

LAB ASSIGNMENT 6.4

NAME: M.Geethika

ID.NO:2303A52276

SUBJECT: AI ASST CODING

Task 1: Student Performance Evaluation System

The screenshot shows a Google Colab notebook titled "Untitled38.ipynb". The code defines a class "Student" with attributes name, roll_number, and marks. It includes a method "display_details" to print student information and a method "check_performance" to compare marks with class average. A sample execution shows creating a student object and calling these methods.

```
# Student Performance Evaluation System

class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # T0001 Copilot - write a method to display student details
    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)

    # T0002 Copilot - write a method to check if marks are above class average
    def check_performance(self, class_average):
        if self.marks > class_average:
            return "Performance: Above Class Average"
        else:
            return "Performance: Below Class Average"

# ---- Sample Execution ----
student1 = Student("Pranitha", 101, 85)
class_avg = 75

student1.display_details()
print(student1.check_performance(class_avg))

*** Name: Pranitha
Roll Number: 101
Marks: 85
Performance: Above Class Average
```

Code Explanation:

- A Student class is created with attributes name, roll_number, and marks.
- display_details() prints student information.
- check_performance() compares student marks with class average using if-else.
- If marks are greater than average, it returns **Above Class Average**, else **Below Class Average**.

PROMPT:

Create a method to display student details.

Create another method to check if student marks are above class average using if-else and return a message.

Task 2: Data Processing in a Monitoring System

The screenshot shows a Google Colab interface with a code cell containing Python code. The code defines a function `process_task` that iterates over a list of sensor readings. It checks if each reading is even or odd and prints the square of even numbers. The output shows the execution results for each reading from 10 to -2.

```
# List of sensor readings (integers)
sensor_readings = [10, 13, 4, 7, 22, 9, 0, -2]

for reading in sensor_readings:
    # Copilot: Identify even numbers
    # Copilot: Calculate their square
    # Copilot: Print the result in a readable format
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Reading {reading} is even -> square = {square}")
    else:
        # odd readings are ignored for further processing
        print(f"Reading {reading} is odd -> skipped")
```

Output:

```
... Reading 10 is even -> square = 100
Reading 13 is odd -> skipped
Reading 4 is even -> square = 16
Reading 7 is odd -> skipped
Reading 22 is even -> square = 484
Reading 9 is odd -> skipped
Reading 0 is even -> square = 0
Reading -2 is even -> square = 4
```

Code Explanation :

- A list of sensor readings is iterated using a for loop.
- The modulus operator % checks for even numbers.
- Squares of even numbers are calculated using **.
- Output is printed in a readable format.

PROMPT:

Iterate through the list.

Check if the number is even.

Calculate the square of even numbers.

Print the result in a readable format.

Task 3: Banking Transaction Simulation

The screenshot shows a Google Colab interface with a notebook titled "Untitled38.ipynb". The code cell contains the following Python code:

```
##TASK
# Banking Transaction Simulation

class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # TODO: Copilot - write a method to deposit money into the account
    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. Current Balance: {self.balance}")

    # TODO: Copilot - write a method to withdraw money
    # prevent withdrawal if balance is insufficient
    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn {amount}. Current Balance: {self.balance}")
        else:
            print("Insufficient balance! Withdrawal not allowed.")

# ---- Sample Execution ----
account = BankAccount("Pranitha", 5000)

account.deposit(2000)
account.withdraw(3000)
account.withdraw(6000)
```

The output of the code execution is shown below the code cell:

```
*** Deposited 2000. Current Balance: 7000
Withdrawn 3000. Current Balance: 4000
Insufficient balance! Withdrawal not allowed.
```

The Colab interface includes a search bar, a help section, and a battery status indicator at the bottom right.

Code Explanation:

- BankAccount class stores account holder name and balance.
- deposit() adds money to balance.
- withdraw() uses if-else to check balance before withdrawing.
- Prevents invalid withdrawals and shows clear messages.

PROMPT:

Write methods to deposit money.

Write a withdrawal method.

Prevent withdrawal if balance is insufficient.

Display user-friendly messages.

Task 4: Student Scholarship Eligibility Check

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▶ Run all
Insufficient balance! Withdrawal not allowed.

[1]: 


```

###TASK 4
Initialize the list of students (each is a dict with name and score)
students = [
 {"name": "Pranitha", "score": 82},
 {"name": "Rahul", "score": 68},
 {"name": "Anjali", "score": 98},
 {"name": "Kiran", "score": 74},
 {"name": "Sneha", "score": 88}
]

Copilot: Generate a while loop that iterates through the list
Copilot: Print the names of students who scored more than 75
i = 0
print("Eligible students for merit-based scholarship (score > 75):")
while i < len(students):
 student = students[i]
 # Copilot: Check if student's score is greater than 75:
 if student.get("score", 0) > 75:
 # Copilot: Print the student's name and score in a readable format
 print(f"- {student.get('name')} (score: {student.get('score')})")
 i += 1

-- Eligible students for merit-based scholarship (score > 75):
- Pranitha (score: 82)
- Anjali (score: 98)
- Sneha (score: 88)

```


```

Code Explanation:

- A list of dictionaries stores student data.
- A while loop iterates using an index.
- if condition checks scholarship eligibility.
- Names of eligible students are printed.

PROMPT:

Use a while loop to iterate through students list.

Check score greater than 75.

Print eligible student names.

Task 5: Online Shopping Cart Module

The screenshot shows a Google Colab notebook titled "Untitled38.ipynb". The code defines a `ShoppingCart` class with methods for adding items, removing items, and calculating total bill. A discount is applied if the total exceeds 2000.

```
##TASK 5
# Online Shopping Cart Module

class ShoppingCart:
    def __init__(self):
        # list to store items (name, price, quantity)
        self.items = []

    # TODO: Copilot - write a method to add an item to the cart
    def add_item(self, name, price, quantity):
        self.items.append({
            "name": name,
            "price": price,
            "quantity": quantity
        })
        print(f"Added {name} to cart.")

    # TODO: Copilot - write a method to remove an item from the cart by name
    def remove_item(self, name):
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"Removed {name} from cart.")
                return
        print("Item not found in cart.")

    # TODO: Copilot - calculate total bill using a loop
    # apply discount if total exceeds a certain amount
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]

        if total > 2000:
            discount = total * 0.10
            total -= discount
            print(f"Discount Applied: ${discount}")

        print(f"Total Bill Amount: ${total}")
        return total
```

The screenshot shows the execution of the `ShoppingCart` class. It adds three items to the cart and then removes one, resulting in a total bill of \$1500.

```
if total > 2000:
    discount = total * 0.10
    total -= discount
    print(f"Discount Applied: ${discount}")

print(f"Total Bill Amount: ${total}")
return total

# ---- Sample Execution ----
cart = ShoppingCart()

cart.add_item("Laptop Bag", 1200, 1)
cart.add_item("Mouse", 400, 2)
cart.add_item("Notebook", 100, 3)

cart.remove_item("Mouse")

cart.calculate_total()

*** Added Laptop Bag to cart.
Added Mouse to cart.
Added Notebook to cart.
Removed Mouse from cart.
Total Bill Amount: $1500
1500
```

Code Explanation:

- Shopping Cart class stores items in a list.
- Each item includes name, price, and quantity.
- `add_item()` adds products to the cart.

- `remove_item()` removes products by name.
- `calculate_total()` computes bill using a loop and applies discount using if.

PROMPT:

Create methods to add items.

Remove items from cart.

Calculate total using a loop.

Apply discount if total exceeds limit.