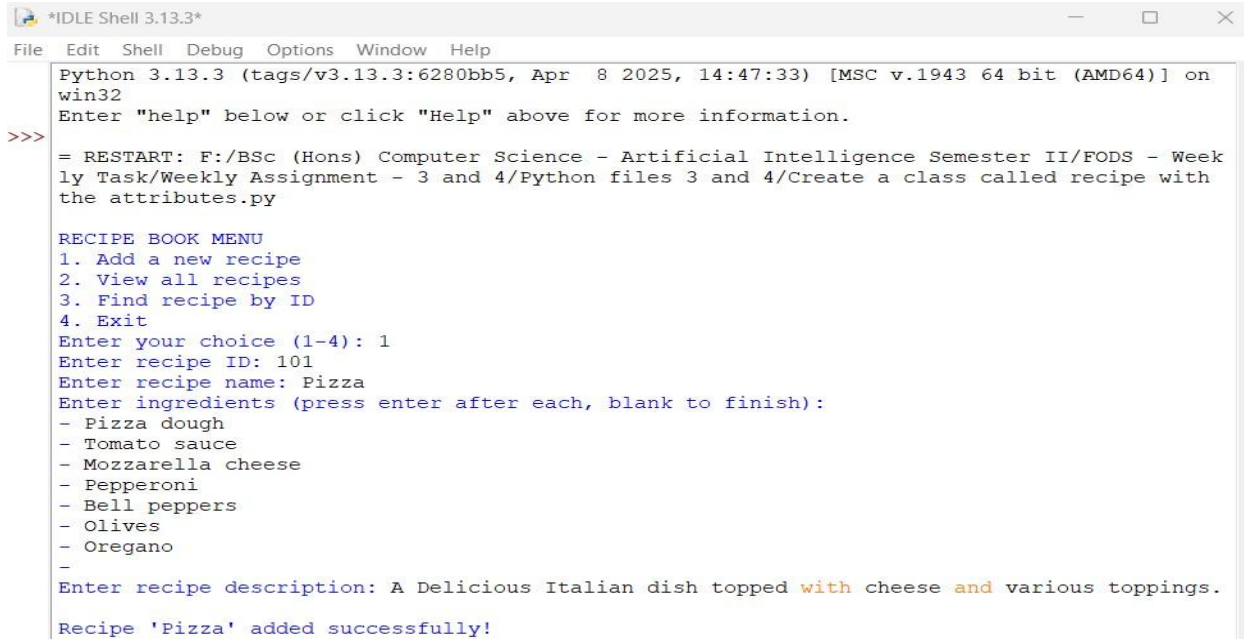
    while True:
        menu()
        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            recipe = get_recipe_input()
            recipe_book.add_recipe(recipe)
        elif choice == "2":
            recipe_book.display_all()
        elif choice == "3":
            recipe_id = input("Enter recipe ID to search: ")
            recipe_book.find_by_id(recipe_id)
        elif choice == "4":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

```

Output obtained in execution:

Adding New Recipe:



```

Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Create a class called recipe with the attributes.py

RECIPE BOOK MENU
1. Add a new recipe
2. View all recipes
3. Find recipe by ID
4. Exit
Enter your choice (1-4): 1
Enter recipe ID: 101
Enter recipe name: Pizza
Enter ingredients (press enter after each, blank to finish):
- Pizza dough
- Tomato sauce
- Mozzarella cheese
- Pepperoni
- Bell peppers
- Olives
- Oregano
-
Enter recipe description: A Delicious Italian dish topped with cheese and various toppings.
Recipe 'Pizza' added successfully!

```

Viewing All Recipe:

```

Recipe 'Pizza' added successfully!

RECIPE BOOK MENU
1. Add a new recipe
2. View all recipes
3. Find recipe by ID
4. Exit
Enter your choice (1-4): 2

===== RECIPE BOOK =====

Recipe ID: 101
Name: Pizza
Ingredients:
- Pizza dough
- Tomato sauce
- Mozzarella cheese
- Pepperoni
- Bell peppers
- Olives
- Oregano
Description: A Delicious Italian dish topped with cheese and various toppings.
-----

```


Finding a Recipe by ID:

```

RECIPE BOOK MENU
1. Add a new recipe
2. View all recipes
3. Find recipe by ID
4. Exit
Enter your choice (1-4): 3
Enter recipe ID to search: 101

Found recipe:

