

Ai Assisted Coding

Assignment – 5.1 & 6

Name : G.Samshray

Batch : 23

Ht.no : 2303A51670

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.

```

❶ Employee.py :- ...
❷ class Employee:
❸     def __init__(self, empid, empname, designation, basic_salary, exp):
❹         self.empid = empid
❺         self.empname = empname
❻         self.designation = designation
❼         self.basic_salary = basic_salary
❼         self.exp = exp
❼     def display_details(self):
⺠         print("Employee ID: (self.empid)")
⺠         print("Employee Name: (self.empname)")
⺠         print("Designation: (self.designation)")
⺠         print("Basic Salary: (self.basic_salary)")
⺠         print("Experience: (self.exp) years")
⺠     def calculate_allowance(self):
⺠         if self.exp < 5:
⺠             allowance = 0.05 * self.basic_salary
⺠         elif 5 <= self.exp < 10:
⺠             allowance = 0.10 * self.basic_salary
⺠         else:
⺠             allowance = 0.20 * self.basic_salary
⺠         print(f"Allowance: {allowance}")
⺠         print(f"Total Salary: {self.basic_salary + allowance}")
⺠ # Example usage
⺠ emp = Employee(101, "John Doe", "Software Engineer", 60000, 7)
⺠ emp.display_details()
⺠ allowance = emp.calculate_allowance()

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL DOCS
```

PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/Employee.py

```

Employee ID: 101
Employee Name: John Doe
Designation: Software Engineer
Basic Salary: 60000
Experience: 7 years
Allowance: 6000.0
Total Salary: 66000.0
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

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.

```
ElectricityBill > ElectricityBill > calculate_bill
1  class ElectricityBill:
2      def __init__(self, customer_id, customer_name, units_consumed):
3          self.customer_id = customer_id
4          self.customer_name = customer_name
5          self.units_consumed = units_consumed
6
7      def display_details(self):
8          print("Customer ID: " + str(self.customer_id))
9          print("Customer Name: " + self.customer_name)
10         print("Units Consumed: " + str(self.units_consumed))
11
12     def calculate_bill(self):
13         if self.units_consumed <= 100:
14             bill_amount = self.units_consumed * 5
15         elif self.units_consumed <= 300:
16             bill_amount = self.units_consumed * 7
17         else:
18             bill_amount = self.units_consumed * 10
19
20         print("Bill Amount: " + str(bill_amount))
21         print("Total Amount to be Paid: " + str(bill_amount))
22
23     # Example usage
24     bill = ElectricityBill(201, "Alice Smith", 250)
25     bill.display_details()
26     bill_amount = bill.calculate_bill()
27
28
PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/Electricity.py
Customer ID: 201
Customer Name: Alice Smith
Units Consumed: 250
Bill Amount: 1750
Total Amount to be Paid: 1750
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

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.

```

Product_Discount_Calculator.py

1 class Product:
2     def __init__(self, product_id, product_name, price, category):
3         self.product_id = product_id
4         self.product_name = product_name
5         self.price = price
6         self.category = category
7
8     def display_details(self):
9         print("Product ID: " + str(self.product_id))
10        print("Product Name: " + self.product_name)
11        print("Price: " + str(self.price))
12        print("Category: " + self.category)
13
14    def calculate_discount(self):
15        if self.category.lower() == "electronics":
16            discount = 0.10 * self.price
17        elif self.category.lower() == "clothing":
18            discount = 0.15 * self.price
19        else:
20            discount = 0.05 * self.price
21
22        print("Discount: " + str(discount))
23        print("Price after Discount: " + str(self.price - discount))
24        print("Total Price to be Paid: " + str(self.price - discount))
25
26 # Example usage
27 product = Product(381, "Smartphone", 50000, "Electronics")
28 product.display_details()
29 product.calculate_discount()

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL FONTS

PS C:\Users\Game\OneDrive\Desktop\python> & "C:\Program Files\Python311\python.exe" c:/Users/Game/OneDrive/Desktop/python/Product_Discount_Calc.py
Product ID: 381
Product Name: Smartphone
Price: 50000
Category: Electronics
Discount: 5000.0
Price after Discount: 45000.0
Total Price to be Paid: 45000.0
PS C:\Users\Game\OneDrive\Desktop\python>

```

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 $\leq 5 \rightarrow$ ₹5 per day
- 6 to 10 days late \rightarrow ₹7 per day
- More than 10 days late \rightarrow ₹10 per day

