

Assignment-6.4

TASK-1:

Prompt:

Write a python program to create a class named student with attributes like name, roll_no, marks.

#Write a python program display_details and is_passed methods to the student class and then add code to input details for multiple students.

Code and Output:



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

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.roll_no}")
        print(f"Marks: {self.marks}")

    def is_passed(self):
        return self.marks >= 40 # Assuming 40 is the passing mark

def get_student_details():
    name = input("Enter student name: ")
    roll_no = input("Enter student roll number: ")
    marks = int(input("Enter student marks: "))
    return Student(name, roll_no, marks)

students = []
num_students = int(input("Enter the number of students: "))

for _ in range(num_students):
    student = get_student_details()
    students.append(student)

print("\nStudent Details:")
for student in students:
    student.display_details()
    if student.is_passed():
        print(f"{student.name} has passed.")
    else:
        print(f"{student.name} has failed.")
print("-" * 20)
```

```

Enter the number of students: 3
Enter student name: savika
Enter student roll number: 102
Enter student marks: 99
Enter student name: nandhu
Enter student roll number: 086
Enter student marks: 90
Enter student name: sreeja
Enter student roll number: 103
Enter student marks: 98

Student Details:
Name: savika
Roll No: 102
Marks: 99
savika has passed.
-----
Name: nandhu
Roll No: 086
Marks: 90
nandhu has passed.
-----
Name: sreeja
Roll No: 103
Marks: 98
sreeja has passed.
-----

```

Explanation:

This code defines a `Student` class to represent student information and then collects and displays details for multiple students.

Here's a breakdown:

1. `Student` Class:

- `__init__(self, name, roll_no, marks)`: This is the constructor. It initializes a new `Student` object with the given `name`, `roll_no`, and `marks`.
- `display_details(self)`: This method prints the name, roll number, and marks of the student.
- `is_passed(self)`: This method checks if the student's marks are 40 or above and returns `True` if they are, indicating the student has passed, and `False` otherwise.

2. `get_student_details()` function:

- This function prompts the user to enter the name, roll number, and marks for a single student and returns a `Student` object with these details.

3. Main part of the script:

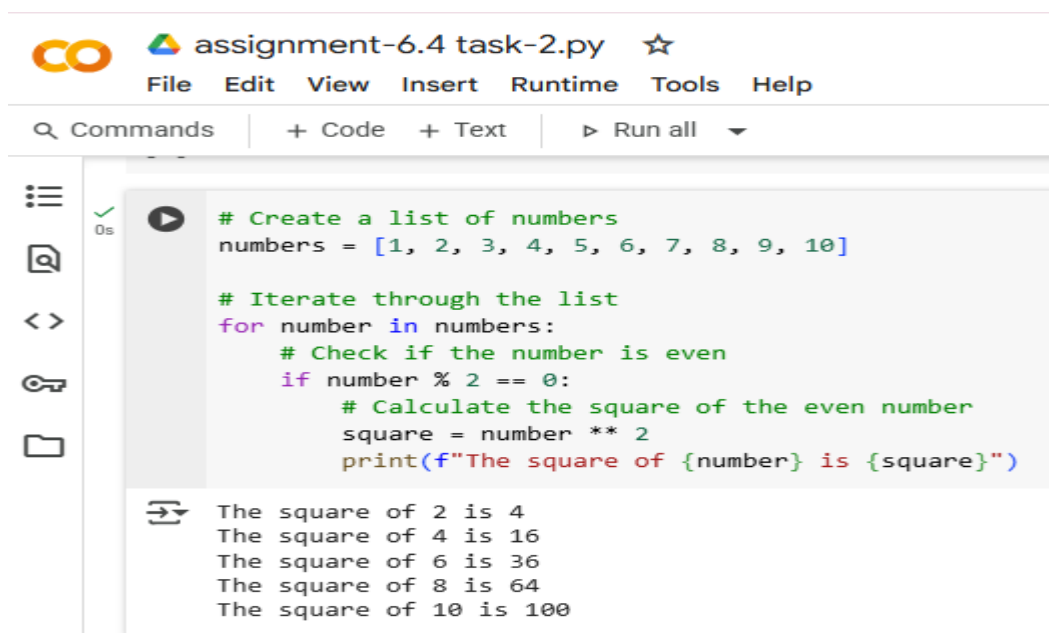
- An empty list called `students` is created to store the `Student` objects.
- The user is asked to enter the number of students they want to process.
- A `for` loop runs for the specified number of students:
 - Inside the loop, `get_student_details()` is called to get the information for one student.
 - The returned `Student` object is added to the `students` list.
- After collecting all student details, the code prints a header "Student Details:".
- Another `for` loop iterates through the `students` list:
 - For each `student` object, `display_details()` is called to show their information.
 - `is_passed()` is called to check if the student passed, and a message is printed accordingly.
 - A line of hyphens is printed to separate the details of each student.

TASK-2:

Prompt:

Write a python program to create a list and write first 2 lines in for loop iterate through the list and calculate the square of even numbers only.

Code with Output:



The screenshot shows a code editor window titled "assignment-6.4 task-2.py". The code defines a list of numbers from 1 to 10, iterates through it, and prints the square of each even number. The output shows the squares of 2, 4, 6, 8, and 10.