Recipe ID: 101
Name: Pizza
Ingredients:
- Pizza dough
- Tomato sauce
- Mozzarella cheese
- Pepperoni
- Bell peppers
- Olives
- Oregano
Description: A Delicious Italian dish topped with cheese and various toppings.

```

Python Program File: Saved the above code in a file named “Create a class called recipe with the attributes.py.”

Explanation of code:

Recipe Class:

It stores the essential information for the recipe (ID, name, ingredients list, description). It includes method ‘display()’ to show formatted recipe details.

RecipeBook Class:

It manages collections of Recipe objects and provided methods such as:

add_recipe(): Adds new recipe to the collection.

display_all(): Shows all the stored recipes.

find_by_id(): Searches for recipes by their ID.

Helper Functions:

The function ‘get_recipe_input()’ collects user input and create recipe objects. The ‘menu()’ function displays the programs.

Main Program:

It creates a RecipeBook instance and implements an interactive menu system for the user control. It handles all user interactions through simple numbered choices.

Output of the Program:

The program allows users to add a recipe for Pizza, view it, and search for it by ID. This showcases the functionality of the 'Recipe' and 'RecipeBook' classes. You can click exit to close the program with display a 'Goodbye!' message

Conclusion of the Program:

The given program is an example of how would it work with a recipe for a dish with the attributes such as id, name, ingredients, descriptions. It also creates another class called recipe book to manage the collection of recipes.

7. Write a program to implement a class called employee with attributes such as empid, name, address, contact_number, spouse_name, number_of_child, salary. Instantiate this class to input the values for multiple employees and write it in a file "employees.csv". Allow the user of your program to see the list of employees and their details as well. Try to use the concept of try/except too in the program.

Answer:

The given python program below implements an Employee class and writes the data to a CSV file, with error handling and user interaction in the program.

Following code for input:

```
Implement a class called employee with the attributes and write it in a file employees.csv file.py - F:/BSc (Hons) Computer Science - Artificial In...
File Edit Format Run Options Window Help
import csv
import os

class Employee:
    def __init__(self, empid, name, address, contact_number, spouse_name, number_of_child, salary):
        self.empid = empid
        self.name = name
        self.address = address
        self.contact_number = contact_number
        self.spouse_name = spouse_name
        self.number_of_child = number_of_child
        self.salary = salary

    def to_dict(self):
        """Convert employee details to a dictionary for CSV writing."""
        return {
            'Employee ID': self.empid,
            'Name': self.name,
            'Address': self.address,
            'Contact Number': self.contact_number,
            'Spouse Name': self.spouse_name,
            'Number of Children': self.number_of_child,
            'Salary': self.salary
        }

def write_employees_to_csv(employees, filename='employees.csv'):
    """Write employee data to a CSV file."""
    try:
        with open(filename, mode='w', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=employees[0].to_dict().keys())
            writer.writeheader()
            for employee in employees:
                writer.writerow(employee.to_dict())
        print(f"Employee data written to {filename} successfully.")
    except Exception as e:
        print(f"An error occurred while writing to the file: {e}")
```

Continue:

```
def display_employees(employees):
    """Display the list of employees."""
    if not employees:
        print("No employee records found.")
        return

    print("\nEmployee Details:")
    for employee in employees:
        print(f"ID: {employee.empid}, Name: {employee.name}, Address: {employee.address}, "
              f"Contact Number: {employee.contact_number}, Spouse Name: {employee.spouse_name}, "
              f"Number of Children: {employee.number_of_child}, Salary: {employee.salary}")

def get_employee_input():
    """Get employee details from user input."""
    empid = input("Enter Employee ID: ")
    name = input("Enter Name: ")
    address = input("Enter Address: ")
    contact_number = input("Enter Contact Number: ")
    spouse_name = input("Enter Spouse Name: ")
    number_of_child = input("Enter Number of Children: ")
    salary = input("Enter Salary: ")

    return Employee(empid, name, address, contact_number, spouse_name, number_of_child, salary)
```

Continue:

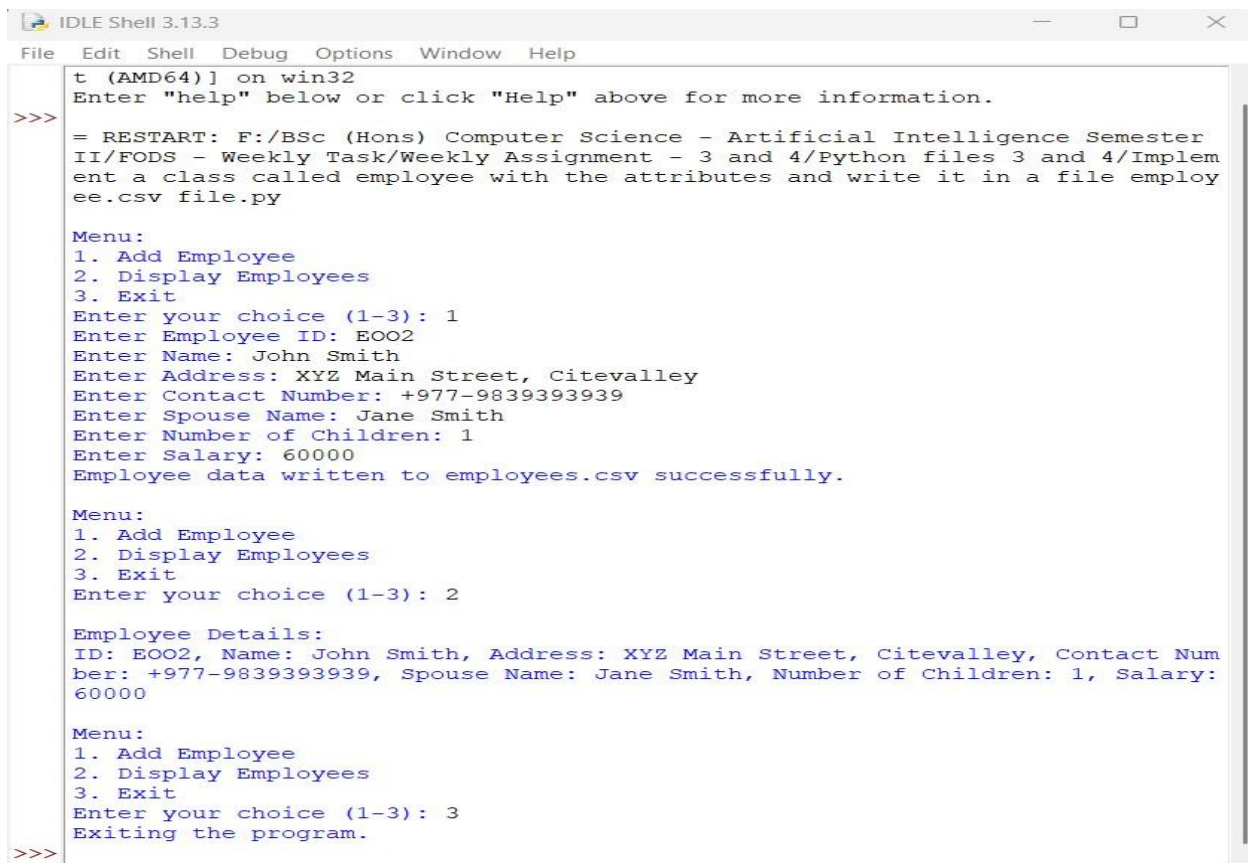
```
def main():
    employees = []

