LAB ASSINGNMENT – WEEK 6.8

NAME : M.Shiva

HALL.NO : 2403A52377

BATCH.NO : AI 14

TOOL USED : GEMINI AND PERPLEXITY AI

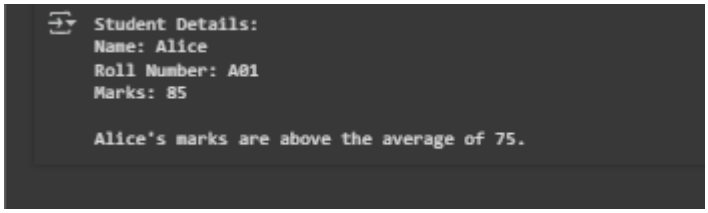
PROMPT 01 :

I need a python code to Start a Python class named Student with attributes name, roll_number, and marks.

Code :

```python
class Student:
    """
    A class to represent a student with name, roll number, and marks.
    """
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    # A method to display the student's details
    def display_details(self):
        """
        Displays the name, roll number, and marks of the student.
        """
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    # A method to check if the student's marks are above an average value
    def is_above_average(self, average_marks):
        """
        Checks if the student's marks are higher than the given average.
        Returns True if marks are above average, False otherwise.
        """
        return self.marks > average_marks

# Example of how to use the class
if __name__ == "__main__":
    # Create a student instance
    student1 = Student("Alice", "A01", 85)

    # Display the student's details
    print("Student Details:")
    student1.display_details()

    # Check if the student's marks are above an average of 75
    class_average = 75
    if student1.is_above_average(class_average):
        print(f"\n{student1.name}'s marks are above the average of {class_average}.")
    else:
        print(f"\n{student1.name}'s marks are not above the average of {class_average}.")
```

OUTPUT :



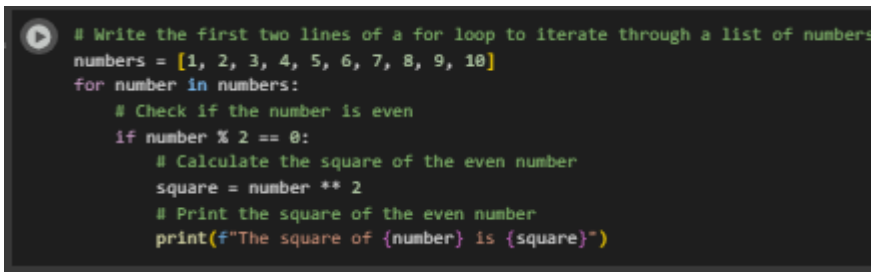
CODE EXPLANATION :

- Class as a Blueprint: The class Student: line defines a blueprint for creating "Student" objects. Each student object will have its own set of data (attributes) and behaviors (methods) based on this blueprint.
- The Constructor (__init__):
    - This special method is called automatically whenever a new Student object is created.
    - Its job is to initialize the object's attributes: name, roll_number, and marks. The self keyword refers to the specific instance of the student being created.
- Methods (Behaviors):
    - display_details(): This method defines a behavior for the Student object. When called, it prints out the student's information in a readable format.
    - is_above_average(average_marks): This method takes an average_marks value as input, compares it to the student's own marks, and returns either True or False.
- Docstrings (Documentation): The text enclosed in triple quotes ("""...""") at the beginning of the class and each method is a docstring. It explains the purpose of the code and is a best practice for making code understandable to others.
- Example Usage Block (if __name__ == "__main__":):
    - This block demonstrates how to use the Student class. It only runs when the script is executed directly (not when imported as a module).
    - It shows how to create a Student object (student1), call its methods, and use the results.

PROMPT 02 :

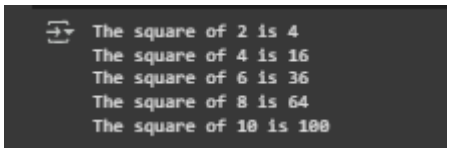
I NEED A PYTHON CODE THAT Write the first two lines of a for loop to iterate through a list of numbers AND how to calculate and print the square of even numbers only

CODE :



```python
# Write the first two lines of a for loop to iterate through a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for number in numbers:
    # Check if the number is even
    if number % 2 == 0:
        # Calculate the square of the even number
        square = number ** 2
        # Print the square of the even number
        print(f"The square of {number} is {square}")
```

OUTPUT :



```
The square of 2 is 4
The square of 4 is 16
The square of 6 is 36
The square of 8 is 64
The square of 10 is 100
```

CODE EXPLANATION :

- Iteration: The for number in numbers: line iterates through each element in the numbers list, assigning the current element to the variable number in each loop.
- Even Number Check: The if number % 2 == 0: line checks if the current number is even by using the modulo operator (%). If the remainder of dividing the number by 2 is 0, the number is even.
- Square Calculation: If the number is even, square = number ** 2 calculates the square of the number by raising it to the power of 2.
- Printing the Result: Finally, print(f"The square of {number} is {square}") prints the calculated square along with the original even number using an f-string for formatted output.

