Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

```python
class BankAccount:
    def __init__(self, account_holder, balance=0.0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposit of ${amount:.2f} successful.")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal of ${amount:.2f} successful.")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"Account balance for {self.account_holder}: ${self.balance:.2f}")
```

```python
# Sample usage
account1 = BankAccount("Alice Smith", 1000)
account1.display_balance()
account1.deposit(500)
account1.display_balance()
account1.withdraw(200)
account1.display_balance()
account1.withdraw(2000) # Attempt to withdraw more than balance
account1.display_balance()
account1.deposit(-100) # Attempt to deposit negative amount
```

```
Account balance for Alice Smith: $1000.00
Deposit of $500.00 successful.
Account balance for Alice Smith: $1500.00
Withdrawal of $200.00 successful.
Account balance for Alice Smith: $1300.00
Insufficient funds.
Account balance for Alice Smith: $1300.00
Deposit amount must be positive.
```

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

TASK-1 The first code cell defines a Python class called BankAccount. This class is a blueprint for creating bank account objects. It has:

An _init_ method (the constructor) that initializes a new bank account with an account_holder name and an optional starting balance (defaulting to 0.0).

A deposit method that allows you to add funds to the account, ensuring the deposit amount is positive.

A withdraw method that allows you to remove funds from the account, checking for sufficient funds and a positive withdrawal amount.

A display_balance method that prints the current account balance.

The second code cell demonstrates how to use the BankAccount class. It:

Creates an instance of the BankAccount class named account1 for "Alice Smith" with an initial balance of $1000. * Calls the display_balance() method to show the initial balance. * Calls the deposit() method to add $$1000. * Calls the display_balance() method to show the initial balance. * Calls the deposit() method to add $500.

Calls display_balance() again to show the updated balance.

Calls the withdraw() method to remove $200. * Calls display_balance() to show the balance after withdrawal. * Attempts to withdraw $$200. * Calls display_balance() to show the balance after withdrawal. * Attempts to withdraw $2000, which demonstrates the insufficient funds check.

Attempts to deposit a negative amount (-$100), which demonstrates the positive deposit amount check.

The output you provided shows the results of these actions, including the successful transactions and the messages for insufficient funds an d invalid deposit amount.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

```python
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
total_even = 0
for number in numbers:
  # Use GitHub Copilot to complete the logic to sum all even numbers in the list.
  if number % 2 == 0:
    total_even += number

print(f"The sum of even numbers in the list is: {total_even}")
```

```
The sum of even numbers in the list is: 30
```

TASK-2

EXPLANATION Certainly! The code cell you're asking about iterates through a list of numbers and calculates the sum of only the even numbers within that list.

Here's a breakdown:

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line creates a list named numbers containing integers from 1 to 10.

total_even = 0: This line initializes a variable called total_even to 0. This variable will be used to store the sum of the even numbers found in the list.

for number in numbers:: This is a for loop that starts iterating through each element in the numbers list. In each iteration, the current element is assigned to the variable number.

if number % 2 == 0:: Inside the loop, this is an if statement that checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it means the number is even.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

total_even += number: If the if condition is true (the number is even), this line adds the current number to the total_even variable. The += is a shorthand for total_even = total_even + number.

print(f"The sum of even numbers in the list is: {total_even}"): After the loop has finished iterating through all the numbers in the list, this line prints the final value of total_even, which is the sum of all the even numbers found.

In essence, the code checks each number in the list, and if it's divisible by 2 with no remainder, it adds that number to a running total. Finally, it prints the accumulated total of the even numbers.

```
def age_group(age):
    """Categorizes a person's age into child, teenager, adult, or senior."""
    # Use GitHub Copilot to complete the conditionals for age groups.
    if age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

print(f"Output for age_group(45) → \"{age_group(45)}\"")
```

```
Output for age_group(45) → "Adult"
```

TASK-3

EXPLANATION-3 The code in the cell you're asking about defines a Python function called age_group that determines a person's age category based on their age.

Here's a breakdown:

def age_group(age):: This line defines a function named age_group that takes one argument, age.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

"""Categorizes a person's age into child, teenager, adult, or senior.""": This is a docstring, which explains what the function does.

if age < 13:: This is the first condition. If the age is less than 13, the code inside this if block is executed.

return "Child": If the age is less than 13, the function returns the string "Child".

elif age < 20:: This is an "else if" condition. If the first if condition is false (the age is not less than 13), this condition is checked. If the age is less than 20 (meaning it's between 13 and 19, inclusive), the code inside this elif block is executed.

return "Teenager": If the age is between 13 and 19, the function returns the string "Teenager".

elif age < 65:: This is another "else if" condition. If the previous conditions are false, this condition is checked. If the age is less than 65 (meaning it's between 20 and 64, inclusive), the code inside this elif block is executed.

return "Adult": If the age is between 20 and 64, the function returns the string "Adult".

else:: This is the final "else" block. If none of the previous if or elif conditions are true, the code inside this else block is executed. This covers all ages 65 and above.

return "Senior": If the age is 65 or greater, the function returns the string "Senior".

print(f"Output for age_group(45) → \"{age_group(45)}\""): This line calls the age_group function with the value 45 and prints the output in a specific format. Since 45 is less than 65 but not less than 20 or 13, the function returns "Adult", and the output shows "Output for age_group(45) → "Adult"".