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

```

● book_late_fee.py » ...
1 class LibraryBook:
2     def __init__(self, BookID, Title, Author, Borrower, Days_late):
3         self.BookID = BookID
4         self.Title = title
5         self.Author = Author
6         self.Borrower = Borrower
7         self.Days_late = Days_late
8     def display_details(self):
9         print("Book ID: " + str(self.BookID))
10        print("Title: " + self.Title)
11        print("Author: " + self.Author)
12        print("Borrower: " + self.Borrower)
13        print("Days Late: " + str(self.Days_late))
14    def calculate_late_fee(self):
15        if self.Days_late <= 5:
16            late_fee = self.Days_late * 5
17        elif 6 <= self.Days_late <= 10:
18            late_fee = self.Days_late * 7
19        else:
20            late_fee = self.Days_late * 10
21        print("Late Fee: " + str(late_fee))
22        print("Total Amount to be Paid: " + str(late_fee))
23    # Example usage
24    book = LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald", "Emily Clark", 8)
25    book.display_details()
26    book.calculate_late_fee()

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/book_late_fee.py
Book ID: 401
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Emily Clark
Days Late: 8
Late Fee: 56
Total Amount to be Paid: 56
PS C:\Users\Ganne\OneDrive\Desktop\python>

```

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.

```

* student_performance_report.py
1  # write a program that defines a function student_report(student_data) that accepts a dictionary containing student names and their marks. This function should
2  # - calculate avg score of each student
3  # - determine pass/fail status (pass if avg >= 40)
4  # - return a summary report as a list of dictionaries
5
6  def student_report(student_data):
7      report = []
8      for student, marks in student_data.items():
9          avg_score = sum(marks) / len(marks)
10         status = "Pass" if avg_score >= 40 else "Fail"
11         report.append({
12             "Student": student,
13             "Average Score": avg_score,
14             "Status": status
15         })
16
17     return report
18
19 # Example usage
20 students = {
21     "Alice": [45, 78, 89],
22     "Bob": [35, 40, 39],
23     "Charlie": [50, 60, 70]
24 }
25 report = student_report(students)
26 for student_summary in report:
27     print(student_summary)

```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORT

```

PS C:\Users\Game\OneDrive\Desktop\python> & "C:\Program Files\Python310\python.exe" C:/Users/Game/OneDrive/Desktop/python/student_performance_report.py
[{"Student": "Alice", "Average Score": 70.33333333333333, "Status": "Pass"}, {"Student": "Bob", "Average Score": 35.0, "Status": "Fail"}, {"Student": "Charlie", "Average Score": 56.0, "Status": "Pass"}]
PS C:\Users\Game\OneDrive\Desktop\python>

```

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.

```

● taxi_fare_calculation.py ...
1  class TaxiRide:
2      def __init__(self, ride_id, driver_name, distance_km, waiting_time_min):
3          self.ride_id = ride_id
4          self.driver_name = driver_name
5          self.distance_km = distance_km
6          self.waiting_time_min = waiting_time_min
7      def display_details(self):
8          print("Ride ID: " + self.ride_id)
9          print("Driver Name: " + self.driver_name)
10         print("Distance (km): " + str(self.distance_km))
11         print("Waiting Time (min): " + str(self.waiting_time_min))
12     def calculate_fare(self):
13         if self.distance_km <= 10:
14             fare = self.distance_km * 15
15         elif 11 <= self.distance_km <= 20:
16             fare = self.distance_km * 12
17         else:
18             fare = self.distance_km * 10
19             waiting_charge = self.waiting_time_min * 2
20             total_fare = fare + waiting_charge
21             print("Fare: " + str(fare))
22             print("Waiting Charge: " + str(waiting_charge))
23             print("Total Fare to be Paid: " + str(total_fare))
24     # Example usage
25     ride = TaxiRide(501, "Michael Scott", 18, 15)
26     ride.display_details()
27     ride.calculate_fare()

PROGRAMS DEBUG CONSOLE OUTPUT TERMINAL FORTS
● PS C:\Users\Ganesh\OneDrive\Desktop\python> & "C:\Program Files\Python312\python.exe" c:/Users/Ganesh/OneDrive/Desktop/python/taxi_fare_calculation.py
Ride ID: 501
Driver Name: Michael Scott
Distance (km): 18
Waiting Time (min): 15
Fare: 216
Waiting Charge: 30
Total Fare to be Paid: 246
● PS C:\Users\Ganesh\OneDrive\Desktop\python>