    while True:
        print("\nMenu:")
        print("1. Add Employee")
        print("2. Display Employees")
        print("3. Exit")
        choice = input("Enter your choice (1-3): ")

        if choice == '1':
            employee = get_employee_input()
            employees.append(employee)
            write_employees_to_csv(employees)
        elif choice == '2':
            display_employees(employees)
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

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

Output obtained in execution:



```
IDLE Shell 3.13.3
File Edit Shell Debug Options Window Help
t (AMD64) on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester
II/FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Implem
ent a class called employee with the attributes and write it in a file employ
ee.csv file.py

Menu:
1. Add Employee
2. Display Employees
3. Exit
Enter your choice (1-3): 1
Enter Employee ID: EOO2
Enter Name: John Smith
Enter Address: XYZ Main Street, Citevalley
Enter Contact Number: +977-9839393939
Enter Spouse Name: Jane Smith
Enter Number of Children: 1
Enter Salary: 60000
Employee data written to employees.csv successfully.

Menu:
1. Add Employee
2. Display Employees
3. Exit
Enter your choice (1-3): 2

Employee Details:
ID: EOO2, Name: John Smith, Address: XYZ Main Street, Citevalley, Contact Num
ber: +977-9839393939, Spouse Name: Jane Smith, Number of Children: 1, Salary:
60000

Menu:
1. Add Employee
2. Display Employees
3. Exit
Enter your choice (1-3): 3
Exiting the program.
>>>
```

Python Program File: Saved the above code in a file named “Implement a class called employee with the attributes and write it in a file employees.csv file.py.”

Explanation of code:

Imports:

The ‘csv’ module is imported to handle the CSV file operations. The ‘os’ module is imported but not in this code, it can be useful for the file path manipulations or checking the existence of the file.

Employee Class:

The ‘Employee’ class defines the structure for employee objects. The ‘__init__’ method initializes employee’s attributes, such as ID, name, address, contact number, spouse name, number of children, and salary. For writing a CSV file, the ‘to_dict’ method converts the employee’s attributes into a dictionary format.

Writing to CSV:

The function ‘write_employees_to_csv(employees, filename=‘employees.csv’)’ takes a list of ‘Employee’ objects and writes their details to a CSV file. A ‘try/except’ block is used to handle potential errors during file operations.

Displaying Employees:

The ‘display_employees(employees)’ function displays a readable format for the details of all employees. It also checks if the employee list is empty and prints a message if no records are found.

Getting Employee Input:

The ‘get_employee_input()’ function prompts the user to input details for a new employee and returns an ‘Employee’ object with the required data.

Main Function:

The 'main' function serves as the entry point of the program. It initializes an empty list to store employee objects. A loop presents a menu to the user, allowing them to add number of employees, display details, or exit the program.

Program Execution:

if __name__ == "__main__", this conditional statement checks if the script is being run directly and calls the 'main' function to start the program.

Output of the Program:

The program implements the 'Employee' class, allows user input for multiple employees, writes the data to the CSV file named 'employee.csv', and finally display the list of employees and their details.

