AI Assisted Coding

Assignment Lab - 5.1 and 6.1

Name : G.Abhiram
Ht No : 2303A51087
Batch No. : 02

Lab 5: Ethical Foundations – Responsible AI Coding Practices Lab Objectives:

• To explore the ethical risks associated with AI-generated code.

• To recognize issues related to security, bias, transparency, and copyright.

• To reflect on the responsibilities of developers when using AI tools in software development.

• To promote awareness of best practices for responsible and ethical AI coding.

Task 1:

Employee Data: Create Python code that defines a class named `Employee` with the following attributes: `empid`, `empname`, `designation`, `basic_salary`, and `exp`. Implement a method `display_details()` to print all employee details. Implement another method `calculate_allowance()` to determine additional allowance
based on experience:

- If `exp > 10 years` → allowance = 20% of `basic_salary`
- If `5 ≤ exp ≤ 10 years` → allowance = 10% of `basic_salary`
- If `exp < 5 years` → allowance = 5% of `basic_salary`

Finally, create at least one instance of the `Employee` class, call the `display_details()` method, and print the calculated allowance.

Copilot Suggested Code :

```python
# Employee data

class Employee:
    def __init__(self, emp_id, emp_name, designation, emp_salary, emp_exp):
        self.emp_id = emp_id
        self.emp_name = emp_name
        self.designation = designation
        self.emp_salary = emp_salary
        self.emp_exp = emp_exp
    def display_details(self):
        print(f"Employee ID: {self.emp_id}")
        print(f"Employee Name: {self.emp_name}")
        print(f"Designation: {self.designation}")
        print(f"Salary: {self.emp_salary}")
        print(f"Experience: {self.emp_exp} years")
    def calculate_allowence(self):
        if self.emp_exp > 10:
            allowence = 0.1 * self.emp_salary
        elif 5 < self.emp_exp <= 10:
            allowence = 0.05 * self.emp_salary
        else:
            allowence = 0.005 * self.emp_salary
        print(f"Allowence: {allowence}")

emp = Employee(101, "John", "Manager", 15000, 12)
emp.display_details()
emp.calculate_allowence()
emp2 = Employee(102, "Alice", "Developer", 8000, 7)
emp2.display_details()
emp2.calculate_allowence()
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 1.py"
Employee ID: 101
Employee Name: John
Designation: Manager
Salary: 15000
Experience: 12 years
Allowence: 1500.0
Employee ID: 102
Employee Name: Alice
Designation: Developer
Salary: 8000
Experience: 7 years
Allowence: 400.0
PS C:\Btech\3 - 2\Ai AC>
```

Task 2:

Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit
- 101 to 300 units → ₹7 per unit
- More than 300 units → ₹10 per unit

Create a bill object, display details, and print the total bill amount.

Copilot Suggested Code :

```python
# Electricity Bill

class ElectricityBill:
    def __init__(self, cus_id, name, units_consumed):
        self.cus_id = cus_id
        self.name = name
        self.units_consumed = units_consumed
        self.bill_amount = 0
    def display_details(self):
        print(f"Customer ID: {self.cus_id}")
        print(f"Customer Name: {self.name}")
        print(f"Units Consumed: {self.units_consumed}")
        print(f"Bill Amount: {self.bill_amount}")
    def calculate_bill(self):
        if self.units_consumed <= 100:
            self.bill_amount = self.units_consumed * 5
        elif 101 <= self.units_consumed <= 300:
            self.bill_amount = 100 * 5 + (self.units_consumed - 100) * 7
        else:
            self.bill_amount = 100 * 5 + 200 * 7 + (self.units_consumed - 300) * 10
        print(f"Total Bill Amount: ₹{self.bill_amount}")

bill = ElectricityBill(1001, "Rahul", 350)
bill.display_details()
bill.calculate_bill()

bill2 = ElectricityBill(1002, "Priya", 250)
bill2.display_details()
bill2.calculate_bill()
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 2.py"
Customer ID: 1001
Customer Name: Rahul
Units Consumed: 350
Bill Amount: 0
Total Bill Amount: ₹2400
Customer ID: 1002
Customer Name: Priya
Units Consumed: 250
Bill Amount: 0
Total Bill Amount: ₹1550
PS C:\Btech\3 - 2\Ai AC>
```

Task 3:

Product Discount Calculation- Create Python code that defines a class named `Product` with attributes: `product_id`, `product_name`, `price`, and `category`. Implement a method `display_details()` to print product details. Implement another method
`calculate_discount()` where:

- Electronics → 10% discount
- Clothing → 15% discount
- Grocery → 5% discount

Create at least one product object, display details, and print the final price after discount.