```

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.

```

# statistics_sub.py3...
# ... a generate a code that defines a function named statistics_subject(scores_list) that accepts a list of 40 student scores and computes key performance statistics. The function should
# return :
#   # Highest score
#   # Lowest score
#   # Average score
#   # Number of students passed(score>=40)
#   # Number of students failed(score<40)
# def statistics_subject(scores_list):
#     highest_score = max(scores_list)
#     lowest_score = min(scores_list)
#     average_score = sum(scores_list) / len(scores_list)
#     passed_count = sum(1 for score in scores_list if score >= 40)
#     failed_count = sum(1 for score in scores_list if score < 40)
#
#     return {
#         "Highest Score": highest_score,
#         "Lowest Score": lowest_score,
#         "Average Score": average_score,
#         "Number of Students Passed": passed_count,
#         "Number of Students Failed": failed_count
#     }
# if __name__ == "__main__":
#     scores = [35, 78, 34, 96, 67, 45, 23, 89, 100, 38, 76, 84, 91, 43, 52, 25, 60, 77, 81, 44, 29, 58, 66, 88, 40, 35, 27, 35, 41, 56, 73, 88, 96, 32, 22, 31, 49, 64, 70, 81, 37, 14, 26, 33, 42, 57, 63, 75, 60, 83, 21, 19, 39, 38, 43, 52, 45, 74, 29, 37, 34, 13, 23, 30]
#     report = statistics_subject(scores)
#     for key, value in report.items():
#         print(f'{key}: {value}')
# 
```

Output from terminal:

```

PS C:\Users\Game\OneDrive\Desktop\python3 & "C:\Program Files\Python311\python.exe" c:/Users/Game/OneDrive/Desktop/python/statistics_sub.py
Highest Score: 100
Lowest Score: 21
Average Score: 56.96175
Number of Students Passed: 43
Number of Students Failed: 26
PS C:\Users\Game\OneDrive\Desktop\python3

```

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.
- Validation that explanations align with runtime behavior.

```

◆ prime.py>-
1  # Generate Python code for two prime-checking methods and explain
2  #- how the optimized version improves performance."
3  # - transparent explanation of time complexity for both methods.
4  # - comparison highlighting efficiency improvements.
5
6  def is_prime_basic(n):
7      """Check if a number is prime using basic method."""
8      if n <= 1:
9          return False
10     for i in range(2, n):
11         if n % i == 0:
12             return False
13     return True
14
15  def is_prime_optimized(n):
16      """Check if a number is prime using optimized method."""
17      if n <= 1:
18          return False
19      if n <= 3:
20          return True
21      if n % 2 == 0 or n % 3 == 0:
22          return False
23      i = 5
24      while i * i <= n:
25          if n % i == 0 or n % (i + 2) == 0:
26              return False
27          i += 6
28      return True

```

```

◆ prime.py>-
28  # Explanation of time complexity:
29  # 1. Basic Method:
30  #   - Time Complexity: O(n)
31  #   - Explanation: In the basic method, we check all numbers from 2 to n
32  #   to see if they divide n evenly. In the worst case, we perform n-2
33  #   checks, leading to linear time complexity.
34  # 2. Optimized Method:
35  #   - Time Complexity: O( $\sqrt{n}$ )
36  #   - Explanation: The optimized method reduces the number of checks
37  #   significantly by eliminating even numbers and checking only up to
38  #   the square root of n. This is because if n is divisible by any number
39  #   greater than its square root, the corresponding factor must be less
40  #   than the square root. Thus, we only need to check up to  $\sqrt{n}$ ,
41  #   leading to a much faster algorithm for large n.
42  # Comparison:
43  # The optimized method is significantly more efficient than the basic
44  # method, especially for large numbers. While the basic method's linear
45  # time complexity can become impractical for large n, the optimized
46  # method's square root time complexity allows it to handle much larger
47  # numbers efficiently. This makes the optimized method preferable for
48  # prime-checking in real-world applications.
49  # Example usage:
50  number = 29
51  print("Is {} prime (basic)? {}".format(number, is_prime_basic(number)))
52  print("Is {} prime (optimized)? {}".format(number, is_prime_optimized(number)))

```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL FORKS

```

PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/users/ganne/onedrive/desktop/python/prime.py
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/users/ganne/onedrive/Desktop/python/prime.py
Is 29 prime (basic)? True
Is 29 prime (optimized)? True
PS C:\Users\Ganne\OneDrive\Desktop\python>