In summary, the function uses a series of conditional checks (if, elif, else) to categorize an input age into one of four groups: Child, Teenager, Adult, or Senior.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

```python
num = 1234
reversed_num = 0

# Use GitHub Copilot to complete the while loop logic.
while num > 0:
    digit = num % 10
    reversed_num = reversed_num * 10 + digit
    num = num // 10

print(f"Output: {reversed_num}")
# Output: 4321
```

```
Output: 4321
```

TASK-4

EXPLANATION The code cell you're asking about reverses the digits of an integer using a while loop.

Here's a breakdown of how it works:

num = 1234: This line initializes an integer variable num with the value 1234, which is the number we want to reverse.

reversed_num = 0: This line initializes an integer variable reversed_num to 0. This variable will store the reversed number as it's being built.

while num > 0:: This is a while loop that continues as long as the value of num is greater than 0. Once num becomes 0, it means all digits have been processed, and the loop terminates.

digit = num % 10: Inside the loop, this line extracts the last digit of num. The modulo operator (%) gives the remainder when num is divided by 10. For example, if num is 1234, 1234 % 10 is 4.

reversed_num = reversed_num * 10 + digit: This is the core of the reversal logic.

reversed_num * 10: This shifts the existing digits in reversed_num one place to the left, making space for the new digit.

+ digit: This adds the extracted digit to the rightmost position of reversed_num.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

Let's trace this with num = 1234:

Iteration 1: digit is 4. reversed_num becomes 0 * 10 + 4 = 4. num becomes 123.

Iteration 2: digit is 3. reversed_num becomes 4 * 10 + 3 = 43. num becomes 12.

Iteration 3: digit is 2. reversed_num becomes 43 * 10 + 2 = 432. num becomes 1.

Iteration 4: digit is 1. reversed_num becomes 432 * 10 + 1 = 4321. num becomes 0.

num = num // 10: This line updates num by removing the last digit. The integer division operator (//) divides num by 10 and discards the remainder. This effectively removes the last digit.

print(f"Output: {reversed_num}"): After the while loop finishes (when num is 0), this line prints the final value of reversed_num, which is the original number with its digits reversed.

So, the loop repeatedly takes the last digit of the original number, adds it to the right of the reversed_num, and then removes the last digit from the original number until the original number becomes 0.

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    # Use GitHub Copilot to complete methods for Employee class

class Manager(Employee):
    def __init__(self, name, salary, department):
        super().__init__(name, salary)
        self.department = department

    # Use GitHub Copilot to complete methods and constructor chaining for Manager class

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

# Sample usage
manager1 = Manager("John", 50000, "IT")
manager1.display_info()
```

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

TASK-5 EXPLANATION The code cell you're asking about demonstrates object-oriented programming in Python by defining a base class Employee and a derived class Manager.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

Here's a breakdown:

Employee Class:

class Employee:: This line defines a class named Employee. This is the base class from which other classes can inherit.

def _init(self, name, salary):: This is the constructor (init_) for the Employee class. It's called when you create a new Employee object.

self: Refers to the instance of the class being created.

name: An argument to pass the employee's name.

salary: An argument to pass the employee's salary.

self.name = name: This line assigns the provided name argument to the name attribute of the Employee object.

self.salary = salary: This line assigns the provided salary argument to the salary attribute of the Employee object.

Use GitHub Copilot to complete methods for Employee class: This is a comment indicating where additional methods for the Employee class could be added (though in this specific code, no additional methods are defined for Employee).

Manager Class:

class Manager(Employee):: This line defines a class named Manager that inherits from the Employee class. The (Employee) in parentheses indicates that Manager is a subclass of Employee. This means Manager objects will automatically have all the attributes and methods of the Employee class, plus any new ones defined in Manager.

def _init_(self, name, salary, department):: This is the constructor for the Manager class. It takes name, salary, and an additional department argument.

Assignment-04

S.MOHANTH SIDDARTHA

2403A52047

BATCH:03

super()._init(name, salary): This is called "constructor chaining." super() refers to the parent class (Employee). super().init_(name, salary) calls the constructor of the parent Employee class, passing the name and salary arguments to it. This initializes the name and salary attributes inherited from Employee.

self.department = department: This line assigns the provided department argument to the department attribute, which is specific to the Manager class.

Use GitHub Copilot to complete methods and constructor chaining for Manager class: This is a comment indicating where additional methods for the Manager class could be added.

def display_info(self):: This defines a method named display_info within the Manager class.

print(f"Name: {self.name}, Salary: {self.salary}, Dept: {self.department}"): This line prints the name, salary, and department attributes of the Manager object in a formatted string. It can access self.name and self.salary because they were inherited from the Employee class and initialized by calling the parent constructor.

Sample Usage:

manager1 = Manager("John", 50000, "IT"): This line creates an instance (an object) of the Manager class named manager1. It passes "John" for name, 50000 for salary, and "IT" for department to the Manager constructor.

manager1.display_info(): This line calls the display_info method on the manager1 object, which prints the manager's information in the specified format.

In summary, this code demonstrates how to create a base class and a derived class that inherits from it, how to use constructors to initialize object attributes, and how to call the parent class's constructor using super() (constructor chaining). It also shows how to define and call methods within these classes.