- Exercises

Rules

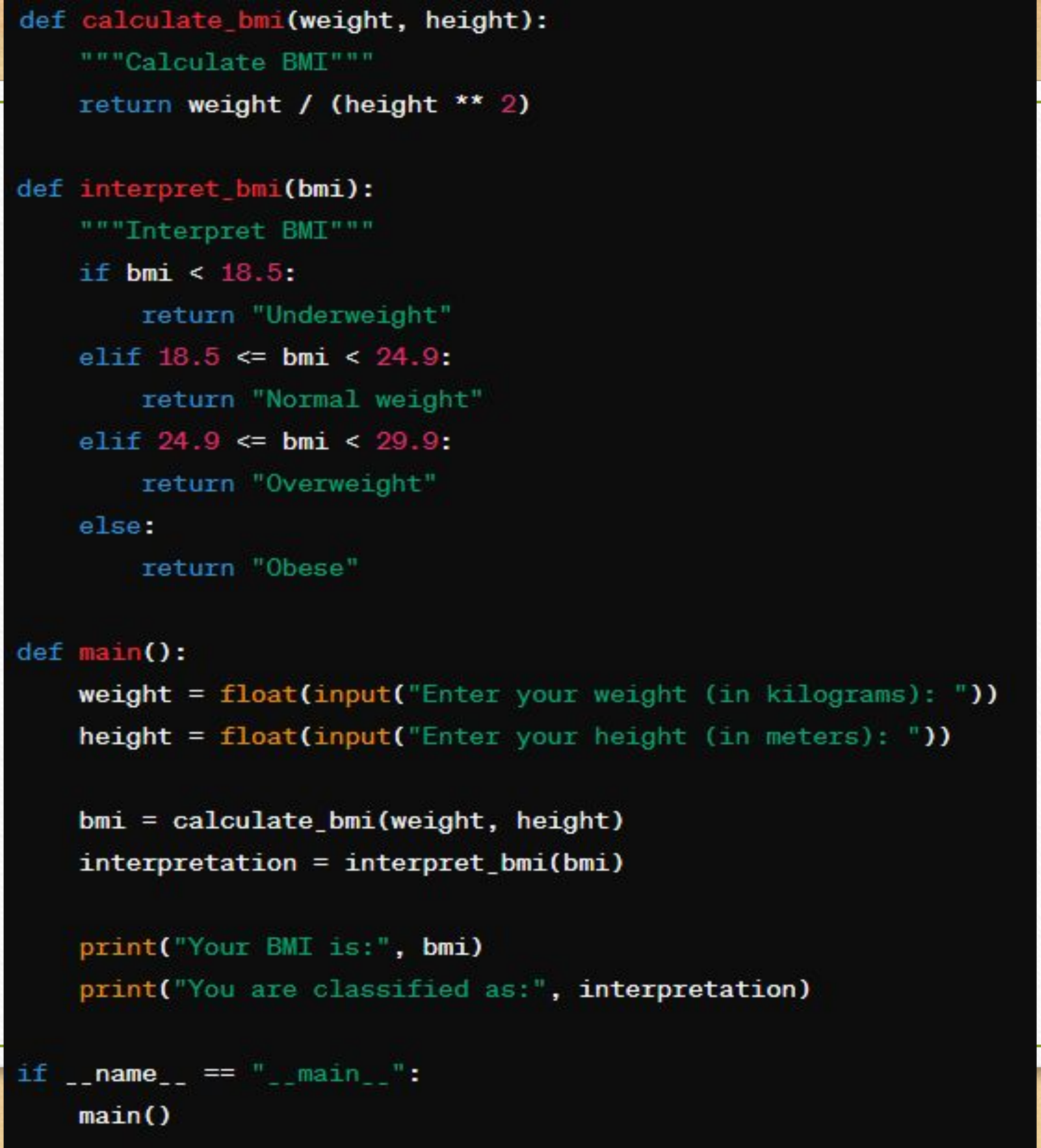
- No Chatgpt
- No questions and No assistance from others
    - Self-learning capability
    - You need to learn how to solve complex problems on your own when faced with complex problems. For example, how to quickly find solutions online
    - The task may beyond the scope of your knowledge, Try.
- Can check online resources or lecture notes
- Solutions will be given later