Copilot Suggested Code :

```python
# Product Discount Calculation

class Product:
    def __init__(self, product_id, product_name, category):
        self.product_id = product_id
        self.product_name = product_name
        self.category = category
        self.discount = 0
    def display_details(self):
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Discount: {self.discount}%")
    def calculate_discount(self):
        if self.category == "Electronics":
            self.discount = 10
        elif self.category == "Clothing":
            self.discount = 15
        elif self.category == "Grocery":
            self.discount = 5

        print(f"Final price after discount: {self.discount}%")

Product1 = Product(1, "Laptop", "Electronics")
Product1.display_details()
Product1.calculate_discount()

Product2 = Product(2, "T-Shirt", "Clothing")
Product2.display_details()
Product2.calculate_discount()
```

Output :

```
● PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 3.py"
  Product ID: 1
  Product Name: Laptop
  Category: Electronics
  Discount: 0%
  Final price after discount: 10%
  Product ID: 2
  Product Name: T-Shirt
  Category: Clothing
  Discount: 0%
  Final price after discount: 15%
○ PS C:\Btech\3 - 2\Ai AC> 
```

Task 4:

Book Late Fee Calculation- Create Python code that defines a class named \`LibraryBook\` with attributes: \`book_id\`, \`title\`, \`author\`, \`borrower\`, and \`days_late\`. Implement a method \`display_details()\` to print book details, and a method \`calculate_late_fee()\` where:

- Days late ≤ 5 → ₹5 per day
- 6 to 10 days late → ₹7 per day
- More than 10 days late → ₹10 per day

Create a book object, display details, and print the late fee.

Copilot Suggested Code :

```python
# Book Late Fee Calculation

class LibraryBook:
    def __init__(self, book_id, title, author, borrower, days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.borrower = borrower
        self.days_late = days_late
        self.late_fee = 0

    def display_details(self):
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Borrower: {self.borrower}")
        print(f"Days Late: {self.days_late}")
        print(f"Late Fee: ₹{self.late_fee}")

    def calculate_late_fee(self):
        if self.days_late <= 5:
            self.late_fee = self.days_late * 5
        elif self.days_late <= 10:
            self.late_fee = 5 * 5 + (self.days_late - 5) * 7
        else:
            self.late_fee = 5 * 5 + 5 * 7 + (self.days_late - 10) * 10
        print(f"Late Fee calculated: ₹{self.late_fee}")

book = LibraryBook(101, "The Great Gatsby", "F. Scott Fitzgerald", "John Doe", 7)
book.display_details()
book.calculate_late_fee()

book2 = LibraryBook(102, "To Kill a Mockingbird", "Harper Lee", "Jane Smith", 12)
book2.display_details()
book2.calculate_late_fee()
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 4.py"
Book ID: 101
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: John Doe
Days Late: 7
Late Fee: ₹0
Late Fee calculated: ₹39
Book ID: 102
Title: To Kill a Mockingbird
Author: Harper Lee
Borrower: Jane Smith
Days Late: 12
Late Fee: ₹0
Late Fee calculated: ₹80
PS C:\Btech\3 - 2\Ai AC>
```

Task 5:

Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks.

The function should:

- Calculate the average score for each student
- Determine pass/fail status (pass ≥ 40)
- Return a summary report as a list of dictionaries

Use Copilot suggestions as you build the function and format the output.

