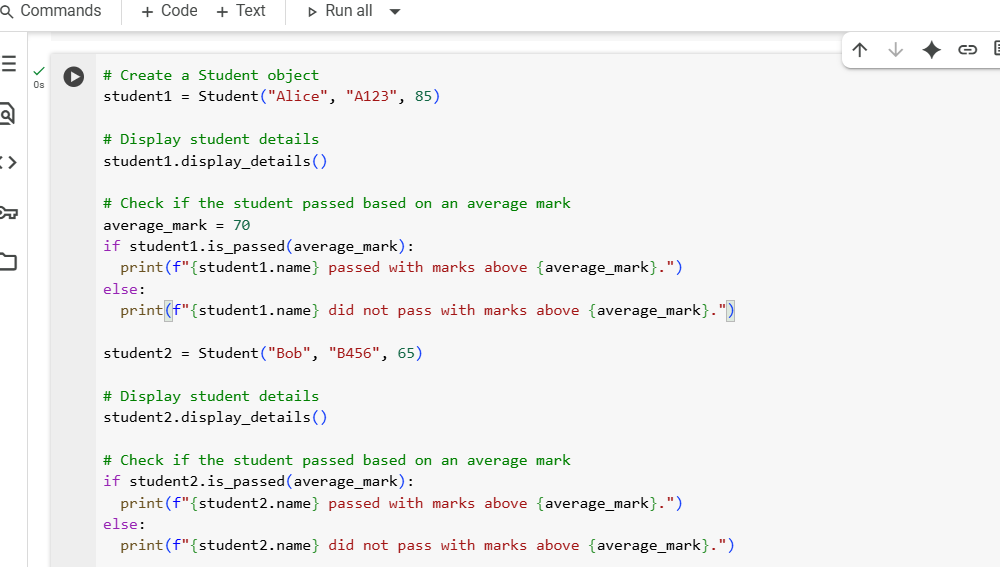
Assignment-6.4

Task-1:

Code:



Output:

A screenshot of a computer

AI-generated content may be incorrect.

EXPLANATION:

**Explanation of this cell (Cell 1):**

This cell defines a Python class named Student. Think of a class as a template for creating objects.

The \_\_init\_\_ method is like a setup function; it runs when you create a new Student. It takes the student's name, roll\_number, and marks and stores them for that specific student object.

The display\_details method simply prints out the stored name, roll number, and marks of a student object.

The is\_passed method takes an average\_marks value. It compares the student's stored marks to the average\_marks. It returns True if the student's marks are higher and False otherwise.

**Explanation of this cell (Cell 2):**

This cell demonstrates how to use the Student class defined in Cell 1.

First, it creates a Student object named student1 with the name "Alice", roll number "A123", and marks 85.

Then, it calls the display\_details() method on student1 to print Alice's information.

It sets an average\_mark variable to 70.

It calls the is\_passed() method on student1 with the average\_mark. An if statement checks the result and prints whether Alice passed or not based on her marks being greater than 70.

It repeats the process for a second student, student2, named "Bob" with marks 65, showing the output for a student who did not pass.

Task-2:

Code:

A screenshot of a computer

AI-generated content may be incorrect.

Output:

A black text on a white background

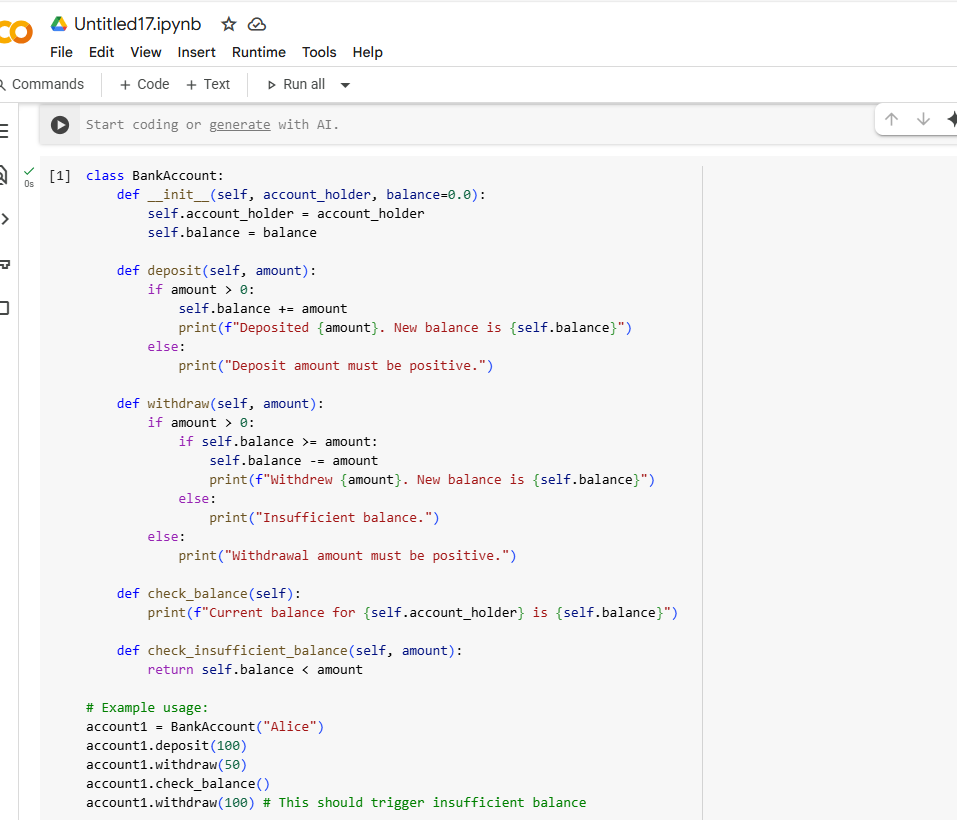
AI-generated content may be incorrect.