BMI Calculator and Interpretation

Requirements:

- Build a BMI (Body Mass Index) calculator that computes the BMI score based on a person's weight and height.
- Use conditional statements to interpret the BMI score into categories such as Underweight, Normal weight, Overweight, and Obese.
- Set the BMI classification thresholds as follows:

  - Underweight: less than 18.5
  - Normal weight: 18.5 to 24.9
  - Overweight: 25 to 29.9
  - Obese: 30 or more
- Print out the person's BMI score and interpretation.



```python
def calculate_bmi(weight, height):
    """Calculate BMI"""
    return weight / (height ** 2)

def interpret_bmi(bmi):
    """Interpret BMI"""
    if bmi < 18.5:
        return "Underweight"
    elif 18.5 <= bmi < 24.9:
        return "Normal weight"
    elif 24.9 <= bmi < 29.9:
        return "Overweight"
    else:
        return "Obese"

def main():
    weight = float(input("Enter your weight (in kilograms): "))
    height = float(input("Enter your height (in meters): "))

    bmi = calculate_bmi(weight, height)
    interpretation = interpret_bmi(bmi)

    print("Your BMI is:", bmi)
    print("You are classified as:", interpretation)

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

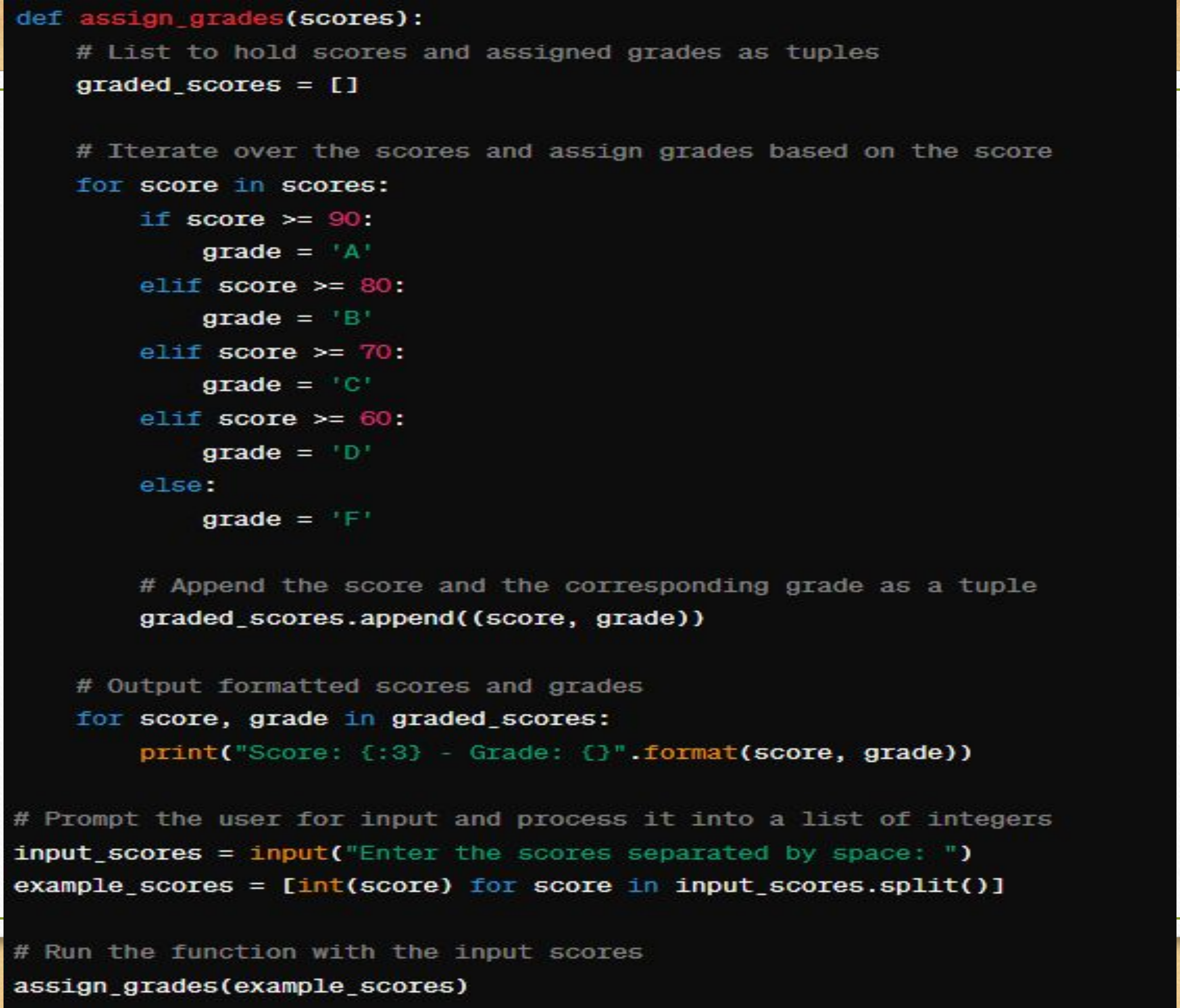
•Exercise2： **Grade Classifier**

Objective: Create a program that takes students' scores as input and assigns a grade based on the score. The grades should be A, B, C, D, or F.

Requirements:

- Ask for user input(format: [score1,score2,score3,…])
- Utilize a list to store scores and their corresponding grades.
- Iterate over the list of scores using a loop.
- Use comparison operators within conditional statements to determine the appropriate grade for each score.
- Print each student's score (keep 1 place after point) along with their respective grade.

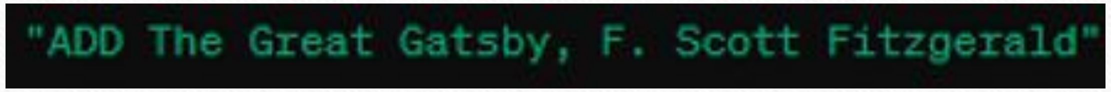
A: 90 and above
B: 80 to 89
C: 70 to 79
D: 60 to 69
F: below 60