Copilot Suggested Code :

```python
# Student Performance Report

def student_report(student_data):
    report = []
    for name, marks in student_data.items():
        average = sum(marks) / len(marks)
        status = "Pass" if average >= 40 else "Fail"
        report.append({
            "name": name,
            "average": average,
            "status": status
        })
    return report

student_data = {
    "Alice": [85, 92, 78],
    "Bob": [35, 45, 50],
    "Charlie": [90, 88, 92],
    "Diana": [40, 42, 38]
}

report = student_report(student_data)
print(report)
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 5.py"
[{'name': 'Alice', 'average': 85.0, 'status': 'Pass'}, {'name': 'Bob', 'average': 43.333333333333336, 'status': 'Pass'}, {'name': 'Charlie',
 'average': 90.0, 'status': 'Pass'}, {'name': 'Diana', 'average': 40.0, 'status': 'Pass'}]
PS C:\Btech\3 - 2\Ai AC>
```

Task 6:

Taxi Fare Calculation-Create Python code that defines a class named `TaxiRide` with attributes: `ride_id`, `driver_name`, `distance_km`, and `waiting_time_min`. Implement a method `display_details()` to print ride details, and a method `calculate_fare()` where:

- ₹15 per km for the first 10 km
- ₹12 per km for the next 20 km
- ₹10 per km above 30 km
- Waiting charge: ₹2 per minute

Create a ride object, display details, and print the total fare.

Copilot Suggested Code :

```python
# Taxi Fare Calculation

class TaxiRide:
    def __init__(self, ride_id, driver_name, distance_km, waiting_time_minutes):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.waiting_time_minutes = waiting_time_minutes

    def display_details(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Waiting Time (minutes): {self.waiting_time_minutes}")

    def calculate_fare(self):
        fare = 0
        if self.distance_km <= 10:
            fare += self.distance_km * 15
        elif self.distance_km <= 30:
            fare += 10 * 15 + (self.distance_km - 10) * 12
        else:
            fare += 10 * 15 + 20 * 12 + (self.distance_km - 30) * 10
        fare += self.waiting_time_minutes * 2

        print(f"Total Fare: ₹{fare}")

ride = TaxiRide("T001", "Ramesh", 35, 15)
ride.display_details()
ride.calculate_fare()

ride2 = TaxiRide("T002", "Suresh", 25, 10)
ride2.display_details()
ride2.calculate_fare()
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 6.py"
Ride ID: T001
Driver Name: Ramesh
Distance (km): 35
Waiting Time (minutes): 15
Total Fare: ₹470
Ride ID: T002
Driver Name: Suresh
Distance (km): 25
Waiting Time (minutes): 10
Total Fare: ₹350
PS C:\Btech\3 - 2\Ai AC>
```

Task 7:

Statistics Subject Performance - Create a Python function \`statistics_subject(scores_list)\` that accepts a list of 60 student scores and computes key performance statistics. The function should return the following:

- Highest score in the class
- Lowest score in the class
- Class average score
- Number of students passed (score ≥ 40)
- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic.

Copilot Suggested Code :

```python
# Statistics Subject Performance

def statistics_subject(scores_list):
    if not scores_list:
        return None
    highest = max(scores_list)
    lowest = min(scores_list)
    average = sum(scores_list) / len(scores_list)
    passed = sum(1 for score in scores_list if score >= 40)
    failed = sum(1 for score in scores_list if score < 40)
    return {
        "highest": highest,
        "lowest": lowest,
        "average": average,
        "passed": passed,
        "failed": failed
    }

print(statistics_subject([85, 92, 78, 96, 88, 73, 91, 84, 89, 76]))
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 7.py"
{'highest': 96, 'lowest': 73, 'average': 85.2, 'passed': 10, 'failed': 0}
PS C:\Btech\3 - 2\Ai AC>
```

Task Description #8 (Transparency in Algorithm Optimization) Task:

Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.

Copilot Suggested Code : Naive

Approach :

```python
# Naive Approach
def is_prime_naive(n):
    """Check if a number is prime using the naive approach."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

Optimized Approach :

```python
# Optimized Approach
def is_prime_optimized(n):
    """Check if a number is prime using the optimized approach."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

Output :

```python
# Example Usage
print(is_prime_naive(29))        # True
# Statistics Subject Performance
print(is_prime_optimized(100))    # False
```

Explanation of Optimization:

- The optimized version improves performance by reducing the number of checks needed to determine if a number is prime.
- Instead of checking all numbers up to n-1, it only checks up to the square root of n.
- Additionally, it skips even numbers and multiples of 3 after handling them separately, which significantly reduces the number of iterations for larger numbers.