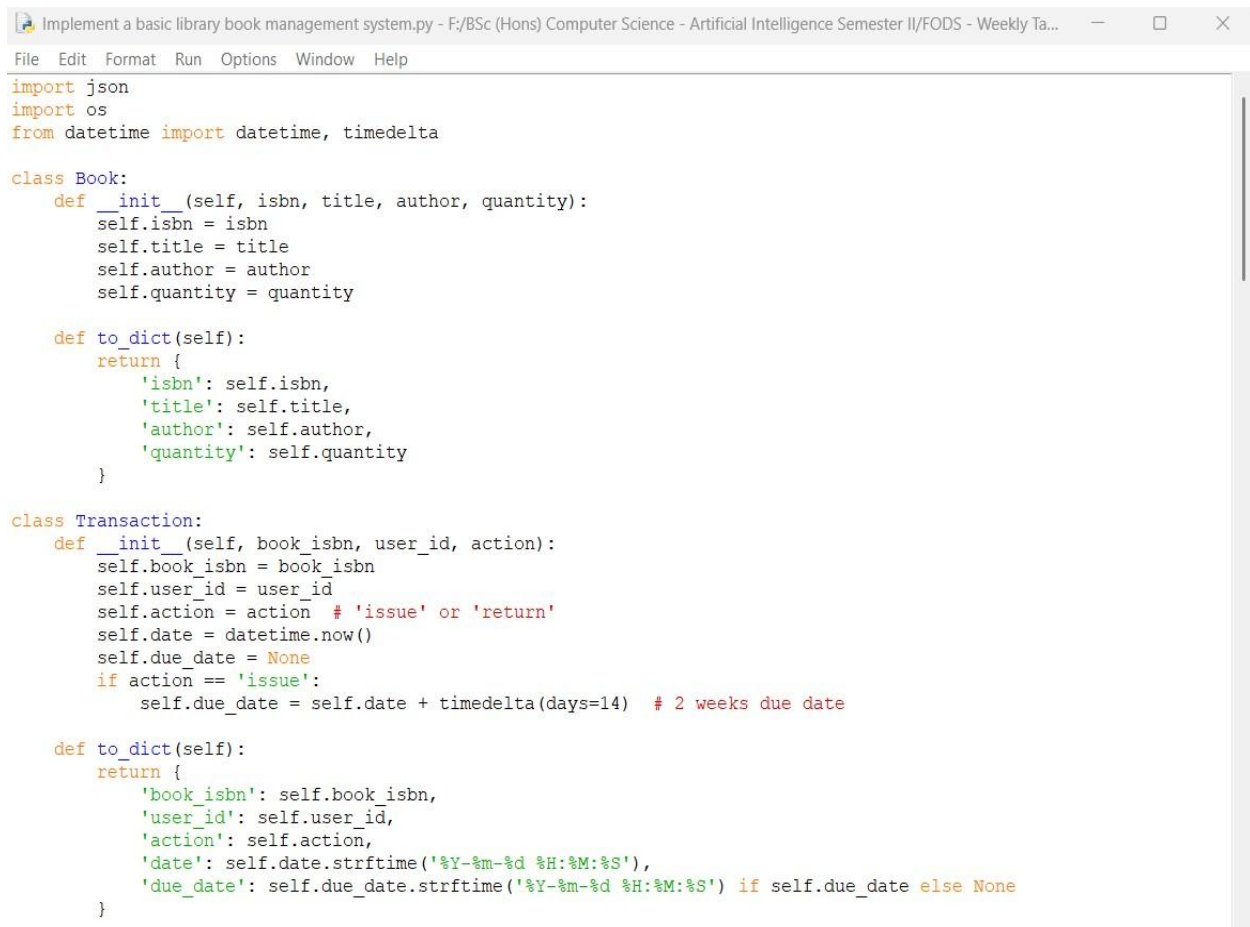
Conclusion of the Program:

The given program is an example of how to create the CSV file through your code, and it will be created automatically when you attempt to write to it.

8. Write a program to implement a basic library book management with the functionalities such as issue the book, return the book and search the book. Use the concept of OOP to create the necessary classes on your own and implement the concept of other OOP features. For the storage of book details, use the file handling along with the exception handling.

Answer:

The given python program below implements a library book management with the functionalities such as issue the book, return the book, and search the book. The program uses the concepts of OOP to create necessary classes and implements them in the program. File handling was used for the storage of book details with the exception handling.

Following code for input:


```
Implement a basic library book management system.py - F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/FODS - Weekly Ta...
File Edit Format Run Options Window Help

import json
import os
from datetime import datetime, timedelta

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

    def to_dict(self):
        return {
            'isbn': self.isbn,
            'title': self.title,
            'author': self.author,
            'quantity': self.quantity
        }

class Transaction:
    def __init__(self, book_isbn, user_id, action):
        self.book_isbn = book_isbn
        self.user_id = user_id
        self.action = action # 'issue' or 'return'
        self.date = datetime.now()
        self.due_date = None
        if action == 'issue':
            self.due_date = self.date + timedelta(days=14) # 2 weeks due date

    def to_dict(self):
        return {
            'book_isbn': self.book_isbn,
            'user_id': self.user_id,
            'action': self.action,
            'date': self.date.strftime('%Y-%m-%d %H:%M:%S'),
            'due_date': self.due_date.strftime('%Y-%m-%d %H:%M:%S') if self.due_date else None
        }
```