```python
def assign_grades(scores):
    # List to hold scores and assigned grades as tuples
    graded_scores = []

    # Iterate over the scores and assign grades based on the score
    for score in scores:
        if score >= 90:
            grade = 'A'
        elif score >= 80:
            grade = 'B'
        elif score >= 70:
            grade = 'C'
        elif score >= 60:
            grade = 'D'
        else:
            grade = 'F'

        # Append the score and the corresponding grade as a tuple
        graded_scores.append((score, grade))

    # Output formatted scores and grades
    for score, grade in graded_scores:
        print("Score: {:3} - Grade: {}".format(score, grade))

# Prompt the user for input and process it into a list of integers
input_scores = input("Enter the scores separated by space: ")
example_scores = [int(score) for score in input_scores.split()]

# Run the function with the input scores
assign_grades(example_scores)
```

Exercise3： **Simple Book Management System**

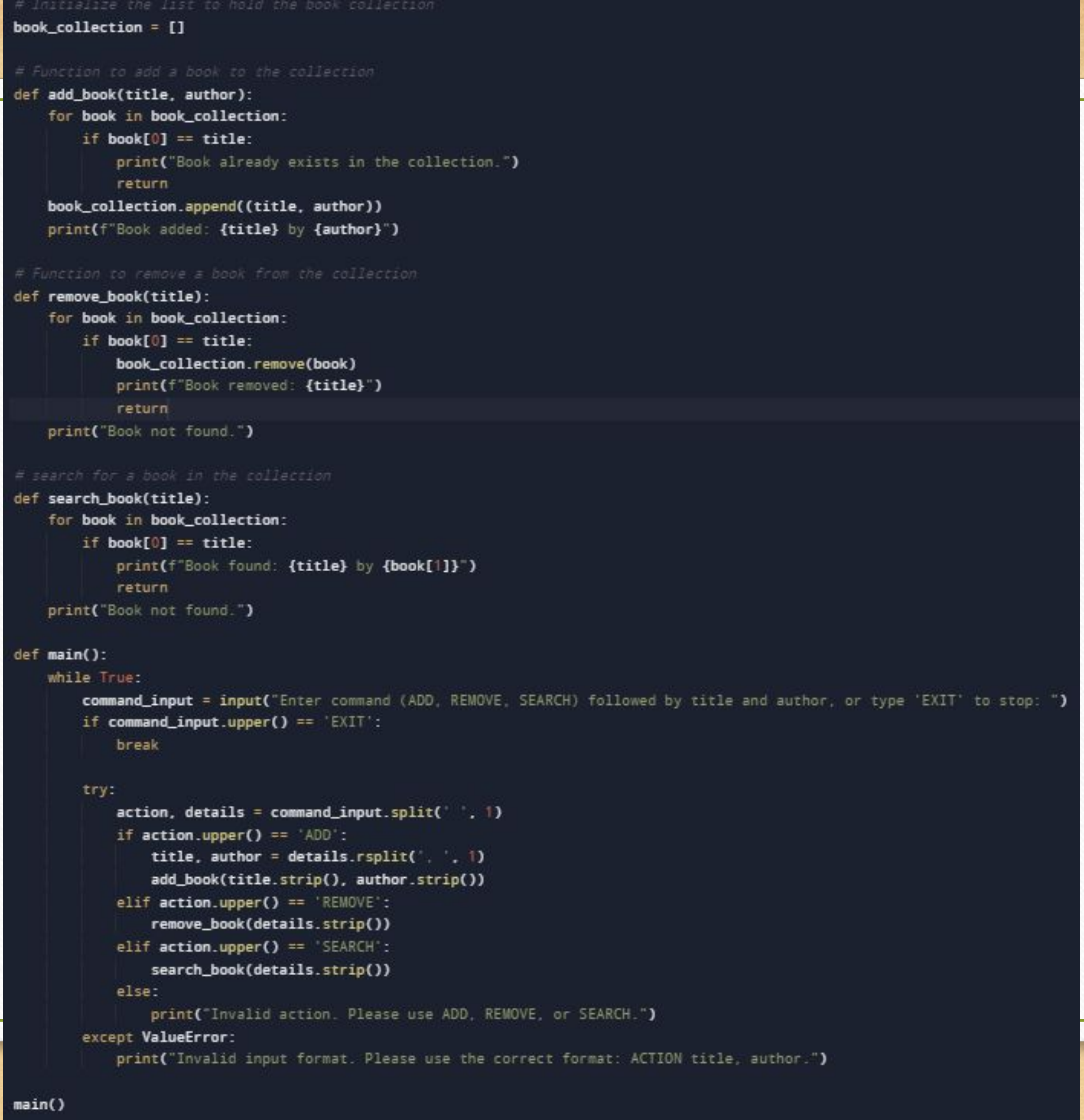
Objective: Write a program to help users manage their personal book collection. The program should allow the user to add, remove, and search for books.



```
"ADD The Great Gatsby, F. Scott Fitzgerald"
```

Requirements :

- **User Input**: The user will input commands like "ADD title, author", "REMOVE title", or "SEARCH title".
- **Book List**: The program should maintain a list of books, where each book is represented by a dictionary containing the book's title and author.
- **Adding Books**: When adding a book, the program should check to see if the book already exists in the collection.
- **Removing Books**: When removing a book, the program should verify that the book is in the list.
- **Searching for Books**: When searching for a book, if found, the program should display "Book found: title by author". If the book is not found, it should display "Book not found".
- **Error Handling**: If the user enters an incorrect command format, the program should prompt them with "Invalid input. Please use ADD, REMOVE, or SEARCH followed by the book title and author."



