

Lab Assignment 5.1 and 6

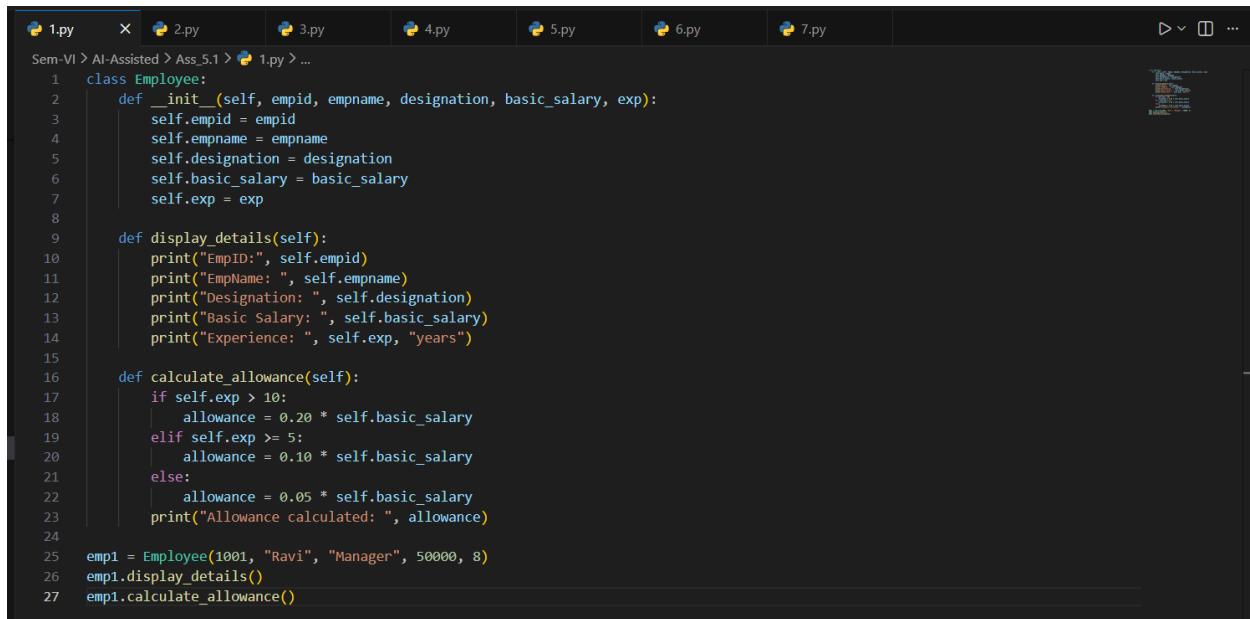
Task 1:

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.



The screenshot shows a code editor window with multiple tabs at the top labeled 1.py, 2.py, 3.py, 4.py, 5.py, 6.py, and 7.py. The active tab is 1.py. The code in 1.py is as follows:

```
1 class Employee:
2     def __init__(self, empid, empname, designation, basic_salary, exp):
3         self.empid = empid
4         self.empname = empname
5         self.designation = designation
6         self.basic_salary = basic_salary
7         self.exp = exp
8
9     def display_details(self):
10        print("EmpID:", self.empid)
11        print("EmpName: ", self.empname)
12        print("Designation: ", self.designation)
13        print("Basic Salary: ", self.basic_salary)
14        print("Experience: ", self.exp, "years")
15
16    def calculate_allowance(self):
17        if self.exp > 10:
18            allowance = 0.20 * self.basic_salary
19        elif self.exp >= 5:
20            allowance = 0.10 * self.basic_salary
21        else:
22            allowance = 0.05 * self.basic_salary
23        print("Allowance calculated: ", allowance)
24
25 emp1 = Employee(1001, "Ravi", "Manager", 50000, 8)
26 emp1.display_details()
27 emp1.calculate_allowance()
```