Continue:

```
class Library:
    BOOKS_FILE = 'books.json'
    TRANSACTIONS_FILE = 'transactions.json'

    def __init__(self):
        self.books = []
        self.transactions = []
        self._load_data()

    def _load_data(self):
        try:
            # Load books

            if os.path.exists(self.BOOKS_FILE):
                with open(self.BOOKS_FILE, 'r') as f:
                    books_data = json.load(f)
                    self.books = [Book(*data) for data in books_data]

            # Load transactions

            if os.path.exists(self.TRANSACTIONS_FILE):
                with open(self.TRANSACTIONS_FILE, 'r') as f:
                    transactions_data = json.load(f)
                    self.transactions = [Transaction(
                        data['book_isbn'],
                        data['user_id'],
                        data['action']
                    ) for data in transactions_data]
        except Exception as e:
            print(f"Error loading data: {e}")

    def _save_data(self):
        try:
            # Save books

            with open(self.BOOKS_FILE, 'w') as f:
                json.dump([book.to_dict() for book in self.books], f, indent=2)
```

Continue:

```
            # Save transactions

            with open(self.TRANSACTIONS_FILE, 'w') as f:
                json.dump([t.to_dict() for t in self.transactions], f, indent=2)
        except Exception as e:
            print(f"Error saving data: {e}")

    def add_book(self, isbn, title, author, quantity):
        for book in self.books:
            if book.isbn == isbn:
                book.quantity += quantity
                break
        else:
            self.books.append(Book(isbn, title, author, quantity))
        self._save_data()

    def issue_book(self, isbn, user_id):
        for book in self.books:
            if book.isbn == isbn and book.quantity > 0:
                book.quantity -= 1
                transaction = Transaction(isbn, user_id, 'issue')
                self.transactions.append(transaction)
                self._save_data()
                return transaction.due_date
        return None

    def return_book(self, isbn, user_id):
        for book in self.books:
            if book.isbn == isbn:
                book.quantity += 1
                transaction = Transaction(isbn, user_id, 'return')
                self.transactions.append(transaction)
                self._save_data()
                return True
        return False
```


Continue:

```
def search_book(self, search_term):
    results = []
    search_term = search_term.lower()
    for book in self.books:
        if (search_term in book.isbn.lower() or
            search_term in book.title.lower() or
            search_term in book.author.lower()):
            results.append(book)
    return results

def get_book_status(self, isbn):
    for book in self.books:
        if book.isbn == isbn:
            issued_count = sum(
                1 for t in self.transactions
                if t.book_isbn == isbn and t.action == 'issue' and (
                    not any(
                        rt.book_isbn == isbn and rt.action == 'return'
                        for rt in self.transactions
                        if rt.date > t.date
                    )
                )
            )
            return {
                'total': book.quantity + issued_count,
                'available': book.quantity,
                'issued': issued_count
            }
    return None
```

Continue:

```
def display_menu():
    print("\nLibrary Management System")
    print("1. Add Book")
    print("2. Issue Book")
    print("3. Return Book")
    print("4. Search Book")
    print("5. Check Book Status")
    print("6. Exit")

def main():
    library = Library()

    # Add some sample books if the database is empty
    if not library.books:
        library.add_book("978-3-16-148410-0", "Python Programming", "John Smith", 5)
        library.add_book("978-1-23-456789-0", "Advanced Python", "Jane Doe", 3)

    while True:
        display_menu()
        choice = input("Enter your choice (1-6): ")

        if choice == '1':
            try:
                isbn = input("Enter ISBN: ")
                title = input("Enter Title: ")
                author = input("Enter Author: ")
                quantity = int(input("Enter Quantity: "))
                library.add_book(isbn, title, author, quantity)
                print("Book added successfully!")
            except ValueError:
                print("Invalid quantity. Please enter a number.")
```


Continue:

```
def main():
    library = Library()

    # Add some sample books if the database is empty

    if not library.books:
        library.add_book("978-3-16-148410-0", "Python Programming", "John Smith", 5)
        library.add_book("978-1-23-456789-0", "Advanced Python", "Jane Doe", 3)

    while True:
        display_menu()
        choice = input("Enter your choice (1-6): ")

        if choice == '1':
            try:
                isbn = input("Enter ISBN: ")
                title = input("Enter Title: ")
                author = input("Enter Author: ")
                quantity = int(input("Enter Quantity: "))
                library.add_book(isbn, title, author, quantity)
                print("Book added successfully!")
            except ValueError:
                print("Invalid quantity. Please enter a number.")
```