```python
# Initialize the list to hold the book collection
book_collection = []

# Function to add a book to the collection
def add_book(title, author):
    for book in book_collection:
        if book[0] == title:
            print("Book already exists in the collection.")
            return
    book_collection.append((title, author))
    print(f"Book added: {title} by {author}")

# Function to remove a book from the collection
def remove_book(title):
    for book in book_collection:
        if book[0] == title:
            book_collection.remove(book)
            print(f"Book removed: {title}")
            return
    print("Book not found.")

# search for a book in the collection
def search_book(title):
    for book in book_collection:
        if book[0] == title:
            print(f"Book found: {title} by {book[1]}")
            return
    print("Book not found.")

def main():
    while True:
        command_input = input("Enter command (ADD, REMOVE, SEARCH) followed by title and author, or type 'EXIT' to stop: ")
        if command_input.upper() == 'EXIT':
            break

        try:
            action, details = command_input.split(' ', 1)
            if action.upper() == 'ADD':
                title, author = details.rsplit(', ', 1)
                add_book(title.strip(), author.strip())
            elif action.upper() == 'REMOVE':
                remove_book(details.strip())
            elif action.upper() == 'SEARCH':
                search_book(details.strip())
            else:
                print("Invalid action. Please use ADD, REMOVE, or SEARCH.")
        except ValueError:
            print("Invalid input format. Please use the correct format: ACTION title, author.")