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.

```

1.py  x  2.py  x  3.py  x  4.py  5.py  6.py  7.py  ... Sem-VI > AI-Assisted > Ass_5.1 > 2.py > ...
1  class ElectricBill:
2      def __init__(self, customer_id, name, units_consumed):
3          self.customer_id = customer_id
4          self.name = name
5          self.units_consumed = units_consumed
6
7      def display_details(self):
8          print("Customer ID:", self.customer_id)
9          print("Name:", self.name)
10         print("Units Consumed:", self.units_consumed)
11
12     def calculate_bill(self):
13         if self.units_consumed <= 100:
14             bill = self.units_consumed * 5
15         elif self.units_consumed <= 300 and self.units_consumed > 100:
16             bill = self.units_consumed * 7
17         else:
18             bill = self.units_consumed * 10
19         print("Total Electric Bill for", self.name, "is:", bill)
20
21 customer1 = ElectricBill(2001, "Anita", 250)
22 customer1.display_details()
23 customer1.calculate_bill()

```

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.

```

1.py  x  2.py  x  3.py  x  4.py  5.py  6.py  7.py  9.py  1  ... Sem-VI > AI-Assisted > Ass_5.1 > 3.py > Product > __init__
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:", self.product_id)
10         print("Product Name:", self.product_name)
11         print("Price:", 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         elif self.category.lower() == "groceries":
20             discount = 0.05 * self.price
21         print("Discount for", self.product_name, "is:", discount)
22
23 product1 = Product(3001, "Smartphone", 20000, "Electronics")
24 product1.display_details()
25 product1.calculate_discount()

```

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.

```
1.py 2.py 3.py 4.py 5.py 6.py 7.py 9.py 1 ... Sem-VI > AI-Assisted > Ass_5.1 > 4.py ...  
1 class LibraryBook:  
2     def __init__(self, book_id, author, borrower, days_late):  
3         self.book_id = book_id  
4         self.author = author  
5         self.borrower = borrower  
6         self.days_late = days_late  
7  
8     def display_details(self):  
9         print("Book ID:", self.book_id)  
10        print("Author:", self.author)  
11        print("Borrower:", self.borrower)  
12        print("Days Late:", self.days_late)  
13  
14    def calculate_late_fee(self):  
15        if self.days_late <= 5:  
16            late_fee = self.days_late * 5  
17        elif self.days_late <= 10:  
18            late_fee = self.days_late * 7  
19        elif self.days_late > 10:  
20            late_fee = self.days_late * 10  
21        print("Late fee for", self.borrower, "is:", late_fee)  
22  
23 book1 = LibraryBook(4001, "J.K. Rowling", "Sam", 12)  
24 book1.display_details()  
25 book1.calculate_late_fee()
```

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.

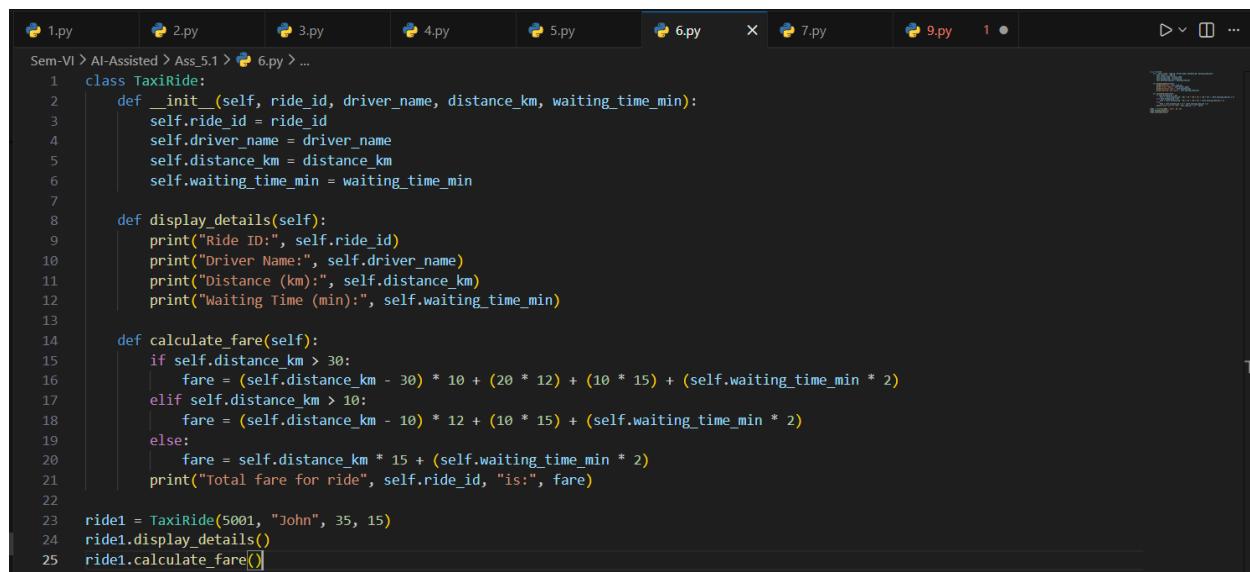
```
1.py 2.py 3.py 4.py 5.py 6.py 7.py 9.py 1 ... Sem-VI > AI-Assisted > Ass_5.1 > 5.py > student_report  
1 def student_report(student_data):  
2     for student in student_data:  
3         avg_marks = sum(student_data[student]["Marks"]) / len(student_data[student]["Marks"])  
4         student_data[student][“avg_marks”] = avg_marks  
5     return student_data  
6  
7 student_data = {"Student1": {"Student_Name": "Vishwa", "Marks": [10, 30, 30, 40]},  
8                 "Student2": {"Student_Name": "Anandu", "Marks": [40, 40, 30, 40]}}  
9  
10 res = student_report(student_data)  
11  
12 for student in student_data:  
13     print(student_data[student])
```