Task Description #9 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.

Expected Output:

- Well-commented recursive code.
- Clear explanation of how recursion works.
- Verification that explanation matches actual execution.

Copilot Suggested Code :

```python
# Recursive function to calculate Fibonacci numbers with explanation

def fibonacci(n):
    """
    Calculate the nth Fibonacci number using recursion.

    Parameters:
    n (int): The position in the Fibonacci sequence to calculate.

    Returns:
    int: The nth Fibonacci number.
    """
    # Base case: the first Fibonacci number is 0
    if n == 0:
        return 0
    # Base case: the second Fibonacci number is 1
    elif n == 1:
        return 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage
print(fibonacci(10))  # Output: 55
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 9.py"
55
PS C:\Btech\3 - 2\Ai AC>
```

Explanation of Recursion in the Fibonacci Function

Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem. In the case of the Fibonacci function:

1. Base Cases: The function has two base cases:
   - When n is 0, it returns 0.
   - When n is 1, it returns 1.

   These base cases are essential because they provide stopping points for the recursion, preventing infinite loops.
2. Recursive Calls: For any n greater than 1, the function calls itself twice:
   - It calls fibonacci(n - 1) to get the (n-1)th Fibonacci number.
   - It calls fibonacci(n - 2) to get the (n-2)th Fibonacci number. The results of these two calls are then summed to produce the nth Fibonacci number.
3. Execution Flow: When fibonacci(10) is called, the function will keep breaking down the problem into smaller Fibonacci calculations until it reaches the base cases. The results are then combined as the recursive calls return back up the call stack.

This explanation matches the actual execution of the code, as each call to fibonacci(n) results in further calls until the base cases are reached, at which point the results are combined to yield the final output.

Task Description #10 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

- Code with meaningful exception handling.
- Clear comments explaining each error scenario.
- Validation that explanations align with runtime behavior

Copilot Suggested Code :

```python
# “Generate code with proper error handling and clear explanations for each exceptio
# Python program to read a file and process data with error handling

def read_and_process_file(file_path):
    """
    Reads a file and processes its data.

    Parameters:
    file_path (str): The path to the file to be read.

    Returns:
    list: A list of processed data lines.
    """
    processed_data = []
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            for line in file:
                # Process each line (for example, stripping whitespace)
                processed_line = line.strip()
                processed_data.append(processed_line)
    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file at '{file_path}' was not found.")
    except PermissionError:
        # Handle the case where there are permission issues
        print(f"Error: Permission denied when trying to read '{file_path}'.")
    except Exception as e:
        # Handle any other exceptions that may occur
        print(f"An unexpected error occurred: {e}")
    else:
        # If no exceptions were raised, return the processed data
        return processed_data
# Example usage
file_path = 'C:\\Btech\\3 - 2\\Ai AC\\Assignment - 5.1 & 6.1\\data.txt'
data = read_and_process_file(file_path)
if data is not None:
    print("Processed Data:")
    for line in data:
        print(line)
```

Output :

```
PS C:\Btech\3 - 2\Ai AC> & C:\Python314\python.exe "c:/Btech/3 - 2/Ai AC/Assignment - 5.1 & 6.1/Task - 10.py"
Processed Data:
Hello I'm a CS student studying in SR University.
PS C:\Btech\3 - 2\Ai AC>
```

Explanation of Error Handling in the Code

1. FileNotFoundError: This exception is raised when the specified file does not exist at the given path. The code catches this exception and prints a user-friendly message indicating that the file was not found.
2. PermissionError: This exception occurs when the program does not have the necessary permissions to read the file. The code handles this by informing the user about the permission issue.
3. General Exception Handling: The generic Exception catch-all is used to handle any other unforeseen errors that may arise during file operations. This ensures that the program does not crash unexpectedly and provides feedback about the error.
4. Else Clause: The else clause is executed only if no exceptions were raised in the try block. This is where the processed data is returned, ensuring that the function only returns valid data when the file operations are successful.

This explanation aligns with the runtime behavior of the code, as each specific exception is caught and handled appropriately, providing clarity on what went wrong during file processing.