main()
```

Exercise4 : Expense Tracker

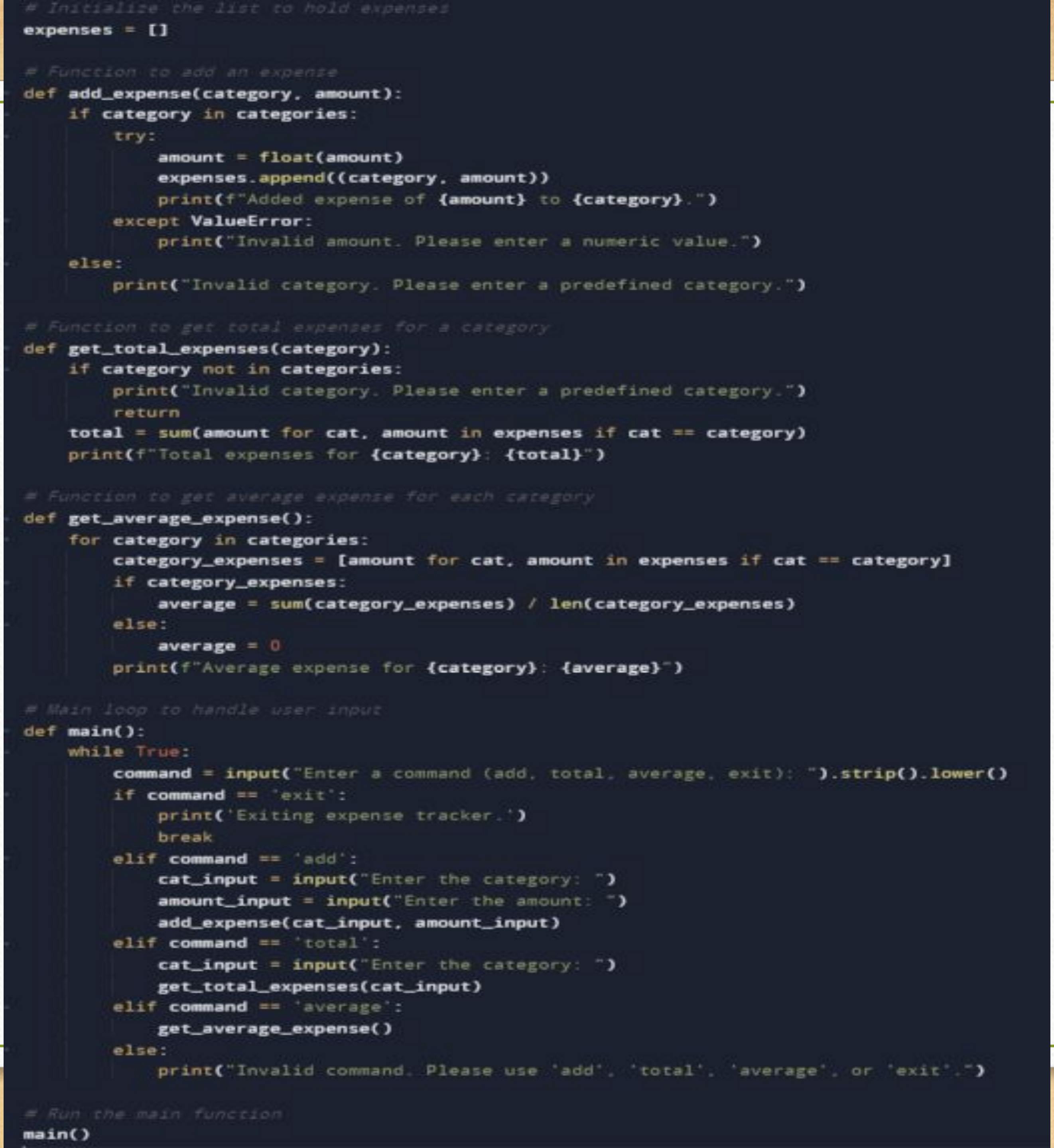
Objective: Create a program to help users manage and analyze their personal expenses by categories over a month.

Refined Requirements:

- The program should have predefined categories: 'Food', 'Utilities', 'Entertainment', 'Transportation', 'Healthcare'.
- The user can add expenses by specifying a category and an amount.
- The user can request the total expenses for a specific category.
- The user can request the average expense for each category.
- The program should prevent the user from entering expenses into undefined categories.

Features to Use:

- Dictionary with predefined categories as keys, and the values as lists that store expenses.
- Functions for:
    - Adding expenses to categories
    - Calculating total expenses for a specific category
    - Calculating total and average expenses for all categories
- Input validation to ensure correct category usage.
- Exception handling for invalid inputs (e.g., non-numeric expense amounts).



```python
# Initialize the list to hold expenses
expenses = []

# Function to add an expense
def add_expense(category, amount):
    if category in categories:
        try:
            amount = float(amount)
            expenses.append((category, amount))
            print(f"Added expense of {amount} to {category}.")
        except ValueError:
            print("Invalid amount. Please enter a numeric value.")
    else:
        print("Invalid category. Please enter a predefined category.")

# Function to get total expenses for a category
def get_total_expenses(category):
    if category not in categories:
        print("Invalid category. Please enter a predefined category.")
        return
    total = sum(amount for cat, amount in expenses if cat == category)
    print(f"Total expenses for {category}: {total}")

# Function to get average expense for each category
def get_average_expense():
    for category in categories:
        category_expenses = [amount for cat, amount in expenses if cat == category]
        if category_expenses:
            average = sum(category_expenses) / len(category_expenses)
        else:
            average = 0
        print(f"Average expense for {category}: {average}")

# Main loop to handle user input
def main():
    while True:
        command = input("Enter a command (add, total, average, exit): ").strip().lower()
        if command == 'exit':
            print('Exiting expense tracker.')
            break
        elif command == 'add':
            cat_input = input("Enter the category: ")
            amount_input = input("Enter the amount: ")
            add_expense(cat_input, amount_input)
        elif command == 'total':
            cat_input = input("Enter the category: ")
            get_total_expenses(cat_input)
        elif command == 'average':
            get_average_expense()
        else:
            print("Invalid command. Please use 'add', 'total', 'average', or 'exit'.")

# Run the main function
main()
```

- Thank you