AI ASSISTED CODING

ASSIGNMENT-6.4_1557

NAME:T.TEJASRI

HT NO:2303A51557

BATCH:09

Task 1: Student Performance Evaluation System

# Create method to display student details with attributes name,roll number,marks,pass or fail based on marks

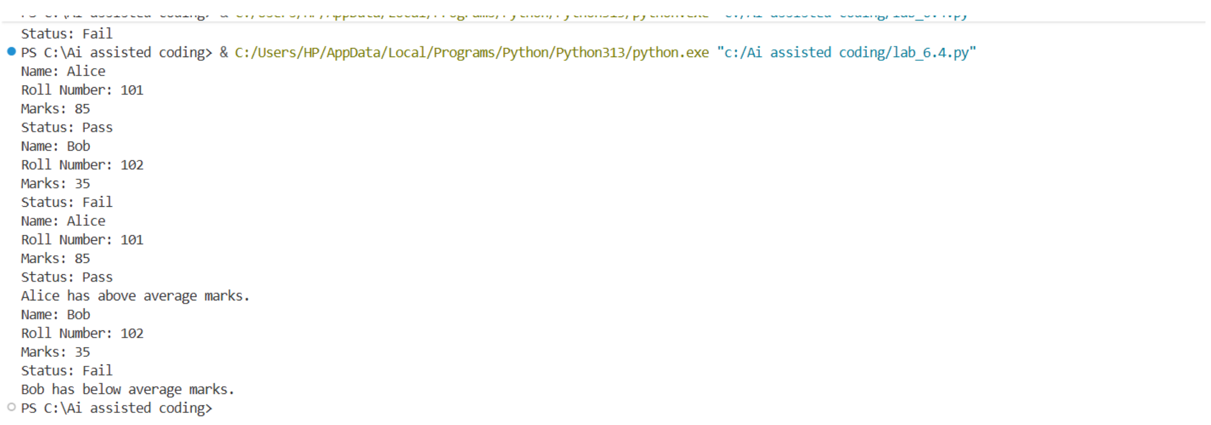
```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        status = "Pass" if self.marks >= 40 else "Fail"
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
        print(f"Status: {status}")
# Example usage
student1 = Student("Alice", 101, 85)
student2 = Student("Bob", 102, 35)
student1.display_details()
student2.display_details()
```

#Create method to compare marks with class average using if-else

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
    def display_details(self):
        status = "Pass" if self.marks >= 40 else "Fail"
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
        print(f"Status: {status}")
    def compare_with_average(self, average_marks):
        if self.marks > average_marks:
            print(f"{self.name} has above average marks.")
        elif self.marks < average_marks:
            print(f"{self.name} has below average marks.")
        else:
            print(f"{self.name} has average marks.")
# Example usage
student1 = Student("Alice", 101, 85)
student2 = Student("Bob", 102, 35)
average_marks = 60
student1.display_details()
student1.compare_with_average(average_marks)
student2.display_details()
student2.compare_with_average(average_marks)
```

OUTPUT:



```
Status: Fail
PS C:\Ai assisted coding> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Ai assisted coding/lab_6.4.py"
Name: Alice
Roll Number: 101
Marks: 85
Status: Pass
Name: Bob
Roll Number: 102
Marks: 35
Status: Fail
Name: Alice
Roll Number: 101
Marks: 85
Status: Pass
Alice has above average marks.
Name: Bob
Roll Number: 102
Marks: 35
Status: Fail
Bob has below average marks.
PS C:\Ai assisted coding>
```

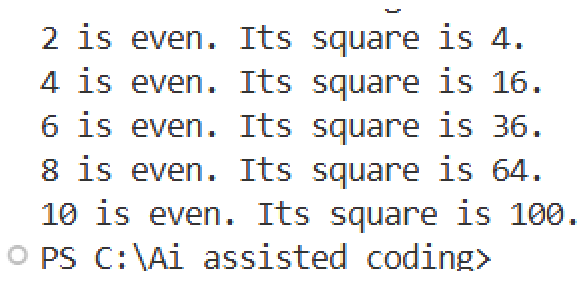
EXPLANATION: This task is about creating a Student class with basic details like name, roll number, and marks. Using Copilot, methods are added to display student details and compare marks with the class average using if-else.

Task 2: Data Processing in a Monitoring System

# Check if number is even using for loop if even print their squares

```
for num in range(1, 11):
    if num % 2 == 0:
        print(f"{num} is even. Its square is {num ** 2}.")
```

OUTPUT:



```
2 is even. Its square is 4.
4 is even. Its square is 16.
6 is even. Its square is 36.
8 is even. Its square is 64.
10 is even. Its square is 100.
PS C:\Ai assisted coding>
```

EXPLANATION:

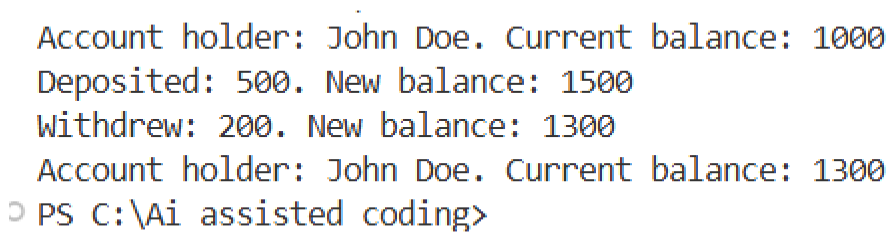
This task focuses on using a for loop to check sensor readings. Copilot helps complete the logic to find even numbers, calculate their squares, and print them using conditions.

Task 3: Banking Transaction Simulation

#create a python class for bankaccount with methods to deposit,withdraw and check balance

```
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited: {amount}. New balance: {self.balance}")
        else:
            print("Deposit amount must be positive.")
    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew: {amount}. New balance: {self.balance}")
        else:
            print("Insufficient funds or invalid withdrawal amount.")
    def check_balance(self):
        print(f"Account holder: {self.account_holder}. Current balance: {self.balance}")
# Example usage
account = BankAccount("John Doe", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(200)
account.check_balance()
```

OUTPUT:



```
Account holder: John Doe. Current balance: 1000
Deposited: 500. New balance: 1500
Withdrew: 200. New balance: 1300
Account holder: John Doe. Current balance: 1300
PS C:\Ai assisted coding>
```

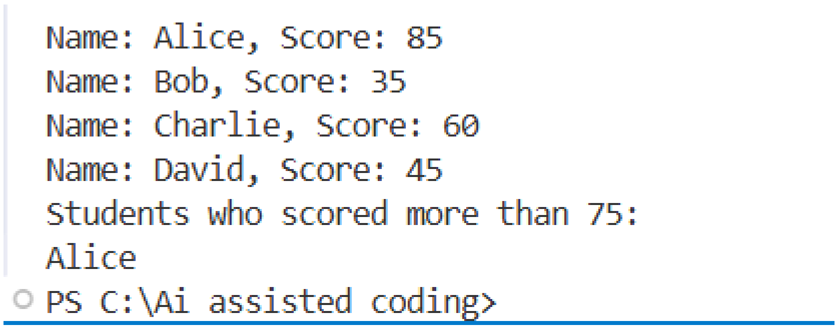
EXPLANATION: In this task, a BankAccount class is created with account holder and balance. Copilot generates methods for deposit and withdrawal, ensuring withdrawals are allowed only when enough balance is available.

Task 4: Student Scholarship Eligibility Check

#Define a list of dictionaries where each dictionary represents a student with name,score using while loop

```
students = [
   {"name": "Alice", "score": 85},
   {"name": "Bob", "score": 35},
   {"name": "Charlie", "score": 60},
   {"name": "David", "score": 45}
]
index = 0
while index < len(students):
   student = students[index]
   print(f"Name: {student['name']}, Score: {student['score']}")
   index += 1
#print names of student who scored more than 75
print("Students who scored more than 75:")
index = 0
while index < len(students):
   student = students[index]
   if student['score'] > 75:
      print(student['name'])
   index += 1
```

OUTPUT:



```
Name: Alice, Score: 85
Name: Bob, Score: 35
Name: Charlie, Score: 60
Name: David, Score: 45
Students who scored more than 75:
Alice
PS C:\Ai assisted coding>
```

EXPLANATION:

This task uses a list of student dictionaries. With Copilot's help, a while loop is created to check scores and print names of students who scored above 75.

Task 5: Online Shopping Cart Module

#writing a Python class named ShoppingCart with:An empty list to store items (each item may include name, price,quantity)

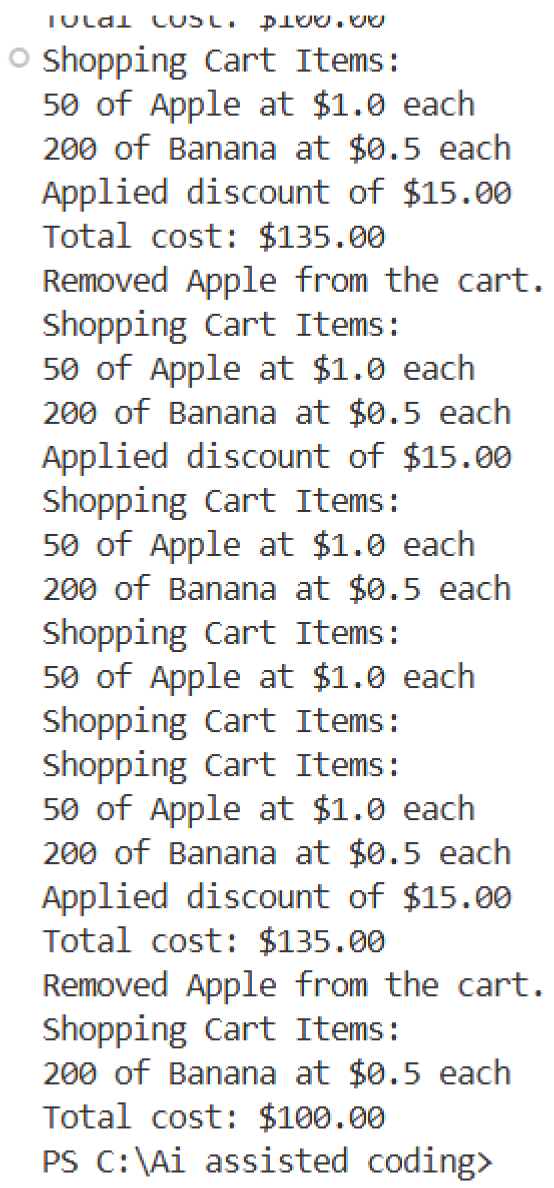
```
class ShoppingCart:
    def __init__(self):
        self.items = []


    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})
        print(f"Added {quantity} of {name} at ${price} each to the cart.")
    def remove_item(self, name):
        self.items = [item for item in self.items if item['name'] != name]
        print(f"Removed {name} from the cart.")
    def view_cart(self):
        print("Shopping Cart Items:")
        for item in self.items:
            print(f"{item['quantity']} of {item['name']} at ${item['price']} each")
    def calculate_total(self):
        total = sum(item['price'] * item['quantity'] for item in self.items)
        print(f"Total cost: ${total}")
        return total
# Example usage
cart = ShoppingCart()
cart.add_item("Apple", 1.0, 5)
cart.add_item("Banana", 0.5, 10)
cart.view_cart()
cart.calculate_total()
cart.remove_item("Apple")
cart.view_cart()
cart.calculate_total()
```

# Methods to add items, remove items, view cart, and calculate total bill using a loop ,apply conditional discounts (e.g., discount if total exceeds acertain amount)

```
class ShoppingCart:
    def __init__(self):
        self.items = []
    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})
        print(f"Added {quantity} of {name} at ${price} each to the cart.")
    def remove_item(self, name):
        self.items = [item for item in self.items if item['name'] != name]
        print(f"Removed {name} from the cart.")
    def view_cart(self):
        print("Shopping Cart Items:")
        for item in self.items:
            print(f"{item['quantity']} of {item['name']} at ${item['price']} each")
    def calculate_total(self):
        total = sum(item['price'] * item['quantity'] for item in self.items)
        if total > 100:
            discount = total * 0.1  # 10% discount
            total -= discount
            print(f"Applied discount of ${discount:.2f}")
        print(f"Total cost: ${total:.2f}")
        return total
# Example usage
cart = ShoppingCart()
cart.add_item("Apple", 1.0, 50)
cart.add_item("Banana", 0.5, 200)
cart.view_cart()
cart.calculate_total()
cart.remove_item("Apple")
cart.view_cart()
cart.calculate_total()
```

OUTPUT:



```
Total cost: $100.00
Shopping Cart Items:
50 of Apple at $1.0 each
200 of Banana at $0.5 each
Applied discount of $15.00
Total cost: $135.00
Removed Apple from the cart.
Shopping Cart Items:
50 of Apple at $1.0 each
200 of Banana at $0.5 each
Applied discount of $15.00
Shopping Cart Items:
50 of Apple at $1.0 each
200 of Banana at $0.5 each
Shopping Cart Items:
50 of Apple at $1.0 each
Shopping Cart Items:
Shopping Cart Items:
50 of Apple at $1.0 each
200 of Banana at $0.5 each
Applied discount of $15.00
Total cost: $135.00
Removed Apple from the cart.
Shopping Cart Items:
200 of Banana at $0.5 each
Total cost: $100.00
PS C:\Ai assisted coding>
```

EXPLANATON: This task involves building a ShoppingCart class to manage items. Copilot helps create methods to add/remove items, calculate the total bill, and apply discounts using loops and conditions.