```
# Create a list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Iterate through the list
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(f"The square of {number} is {square}")
```

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

Explanation:

- `numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`: This line creates a list named `numbers` containing the integers from 1 to 10.
- `for number in numbers:`: This starts a loop that will go through each item in the `numbers` list, one by one. In each turn of the loop, the current item from the list is temporarily stored in the variable `number`.
- `if number % 2 == 0:`: Inside the loop, this line checks if the current `number` is even. The `%` operator is the modulo operator, which gives you the remainder of a division. If a number divided by 2 has a remainder of 0, it means the number is even.
- `square = number ** 2`: If the `if` condition is true (the number is even), this line calculates the square of the `number` using the `**` operator (exponentiation) and stores the result in a variable named `square`.
- `print(f"The square of {number} is {square}")`: This line prints a message to the console. The `f""` is a formatted string literal, which allows you to easily include the values of variables directly within the string. It will output something like "The square of 2 is 4", "The square of 4 is 16", and so on, for each even number found in the list.

In summary, the code iterates through the list of numbers, identifies the even ones, calculates their squares, and then prints the result for each even number.

TASK-3

Prompt:

Write a python program to create a class named bank account with attributes like account_holder, balance.

#Write a python program to check deposit, withdraw and check insufficient balance.

Code with Output:

```
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"Deposit of ${amount} successful. New balance: ${self.balance}")
        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} successful. New balance: ${self.balance}")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    def check_balance(self):
        print(f"Account holder: {self.account_holder}, Current balance: ${self.balance}")