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.



```
1.py 2.py 3.py 4.py 5.py 6.py 7.py 9.py ...
```

```
Sem-VI > AI-Assisted > Ass_5.1 > 6.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
8      def display_details(self):
9          print("Ride ID:", self.ride_id)
10         print("Driver Name:", self.driver_name)
11         print("Distance (km):", self.distance_km)
12         print("Waiting Time (min):", self.waiting_time_min)
13
14     def calculate_fare(self):
15         if self.distance_km > 30:
16             fare = (self.distance_km - 30) * 10 + (20 * 12) + (10 * 15) + (self.waiting_time_min * 2)
17         elif self.distance_km > 10:
18             fare = (self.distance_km - 10) * 12 + (10 * 15) + (self.waiting_time_min * 2)
19         else:
20             fare = self.distance_km * 15 + (self.waiting_time_min * 2)
21         print("Total fare for ride", self.ride_id, "is:", fare)
22
23 ride1 = TaxiRide(5001, "John", 35, 15)
24 ride1.display_details()
25 ride1.calculate_fare()
```

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

```
1.py 2.py 3.py 4.py 5.py 6.py 7.py 8.py 9.py ...  
Sem-VI > AI-Assisted > Ass_5.1 > 7.py > ...  
1 def statistics_subject(scores_list):  
2     highest_score = max(scores_list)  
3     lowest_score = min(scores_list)  
4     class_average = sum(scores_list) / len(scores_list)  
5     passed_students = 0  
6     failed_students = 0  
7     for score in scores_list:  
8         if score >= 40:  
9             passed_students += 1  
10        else:  
11            failed_students += 1  
12  
13    return highest_score, lowest_score, class_average, passed_students, failed_students  
14  
15  
16    from random import randint  
17    scores_list = [randint(0, 100) for _ in range(60)]  
18  
19    highest_score, lowest_score, class_average, passed_students, failed_students = statistics_subject(scores_list)  
20  
21    print("Highest Score:", highest_score)  
22    print("Lowest Score:", lowest_score)  
23    print("Class Average:", class_average)  
24    print("Number of Passed Students:", passed_students)  
25    print("Number of Failed Students:", failed_students)  
26
```

Task 8: (Transparency in Algorithm Optimization)

Description:

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.

```

1.py | 2.py | 3.py | 4.py | 5.py | 6.py | 7.py | 8.py | 9.py | sample.py | notes.txt | D v I ...
Sem-VI > AI-Assisted > Ass_5.1 > 8.py > ...
1
2   Generate two Python code one is basic brute force whereas the other is optimized approach for two prime-checking methods
3   and explain how the optimized version improves performance.
4   ***
5
6   def brute_force_prime(n):
7       # Handle edge cases:
8       # numbers less than or equal to 1 are NOT prime
9       if n <= 1:
10           return False
11
12       # Try dividing n by every number from 2 to n-1
13       for i in range(2, n):
14           # If i is divisible by any number other than 1 and itself
15           if n % i == 0:
16               return False # Not a prime number
17
18       # If no divisors were found
19       return True
20
21   def optimized_prime(n):
22       # Handle edge cases first
23       if n <= 1:
24           return False
25
26       # 2 is the only even prime number
27       if n == 2:
28           return True
29
30       # Eliminates all other even numbers early
31       # This avoids unnecessary loop iterations
32       if n % 2 == 0:
33           return False
34
35       # Only check up to sqrt(n) instead of n-1
36       # Because if n = a * b, at least one of them must be ≤ sqrt(n)
37       limit = int(n ** 0.5) + 1
38
39       # Skip even numbers by incrementing by 2
40       # This halves the number of iterations
41       for i in range(3, limit, 2):
42           if n % i == 0:
43               return False # Composite number found early
44
45       # If no divisors found up to sqrt(n)
46       return True
47
48   # Example usage:
49   number = 29
50   print("Brute-force check: Is {} prime? {}".format(number, brute_force_prime(number)))
51   print("Optimized check: Is {} prime? {}".format(number, optimized_prime(number)))

```

Task 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.

```

1.py | 2.py | 3.py | 4.py | 5.py | 6.py | 7.py | 8.py | 9.py | ... | D v I ...
Sem-VI > AI-Assisted > Ass_5.1 > 9.py > ...
1   """Generate python code that develops fibonacci sequence using recursion Provide well commented and
2   docstring explanation for each step"""
3
4   def fibonacci_recursive(n):
5       """
6           Generate the nth Fibonacci number using recursion.
7
8           The Fibonacci sequence is defined as:
9           F(0) = 0
10          F(1) = 1
11          F(n) = F(n-1) + F(n-2) for n > 1
12
13          Parameters:
14          n (int): The position in the Fibonacci sequence to compute.
15
16          Returns:
17          int: The nth Fibonacci number.
18      """
19
20      # Base case: if n is 0, return 0
21      if n == 0:
22          return 0
23      # Base case: if n is 1, return 1
24      elif n == 1:
25          return 1
26      else:
27          # Recursive case: return the sum of the two preceding numbers
28          return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
29
30  # Example usage:
31  position = 10
32  print("The {}th Fibonacci number is: {}".format(position, fibonacci_recursive(position)))

```

Task 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.

```
1 1py > 2.py > 3.py > 4.py > 5.py > 6.py > 7.py > 8.py > 9.py > 10.py > 10_v2.py > notes.txt
Sem-VI > AI-Assisted > Ass_5.1 > 10_v2.py - ...
1 """
2 Write a python program that reads a text file and processes the data
3 Generate code with proper error handling and clear explanations for each exception
4 """
5
6 try:
7     # Attempt to open the file in read mode
8     with open('data.txt', 'r') as file:
9         try:
10             # Read the contents of the file
11             data = file.read()
12         |
13             # Check if file is empty
14             if not data:
15                 print("The file is empty.")
16             else:
17                 # Process the data (for example, count the number of lines)
18                 lines = data.splitlines()
19                 print(f"The file contains {len(lines)} lines.")
20
21         except MemoryError:
22             # Handle case when file is too large
23             print("Error: File is too large to process.")
24
25         except UnicodeDecodeError:
26             # Handle encoding issues
27             print("Error: Unable to read file due to encoding issues.")
28
29         except Exception as e:
30             # Handle any exceptions that occur during data processing
31             print(f"An error occurred while processing the data: ({e})")
32
33 except FileNotFoundError:
34     # Handle the case where the file does not exist
35     print("Error: The file 'data.txt' was not found.")
36
37 except PermissionError:
38     # Handle permission issues
39     print("Error: You don't have permission to read 'data.txt'.")
40
41 except IsADirectoryError:
42     # Handle case where 'data.txt' is actually a directory
43     print("Error: 'data.txt' is a directory, not a file.")
44
45 except IOError:
46     # Handle other I/O errors
47     print("Error: An I/O error occurred while trying to read the file.")
48
49 except Exception as e:
50     # Handle any other unexpected exceptions
51     print(f"An unexpected error occurred: ({e})")
```