Continue:

```
        elif choice == '2':
            isbn = input("Enter ISBN of book to issue: ")
            user_id = input("Enter your User ID: ")
            due_date = library.issue_book(isbn, user_id)
            if due_date:
                print(f"Book issued successfully! Due date: {due_date.strftime('%Y-%m-%d')}")
            else:
                print("Book not available or invalid ISBN.")

        elif choice == '3':
            isbn = input("Enter ISBN of book to return: ")
            user_id = input("Enter your User ID: ")
            if library.return_book(isbn, user_id):
                print("Book returned successfully!")
            else:
                print("Invalid ISBN or user ID.")

        elif choice == '4':
            search_term = input("Enter search term (ISBN, Title, or Author): ")
            results = library.search_book(search_term)
            if results:
                print("\nSearch Results:")
                for book in results:
                    print(f"ISBN: {book.isbn}, Title: {book.title}, "
                          f"Author: {book.author}, Available: {book.quantity}")
            else:
                print("No books found matching your search.")

        elif choice == '5':
            isbn = input("Enter ISBN to check status: ")
            status = library.get_book_status(isbn)
            if status:
                print(f"\nBook Status:")
                print(f"Total Copies: {status['total']}")
                print(f"Available: {status['available']}")
                print(f"Issued: {status['issued']}")
            else:
                print("Book not found.")

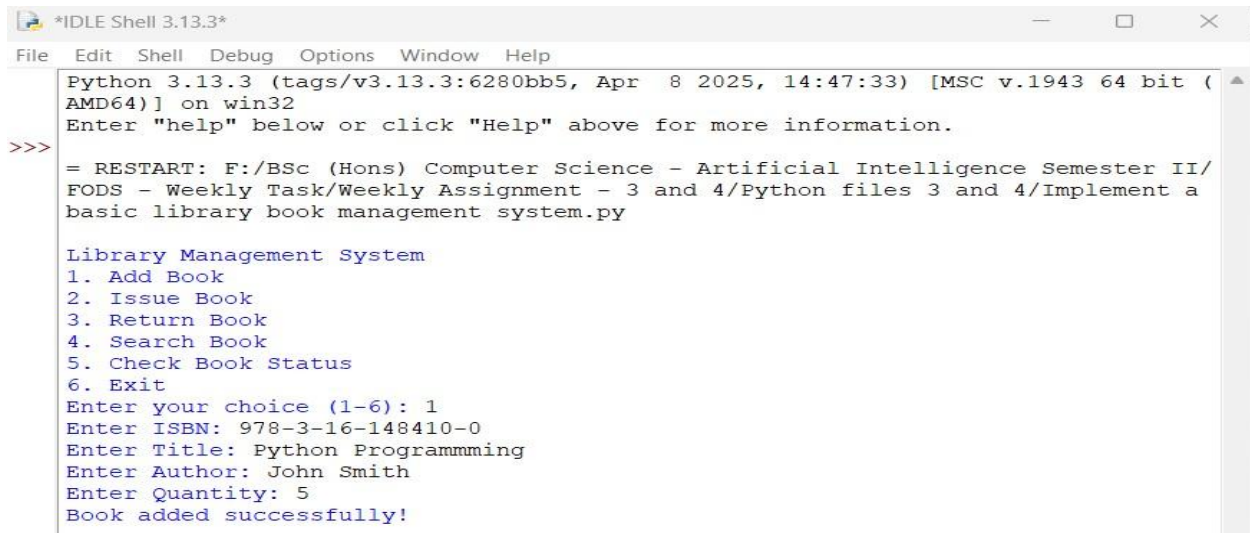
        elif choice == '6':
            print("Exiting Library Management System. Goodbye!")
            break

        else:
            print("Invalid choice. Please try again.")

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

Output obtained in execution:

Starting the Program and Adding a Book:



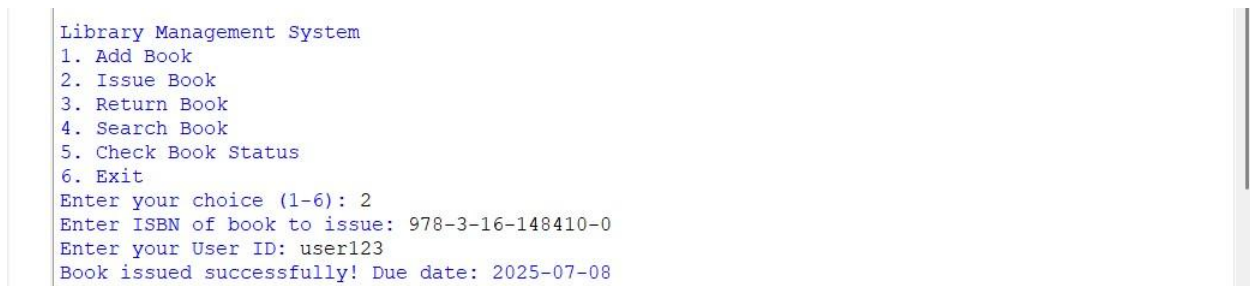
```

Python 3.13.3 (tags/v3.13.3:6280bb5, Apr  8 2025, 14:47:33) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: F:/BSc (Hons) Computer Science - Artificial Intelligence Semester II/
FODS - Weekly Task/Weekly Assignment - 3 and 4/Python files 3 and 4/Implement a
basic library book management system.py

Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 1
Enter ISBN: 978-3-16-148410-0
Enter Title: Python Programming
Enter Author: John Smith
Enter Quantity: 5
Book added successfully!

```

Issuing a Book:




```

Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 2
Enter ISBN of book to issue: 978-3-16-148410-0
Enter your User ID: user123
Book issued successfully! Due date: 2025-07-08

```

Returning a Book:



```

Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 3
Enter ISBN of book to return: 978-3-16-148410-0
Enter your User ID: user123
Book returned successfully!

```

Searching a Book:

```
Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 4
Enter search term (ISBN, Title, or Author): Python

Search Results:
ISBN: 978-3-16-148410-0, Title: Python Programming, Author: John Smith, Available: 10
ISBN: 978-1-23-456789-0, Title: Advanced Python, Author: Jane Doe, Available: 3
```

Checking Book Status:

```
Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 5
Enter ISBN to check status: 978-3-16-148410-0

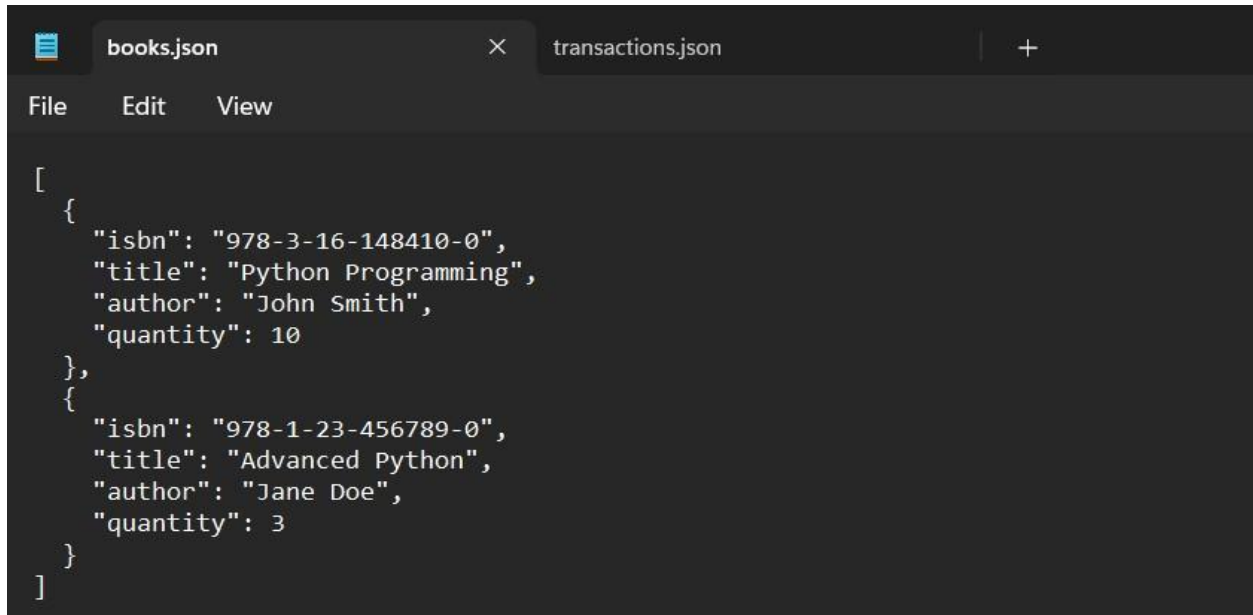
Book Status:
Total Copies: 10
Available: 10
Issued: 0
```

Exiting the Program:

```
Library Management System
1. Add Book
2. Issue Book
3. Return Book
4. Search Book
5. Check Book Status
6. Exit
Enter your choice (1-6): 6
Exiting Library Management System. Goodbye!
>>>
```

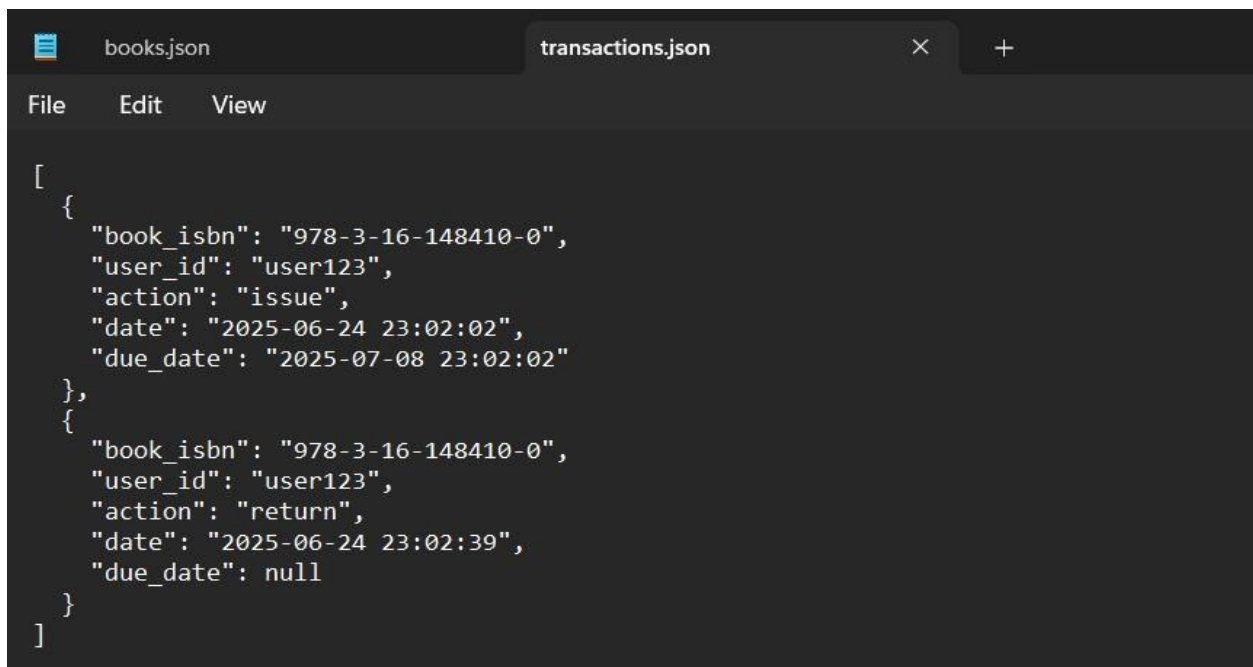
JSON Files Created:

books.json:

A screenshot of a code editor with two tabs: 'books.json' and 'transactions.json'. The 'books.json' tab is active. The editor shows a JSON array with two book objects. The first object has isbn '978-3-16-148410-0', title 'Python Programming', author 'John Smith', and quantity 10. The second object has isbn '978-1-23-456789-0', title 'Advanced Python', author 'Jane Doe', and quantity 3.