# Example usage:
account1 = BankAccount("Alice")
account1.deposit(1000)
account1.withdraw(500)
account1.check_balance()
account1.withdraw(600) # Test insufficient balance
```

Deposit of \$1000 successful. New balance: \$1000
Withdrawal of \$500 successful. New balance: \$500
Account holder: Alice, Current balance: \$500
Insufficient balance.

Explanation:

- `class BankAccount:` : This line defines a new class named `BankAccount` . A class is a blueprint for creating objects (in this case, bank accounts).
- `__init__(self, account_holder, balance=0):` : This is the constructor method. It's called when you create a new `BankAccount` object.
 - `self` : Refers to the instance of the class being created.
 - `account_holder` : An argument to store the name of the account holder.
 - `balance=0` : An optional argument to set the initial balance, defaulting to 0.
- `self.account_holder = account_holder` : This line stores the provided `account_holder` name as an attribute of the `BankAccount` object.
- `self.balance = balance` : This line stores the initial `balance` as an attribute of the `BankAccount` object.
- `deposit(self, amount):` : This method handles deposits.
 - It checks if the `amount` is positive.
 - If it is, it adds the `amount` to the `self.balance` .
 - It then prints a success message with the new balance.
 - If the `amount` is not positive, it prints an error message.
- `withdraw(self, amount):` : This method handles withdrawals.
 - It checks if the `amount` is positive.
 - If it is, it checks if the `self.balance` is greater than or equal to the `amount` .
 - If there are sufficient funds, it subtracts the `amount` from the `self.balance` and prints a success message.
 - If there are insufficient funds, it prints an "Insufficient balance" message.
 - If the `amount` is not positive, it prints an error message.

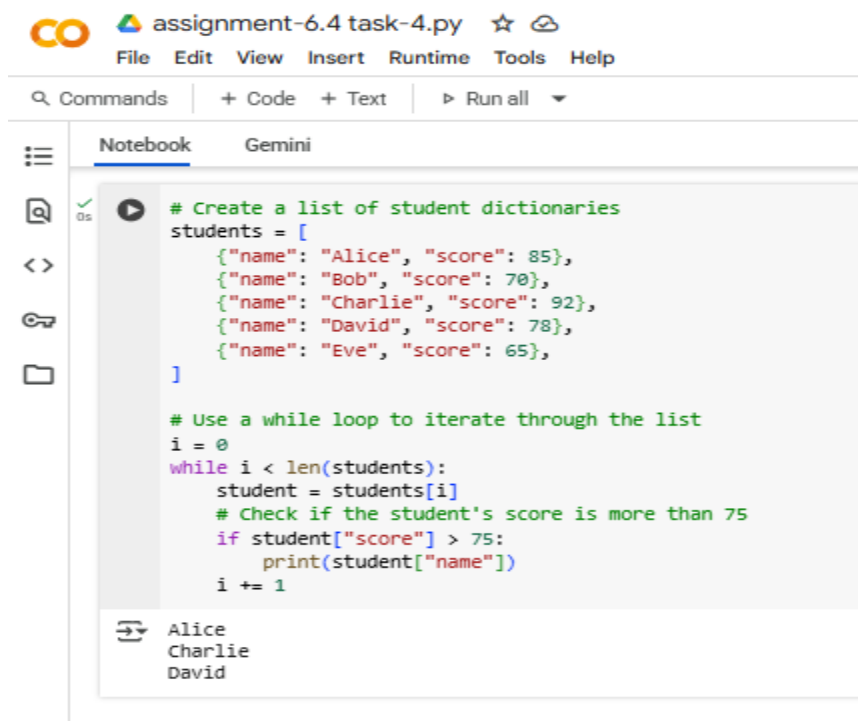
-
- `check_balance(self):` : This method prints the current `account_holder` and `balance` of the `BankAccount` object.
 - `account1 = BankAccount("Alice")` : This line creates a new `BankAccount` object named `account1` with "Alice" as the account holder and a default balance of 0.
 - `account1.deposit(1000)` : This calls the `deposit` method on the `account1` object to deposit 1000.
 - `account1.withdraw(500)` : This calls the `withdraw` method on the `account1` object to withdraw 500.
 - `account1.check_balance()` : This calls the `check_balance` method to display the current balance.
 - `account1.withdraw(600)` : This calls the `withdraw` method again to test the insufficient balance case.

TASK-4:

Prompt:

#write a python program to check a list of student dictionaries with key names and score and write a while loop to print the names of students who scored more than 75.

Code with Output:



The screenshot shows a code editor interface with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar (Commands, + Code, + Text, Run all). The code is written in a notebook format with a left sidebar containing icons for file operations. The code defines a list of student dictionaries and uses a while loop to print the names of students with scores greater than 75. The output shows the names Alice, Charlie, and David.

