Shiva konda

2403A51263

AI ASSISTED CODING – ASSIGNMENT

Lab 5: AI-Based Code Auto-Completion – Classes, Loops, and Conditionals in Python usingGitHub Copilot

Task 1: Auto-Complete a Python Class for Bank Account

Prompt –

Write a class definition comment and construct a class called BankAccount with

account_holder and balance attributes. The class must include methods to deposit, withdraw, and display balance.

Code –

```python
class BankAccount:
    def __init__(self, account_holder, balance=0):
        """
        Initialize a new BankAccount instance.

        Parameters:
        account_holder (str): The name of the account holder.
        balance (float): The initial balance of the account. Defaults to 0.
        """
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        """Deposit the specified amount into the account."""
        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):
        """Withdraw the specified amount from the account if sufficient funds exist."""
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrew ${amount}. New balance: ${self.balance}")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        """Display the current account balance."""
        print(f"Account holder: {self.account_holder}, Balance: ${self.balance}")
```

Code Explanation –

1. Class Definition: The BankAccount class is defined to represent a simple bank account with basic operations.
2. Constructor (__init__): Initializes the account with the holder's name and an optional starting balance (default is 0).
3. Deposit Method: Adds a positive amount to the balance and prints the new balance. If the amount is not positive, it prints an error message.
4. Withdraw Method: Subtracts a positive amount from the balance if there are sufficient funds and prints the new balance. If not enough funds or the amount is not positive, it prints an error message.
5. Display Balance Method: Prints the account holder's name and the current balance.
6. Example Usage: Creates an account for "Alice" with $100, deposits $50, withdraws $30, and displays the final balance.

Output –

```
# Example usage:
account = BankAccount("Alice", 100)
account.deposit(50)          # Deposited $50. New balance: $150
account.withdraw(30)         # Withdrew $30. New balance: $120
account.display_balance()    # Account holder: Alice, Balance:
[18]  ✓ 0.0s

Deposited $50. New balance: $150
Withdrew $30. New balance: $120
Account holder: Alice, Balance: $120
```

Task 2: Auto-Complete a For Loop to Sum Even Numbers in a List

Prompt –

write a comment and the initial line of a loop to iterate over a list. complete the logic to sum all even numbers in the list.

Code –

```
# Iterate over a list and sum all even numbers
numbers = [1, 2, 3, 4, 5, 6]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
print(f"Sum of even numbers: {even_sum}")
[ ]
```

Code Explanation –

1. numbers = [...] → creates a list of numbers from 1 to 10.
2. even_sum = 0 → variable to store the total of even numbers.
3. for num in numbers: → loop goes through each number in the list.
4. if num % 2 == 0: → checks if the number is even (remainder 0 when divided by 2).
5. even_sum += num → adds the even number to the total sum.
6. After the loop finishes, print(...) displays the final sum.

Output –

```
# Iterate over a list and sum all even numbers
numbers = [1, 2, 3, 4, 5, 6]
even_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
print(f"Sum of even numbers: {even_sum}")
```

[7] ✓ 0.0s

```
Sum of even numbers: 12
```

Task 3: Auto-Complete Conditional Logic to Check Age Group

Prompt –

Start a function that takes age as input and returns whether the person is a child, teenager, adult, or senior using if-elif-else.

Code –

```
def age_group(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

# Example usage:
print(age_group(45))  # Output: "Adult"
```

[ ]

Code explanation –

1. def age_group(age): → defines a function that takes age as input.
2. if age < 13: → if the age is less than 13, return “Child”.
3. elif age < 20: → if the age is less than 20 (but ≥ 13), return “Teenager”.
4. elif age < 60: → if the age is less than 60 (but ≥ 20), return “Adult”.
5. else: → if age is 60 or more, return “Senior”.
6. print(age_group(45)) → calls the function with 45, which satisfies the “Adult” condition.

Output –

```python
def age_group(age):
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

# Example usage:
print(age_group(45))  # Output: "Adult"
```

[10] ✓ 0.0s

```
Adult
```

Task 4: Auto-Complete a While Loop to Reverse Digits of a Number

Prompt –

Write a comment and start a while loop to reverse the digits of a number. complete the loop logic.

Code –

```python
# Reverse the digits of a number using a while loop
num = 1234
reversed_num = 0
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
print(reversed_num)  # Output: 4321
```

Code explanation –

1. num = 1234 → the number we want to reverse.
2. reversed_num = 0 → variable to store the reversed number.
3. while num > 0: → loop continues until all digits are processed.
4. digit = num % 10 → extracts the last digit of num.
5. reversed_num = reversed_num * 10 + digit → shifts existing digits left and adds the new digit.
6. num //= 10 → removes the last digit from num (integer division).
7. After loop ends, print(...) shows the reversed number.

Output –

```
# Reverse the digits of a number using a while loop
num = 1234
reversed_num = 0
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num //= 10
print(reversed_num)  # Output:
[11]  ✓ 0.0s
...  4321
```

Task 5: Auto-Complete Class with Inheritance (Employee → Manager)

Prompt –

Begin a class Employee with attributes name and salary. Then, start a derived class Manager that inherits from Employee and adds department. complete the methods and constructor chaining.

Code –

```python
# Employee class with name and salary attributes
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}")

# Manager class inherits from Employee and adds department
class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}")

# Example usage:
mgr = Manager("John", 50000, "IT")
mgr.display()  # Output: Name: John, Salary: 50000, Dept: IT
```

Code explanation –

1. class Employee: → base class with attributes name and salary.
2. __init__ in Employee → constructor initializes name and salary.
3. display() in Employee → prints employee details.
4. class Manager(Employee): → derived class that inherits from Employee.
5. __init__ in Manager → uses super().__init__ to call the parent constructor, then adds department.
6. display() in Manager → overrides parent's display method to also show department.
7. manager = Manager("John", 50000, "IT") → creates a Manager object.

Output –

```python
        super().__init__(name, salary)
        self.department = department

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}

# Example usage:
mgr = Manager("John", 50000, "IT")
mgr.display()  # Output: Name: John, Salary:
```

[12] ✓ 0.0s

```
Name: John, Salary: 50000, Dept: IT
```