```

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.

```
❶ fib.py > fibonacci
1  # write a code to generate a recursive function to calculate fibonacci numbers.
2  # - add clear comments explaining recursion.
3  # - also explain base cases and recursive calls.
4  # - verification that explanation matches actual execution.
5  def fibonacci(n):
6      """
7          Calculate the nth Fibonacci number using recursion.
8
9          The Fibonacci sequence is defined as:
10         F(0) = 0 (base case)
11         F(1) = 1 (base case)
12         F(n) = F(n-1) + F(n-2) for n > 1 (recursive case)
13
14        Parameters:
15        n (int): The position in the Fibonacci sequence to calculate.
16
17        Returns:
18        int: The nth Fibonacci number.
19        """
20
21        # Base cases
22        if n == 0:
23            return 0
24        elif n == 1:
25            return 1
26        else:
27            # Recursive case: sum of the two preceding numbers
28            return fibonacci(n - 1) + fibonacci(n - 2)
29
30    # Example usage and verification
31    n = 6
32    print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
```

```
◆ fib.py > ⚡fibonacci
26     # Example usage and verification
27     n = 6
28     print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
29
30     # Explanation:
31     # When we call fibonacci(6), the function checks if n is 0 or 1, since it's neither, it proceeds to the recursive case:
32     # fibonacci(6) = fibonacci(5) + fibonacci(4)
33     # This pattern continues, breaking down each call until it reaches the base cases:
34     # fibonacci(1) = 1 and fibonacci(0) = 0.
35     # The results are then combined back up the call stack to produce the final result.
36     # The execution for fibonacci(6) would look like this:
37     # fibonacci(6)
38     # = fibonacci(5) + fibonacci(4)
39     # = (fibonacci(4) + fibonacci(3)) + (fibonacci(3) + fibonacci(2))
40     # = ((fibonacci(3) + fibonacci(2)) + (fibonacci(2) + fibonacci(1))) + ((fibonacci(2) + fibonacci(1)) + (fibonacci(1) + fibonacci(0)))
41     # = ...
42     # ... and so on, until all calls reach the base cases.
43     # The final result is 8, which is the 6th Fibonacci number.
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

PS C:\Users\Game\OneDrive\Desktop\python> & "C:\Program Files\Python312\python.exe" c:/Users/Game/OneDrive/Desktop/python/Fib.py
The 6th Fibonacci number is: 8
PS C:\Users\Game\OneDrive\Desktop\python>

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.

```
● error_handling.py > process_file
  1  # Generate a Python program that reads and processes data from a file using try-except blocks. Include meaningful exception handling and clear
  2  # comments explaining each possible error scenario.
  3
  4  def process_file(filename):
  5      """
  6          Reads a file and processes its data.
  7          Processing: counts number of lines and words.
  8      """
  9
 10     try:
 11         # tries to open the file in read mode
 12         with open(filename, "r") as file:
 13             content = file.read()
 14             # Data processing
 15             lines = content.splitlines()
 16             words = content.split()
 17
 18             print("File processed successfully.")
 19             print("Number of lines:", len(lines))
 20             print("Number of words:", len(words))
 21             print(lines)
 22             print(words)
 23
 24     except FileNotFoundError:
 25         # Raised when the file does not exist at the given path
 26         print("Error: The file was not found. Please check the file name or path.")
 27
 28
 29 ● error_handling.py > process_file
 30  def process_file(filename):
 31      """
 32          # Raised when the file does not exist at the given path
 33          print("Error: The file was not found. Please check the file name or path.")
 34
 35      except PermissionError:
 36          # Raised when the program does not have permission to read the file
 37          print("Error: Permission denied. You do not have access to this file.")
 38
 39      except Exception as e:
 40          # Handles any other unexpected runtime errors
 41          print("Unexpected error occurred:", e)
 42
 43      else:
 44          # Executes only if no exception occurs
 45          print("File reading and processing completed without errors.")
 46
 47      finally:
 48          # Always executes, useful for cleanup actions
 49          print("Program execution finished.")
 50
 51 # Example usage
 52 process_file("sample.txt")
 53
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL FOLDERS

```
● PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:\Program Files\Python312\python.exe" c:/Users/Ganne/OneDrive/Desktop/python/error_handling.py
File processed successfully.
Number of lines: 3
Number of words: 9
['AI Assisted Coding', 'SR University', 'Roll no : 2383A51678']
['AI', 'Assisted', 'Coding', 'SR', 'University', 'Roll', 'no', ':', '2383A51678']
File reading and processing completed without errors.
Program execution finished.
● PS C:\Users\Ganne\OneDrive\Desktop\python>
```