```
[
  {
    "isbn": "978-3-16-148410-0",
    "title": "Python Programming",
    "author": "John Smith",
    "quantity": 10
  },
  {
    "isbn": "978-1-23-456789-0",
    "title": "Advanced Python",
    "author": "Jane Doe",
    "quantity": 3
  }
]
```

transactions.json:

A screenshot of a code editor with two tabs: 'books.json' and 'transactions.json'. The 'transactions.json' tab is active. The editor shows a JSON array with two transaction objects. The first object has book_isbn '978-3-16-148410-0', user_id 'user123', action 'issue', date '2025-06-24 23:02:02', and due_date '2025-07-08 23:02:02'. The second object has book_isbn '978-3-16-148410-0', user_id 'user123', action 'return', date '2025-06-24 23:02:39', and due_date null.

```
[
  {
    "book_isbn": "978-3-16-148410-0",
    "user_id": "user123",
    "action": "issue",
    "date": "2025-06-24 23:02:02",
    "due_date": "2025-07-08 23:02:02"
  },
  {
    "book_isbn": "978-3-16-148410-0",
    "user_id": "user123",
    "action": "return",
    "date": "2025-06-24 23:02:39",
    "due_date": null
  }
]
```

Python Program File: Saved the above code in a file named “Implement a basic library management system.py.”

Explanation of code:

Imports:

json: Used for reading and writing the JSON files to store book and transaction data.

Os: Provided functions to interact with the operating system, such as if the file exists or not.

Datetime: Used to handle data and time, particularly for managing due dates for issued books.

Book Class:

It represents a book with attributes such as ISBN, author, and quantity. The ‘to_dict’ method converts the book’s attributes into a dictionary format for easy JSON serialization.

Transaction Class:

Class named ‘Transaction’ for issuing or returning a book. It contains attributes for the book’s ISBN, user ID, action (issue or return), transaction date, and due date. The method ‘to_dict’ formats the transaction data for JSON serialization.

Library Class:

This is the main class that manages the library’s operations. It establishes constants for the JSON file names used to store the book and the transaction data. It initializes an empty list for the books and transactions and loads existing data from files.

Loading Data:

```
def _load_data(self):
```

This method checks for the existence of the JSON files and loads their contents into the 'books' and 'transaction' files. Exception handling to catch and report any errors that occur during the file operations.

Saving Data:

```
def _save_data(self):
```

This method saves the current state of books and transactions to their respective JSON files. It converts the objects to dictionaries using 'to_dict' method for serialization. Again, the exception handling is used to manage potential errors during file writing.

Adding a Book:

```
def add_book(self, isbn, title, author, quantity):
```

This method adds a new book to the library or increases the quantity of an existing book if the ISBN matches. Then, calls '_save_data' to save changes.

Issuing a Book:

```
def issue_book(self, isbn, user_id):
```

This method issues a book to a user if it is available (quantity > 0). It creates a new 'Transaction' object for the issue and appends it to the transactions list. The due date returns for the issued book.

Returning a Book:

```
def return_book(self, isbn, user_id):
```

This method allows to return a book by increasing its quantity for the user. It creates a new 'Transaction' object for the return list and appends it to the transaction list.

Searching for a Book:

```
def search_book(self, isbn, user_id):
```

This method searches for books based on a search term that can match attributes like ISBN, title, or author. It returns a list of matching 'Book' objects.

Getting Book Status:

```
def get_book_status(self, isbn):
```

This method checks the status of a book by its ISBN and calculates the total number of copies, available copies, and how many are currently issued. These information returns a dictionary.

Display Menu Function:

```
def display_menu():
```

A simple function to display the main menu options for the user.

Main Function:

The 'main' function initializes the 'Library' object and provides a loop for user interaction. It displays the menu and processes user input for various operations (adding, issuing, retuning searching, and checking status). Error handling for invalid inputs present, such as non-numeric quantities.

Program Execution:

```
if __name__ == "__main__":
```

This conditional statement checks if the script is being run directly and calls the main function to start the program.

Output of the Program:

The program implements the various functionalities such as adding books, issuing books, retuning books, searching for books, and checking book status.

Conclusion of the Program:

The given program is an example of how to create a basic library book management with the functionalities, providing a solution for managing books in library. It creates JSON files which does not need to be created manually. This system demonstrates the principles of object-oriented programming language, file handling, and user interaction.