Explanation:

1. **Initialize a list:** A list named numbers is created containing integers from 1 to 10.
2. **Start a loop:** The code begins a for loop that will go through each item in the numbers list, assigning the current item to the variable number in each iteration.
3. **Check for even numbers:** Inside the loop, an if statement checks if the current number is even. It does this by using the modulo operator (%) to find the remainder when number is divided by 2. If the remainder is 0, the number is even.
4. **Calculate the square (if even):** If the if condition is true (the number is even), the code calculates the square of the number using the exponentiation operator (\*\* 2) and stores the result in a variable called square.
5. **Print the result (if even):** Still inside the if block, if the number was even, the code prints a formatted string that includes the original even number and its calculated square.
6. **Continue loop:** The loop then moves to the next number in the numbers list and repeats steps 3-5 until all numbers have been processed.

Task-3:

Code:



Output:

A screenshot of a computer

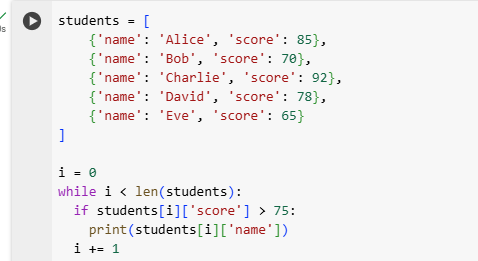
AI-generated content may be incorrect.

Explanation:

1. **Class Definition:**
   * class BankAccount:: This line defines a new class named BankAccount. Classes are blueprints for creating objects (instances).
2. **Constructor (\_\_init\_\_)**:
   * def \_\_init\_\_(self, account\_holder, balance=0.0):: This is the constructor method, which is called when you create a new BankAccount object.
   * self: Refers to the instance of the class being created.
   * account\_holder: This is a parameter that takes the name of the account holder when you create an account.
   * balance=0.0: This is another parameter for the initial balance. It has a default value of 0.0, meaning if you don't provide a balance when creating an account, it will start at zero.
   * self.account\_holder = account\_holder: This line assigns the value passed for account\_holder to an attribute of the object called account\_holder.
   * self.balance = balance: This line assigns the value passed for balance to an attribute of the object called balance.
3. **deposit() Method**:
   * def deposit(self, amount):: This method is used to add funds to the account.
   * self: Refers to the instance of the class.
   * amount: The amount to be deposited.
   * if amount > 0:: Checks if the deposit amount is positive.
   * self.balance += amount: If the amount is positive, it adds the amount to the current balance.
   * print(f"Deposited {amount}. New balance is {self.balance}"): Prints a confirmation message with the new balance.
   * else:: If the amount is not positive.
   * print("Deposit amount must be positive."): Prints an error message.
4. **withdraw() Method**:
   * def withdraw(self, amount):: This method is used to withdraw funds from the account.
   * self: Refers to the instance of the class.
   * amount: The amount to be withdrawn.
   * if amount > 0:: Checks if the withdrawal amount is positive.
   * if self.balance >= amount:: If the amount is positive, it then checks if the current balance is sufficient for the withdrawal.
   * self.balance -= amount: If the balance is sufficient, it subtracts the amount from the balance.
   * print(f"Withdrew {amount}. New balance is {self.balance}"): Prints a confirmation message with the new balance.
   * else:: If the balance is insufficient.
   * print("Insufficient balance."): Prints an error message.
   * else:: If the withdrawal amount is not positive.
   * print("Withdrawal amount must be positive."): Prints an error message.
5. **check\_balance() Method**:
   * def check\_balance(self):: This method is used to display the current balance.
   * self: Refers to the instance of the class.
   * print(f"Current balance for {self.account\_holder} is {self.balance}"): Prints the account holder's name and their current balance.
6. **check\_insufficient\_balance() Method**:
   * def check\_insufficient\_balance(self, amount):: This method checks if a given withdrawal amount would result in an insufficient balance.
   * self: Refers to the instance of the class.
   * amount: The amount to check against the balance.
   * return self.balance < amount: Returns True if the current balance is less than the amount, indicating insufficient funds, and False otherwise.
7. **Example Usage**:
   * account1 = BankAccount("Alice"): Creates a new instance of the BankAccount class named account1 with the account holder "Alice" and the default balance of 0.0.
   * account1.deposit(100): Calls the deposit method on account1 to add 100 to the balance.
   * account1.withdraw(50): Calls the withdraw method on account1 to subtract 50 from the balance.
   * account1.check\_balance(): Calls the check\_balance method to print the current balance.
   * account1.withdraw(100): Calls the withdraw method again, this time attempting to withdraw 100. Since the balance is 50, this will trigger the "Insufficient balance" message.