PROMPT 03 :

i need a python code that Create a class called BankAccount with attributes account_holder and balance and complete methods for deposit(), withdraw(), and check for insufficient balance.

CODE :

```python
class BankAccount:
    """
    A class to represent a bank account with account holder and balance.
    """
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """
        Deposits a specified amount into the bank account.
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposit of ${amount} successful. New balance: ${self.balance}")
        else:
            print("Invalid deposit amount. Amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraws a specified amount from the bank account.
        Checks for insufficient balance before withdrawing.
        """
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal of ${amount} successful. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Invalid withdrawal amount. Amount must be positive.")

    def check_balance(self):
        """
        Displays the current balance of the bank account.
        """
        print(f"Current balance for {self.account_holder}: ${self.balance}")

# Example usage:
if __name__ == "__main__":
    # Create a BankAccount instance
    account1 = BankAccount("Alice Smith", 1000)

    # Check initial balance
    account1.check_balance()
```

```
[3]  # Example usage:
     if __name__ == "__main__":
         # Create a BankAccount instance
         account1 = BankAccount("Alice Smith", 1000)

         # Check initial balance
         account1.check_balance()

         # Deposit money
         account1.deposit(500)

         # Withdraw money
         account1.withdraw(200)

         # Try to withdraw more than the balance
         account1.withdraw(2000)

         # Check final balance
         account1.check_balance()
```

OUT PUT :

```
Current balance for Alice Smith: $1000
Deposit of $500 successful. New balance: $1500
Withdrawal of $200 successful. New balance: $1300
Insufficient balance.
Current balance for Alice Smith: $1300
```

CODE EXPLANATION :

- Class Definition: The code defines a class named BankAccount to model a bank account.
- Constructor (__init__): This method is called when you create a new BankAccount object. It initializes the account_holder and sets the initial balance (defaulting to 0 if not provided).
- deposit() Method: This method takes an amount and adds it to the balance. It includes a check to ensure the deposit amount is positive.
- withdraw() Method: This method takes an amount and subtracts it from the balance. It includes a check to ensure the withdrawal amount is positive and another check to prevent withdrawing more than the current balance (insufficient balance).
- check_balance() Method: This method simply prints the current account_holder's name and their balance.
- Example Usage (if __name__ == "__main__":): This block demonstrates how to create a BankAccount object and call its methods to perform operations like checking the balance, depositing, and withdrawing.

PROMPT 04 :

I need a python code that Define a list of student dictionaries with keys name and score and write a while loop to print the names of students who scored more than 75.

CODE :

```
[4] # Define a list of student dictionaries
    students = [
        {"name": "Alice", "score": 88},
        {"name": "Bob", "score": 65},
        {"name": "Charlie", "score": 92},
        {"name": "David", "score": 70},
        {"name": "Eve", "score": 78}
    ]

    # Initialize an index for the while loop
    index = 0

    # Write a while loop to iterate through the list
    print("Students who scored more than 75:")
    while index < len(students):
        # Get the current student dictionary
        student = students[index]

        # Check if the student's score is more than 75
        if student["score"] > 75:
            # Print the name of the student
            print(student["name"])

        # Increment the index
        index += 1
```

OUTPUT :

```
Students who scored more than 75:
Alice
Charlie
Eve
```

CODE EXPPLANATION :

- List of Dictionaries: The code starts by creating a list called students, where each item in the list is a dictionary representing a student with 'name' and 'score' keys.
- Initializing Index: An integer variable index is initialized to 0. This will be used to keep track of the current position in the students list.
- While Loop: The while index < len(students): line starts a while loop that will continue as long as the index is less than the total number of elements in the students list.
- Accessing Student Data: Inside the loop, student = students[index] retrieves the dictionary at the current index from the students list and assigns it to the variable student.
- Conditional Check: The if student["score"] > 75: line checks if the value associated with the 'score' key in the current student dictionary is greater than 75.
- Printing Name: If the score is greater than 75, print(student["name"]) prints the value associated with the 'name' key in the current student dictionary.
- Incrementing Index: index += 1 increases the value of index by 1 after each iteration, moving to the next student in the list.