Task-4:

Code:



Output:

A close-up of a white background

AI-generated content may be incorrect.

Explanation:

1. **Initialize the list of students:**
   * students = [...]: This line creates a list named students. Each item in the list is a dictionary, representing a student with their name and score.
2. **Initialize the loop counter:**
   * i = 0: This line initializes a variable i to 0. This variable will be used as an index to access elements in the students list.
3. **Start the while loop:**
   * while i < len(students):: This line starts a while loop. The loop will continue to execute as long as the value of i is less than the total number of items in the students list (len(students)).
4. **Access student data:**
   * Inside the loop, students[i] accesses the dictionary for the student at the current index i.
   * students[i]['score'] accesses the value associated with the key 'score' for the current student.
5. **Check the student's score:**
   * if students[i]['score'] > 75:: This line checks if the current student's score is greater than 75.
6. **Print the student's name (if score is high enough):**
   * print(students[i]['name']): If the condition in the if statement is true (the score is greater than 75), this line prints the name of the current student. students[i]['name'] accesses the value associated with the key 'name' for the current student.
7. **Increment the counter:**
   * i += 1: This line increments the value of i by 1. This moves the loop to the next student in the list.
8. **Loop continues:**
   * The loop then goes back to the while condition (while i < len(students):) to check if it should continue processing the next student.

This process repeats until i is no longer less than the length of the students list, at which point the loop terminates. The output shows the names of the students whose scores were greater than 75.

Task – 5:

Code:

A screenshot of a computer screen

AI-generated content may be incorrect.

Output:

A black text on a white background

AI-generated content may be incorrect.

Explanation:

1. **Class Definition and Initialization (\_\_init\_\_)**:
   * class ShoppingCart:: Defines a new class named ShoppingCart.
   * def \_\_init\_\_(self):: The constructor method, called when a new ShoppingCart object is created.
   * self.items = []: Initializes an empty list called items within the object. This list will store the items added to the shopping cart. Each item will be represented as a dictionary.
2. **add\_item() Method**:
   * def add\_item(self, item, price, quantity=1):: This method adds an item to the cart. It takes the item name, its price, and an optional quantity (defaulting to 1).
   * The code then iterates through the self.items list (for cart\_item in self.items:).
   * if cart\_item['item'] == item:: It checks if the item being added already exists in the cart.
   * If the item exists, cart\_item['quantity'] += quantity: The quantity of the existing item is increased.
   * If the item does not exist, self.items.append({'item': item, 'price': price, 'quantity': quantity}): A new dictionary representing the item is added to the self.items list.
3. **remove\_item() Method**:
   * def remove\_item(self, item, quantity=1):: This method removes a specified quantity of an item from the cart.
   * It iterates through the self.items list to find the item to remove.
   * if cart\_item['item'] == item:: Checks if the current item in the loop matches the item to be removed.
   * if cart\_item['quantity'] > quantity:: If the quantity in the cart is more than the quantity to remove, it decreases the quantity.
   * elif cart\_item['quantity'] == quantity:: If the quantities match, the entire item dictionary is removed from the self.items list.
   * else:: If the quantity to remove is more than what's in the cart, an error message is printed.
   * If the item is not found after checking all items, a "not found" message is printed.
4. **calculate\_total() Method**:
   * def calculate\_total(self):: This method calculates the total cost of the items in the cart, applying discounts.
   * total\_bill = 0: Initializes a variable to keep track of the total bill.
   * It iterates through each item dictionary in the self.items list.
   * item\_total = item['price'] \* item['quantity']: Calculates the cost of the current item (price times quantity).
   * if item['quantity'] >= 5:: Checks if the quantity of the current item is 5 or more.
   * If the quantity is 5 or more, a 10% discount is calculated (discount = item\_total \* 0.10) and applied to the item\_total. A message about the item discount is printed.
   * total\_bill += item\_total: The calculated (and potentially discounted) item total is added to the total\_bill.
   * if total\_bill > 100:: After calculating the sum of all item totals, it checks if the total\_bill is over 100.
   * If the total\_bill is over 100, a 5% total discount is calculated (total\_discount = total\_bill \* 0.05) and applied to the total\_bill. A message about the total discount is printed.
   * return total\_bill: The final calculated total bill is returned.
5. **Example Usage**:
   * cart = ShoppingCart(): Creates a new instance of the ShoppingCart class.
   * The following lines demonstrate calling the add\_item and remove\_item methods with different items and quantities.
   * total = cart.calculate\_total(): Calls the calculate\_total method on the cart object and stores the returned value in the total variable.
   * print(f"\nTotal bill: ${total:.2f}"): Prints the final calculated total bill, formatted to two decimal places.

\*\*\*\*\*