PROMPT 05 :

I NEED A PYHTON CODE that Begin writing a class ShoppingCart with an empty items list and generate methods to add_item, remove_item, and use a loop to calculate the total bill using conditional discounts.

CODE :

```python
class ShoppingCart:
    """
    A class to represent a shopping cart with items.
    """
    def __init__(self):
        self.items = []

    def add_item(self, item_name, price, quantity=1):
        """
        Adds an item to the shopping cart.
        """
        self.items.append({"name": item_name, "price": price, "quantity": quantity})
        print(f"Added {quantity} x {item_name} to the cart.")

    def remove_item(self, item_name):
        """
        Removes an item from the shopping cart by name.
        Removes the first instance found.
        """
        for item in self.items:
            if item["name"] == item_name:
                self.items.remove(item)
                print(f"Removed {item_name} from the cart.")
                return
        print(f"{item_name} not found in the cart.")

    def calculate_total(self):
        """
        Calculates the total bill with conditional discounts.
        Applies a 10% discount if the total before discount is over $100.
        """
        subtotal = 0
        for item in self.items:
            subtotal += item["price"] * item["quantity"]

        discount = 0
        if subtotal > 100:
            discount = subtotal * 0.10
            print("Applying 10% discount for total over $100.")

        total = subtotal - discount
        print(f"Subtotal: ${subtotal:.2f}")
        print(f"Discount: ${discount:.2f}")
        print(f"Total: ${total:.2f}")
        return total
```

```
            print("Applying 10% discount for total over $100.")

        total = subtotal - discount
        print(f"Subtotal: ${subtotal:.2f}")
        print(f"Discount: ${discount:.2f}")
        print(f"Total: ${total:.2f}")
        return total

    def display_cart(self):
        """
        Displays the current items in the shopping cart.
        """
        if not self.items:
            print("The shopping cart is empty.")
            return

        print("Items in the shopping cart:")
        for item in self.items:
            print(f"- {item['name']} (Price: ${item['price']:.2f}, Quantity: {item['quantity']})")

# Example usage:
if __name__ == "__main__":
    # Create a ShoppingCart instance
    my_cart = ShoppingCart()

    # Add items to the cart
    my_cart.add_item("Laptop", 1200, 1)
    my_cart.add_item("Mouse", 25, 2)
    my_cart.add_item("Keyboard", 75, 1)
    my_cart.add_item("Monitor", 300, 1)


    # Display cart contents
    my_cart.display_cart()

    # Calculate total bill
    my_cart.calculate_total()

    # Remove an item
    my_cart.remove_item("Mouse")

    # Display cart contents after removal
    my_cart.display_cart()

    # Calculate total bill again
    my_cart.calculate_total()
```

OUTPUT :

```
Added 1 x Laptop to the cart.
Added 2 x Mouse to the cart.
Added 1 x Keyboard to the cart.
Added 1 x Monitor to the cart.
Items in the shopping cart:
- Laptop (Price: $1200.00, Quantity: 1)
- Mouse (Price: $25.00, Quantity: 2)
- Keyboard (Price: $75.00, Quantity: 1)
- Monitor (Price: $300.00, Quantity: 1)
Applying 10% discount for total over $100.
Subtotal: $1625.00
Discount: $162.50
Total: $1462.50
Removed Mouse from the cart.
Items in the shopping cart:
- Laptop (Price: $1200.00, Quantity: 1)
- Keyboard (Price: $75.00, Quantity: 1)
- Monitor (Price: $300.00, Quantity: 1)
Applying 10% discount for total over $100.
Subtotal: $1575.00
Discount: $157.50
Total: $1417.50
```

CODE EXPLANANTION :

- Class Definition: The code defines a class named ShoppingCart to represent a shopping cart.
- Constructor (__init__): This method is called when you create a new ShoppingCart object. It initializes an empty list called items to store the items in the cart.
- add_item() Method: This method takes item_name, price, and optional quantity as input and appends a dictionary representing the item to the items list.
- remove_item() Method: This method takes an item_name and removes the first occurrence of that item from the items list if found.
- calculate_total() Method: This method iterates through the items list to calculate the subtotal. It then checks if the subtotal is greater than $100 and applies a 10% discount if it is. Finally, it calculates and prints the total bill.
- display_cart() Method: This method checks if the items list is empty. If not, it iterates through the list and prints the details of each item in the cart.
- Example Usage (if __name__ == "__main__":): This block demonstrates how to create a ShoppingCart object, add items, display the cart, calculate the total, remove an item, and then display the cart and calculate the total again.