```
# Create a list of student dictionaries
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 70},
    {"name": "Charlie", "score": 92},
    {"name": "David", "score": 78},
    {"name": "Eve", "score": 65},
]

# Use a while loop to iterate through the list
i = 0
while i < len(students):
    student = students[i]
    # Check if the student's score is more than 75
    if student["score"] > 75:
        print(student["name"])
    i += 1
```

Alice
Charlie
David

Explanation:

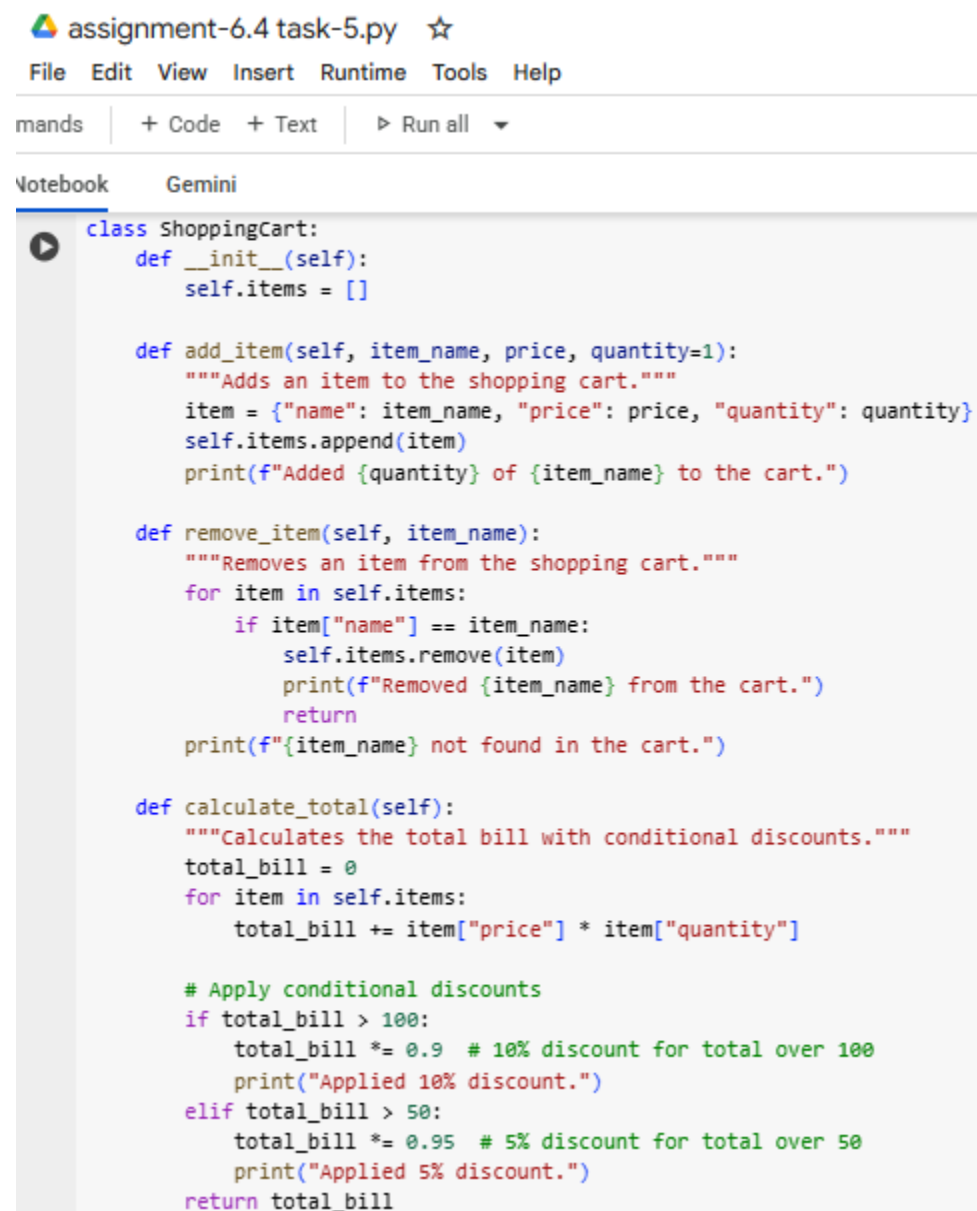
- `students = [...]`: This line creates a list named `students`. Each item in the list is a dictionary, and each dictionary represents a student with their name and score.
- `i = 0`: This line initializes a variable `i` to 0. This variable will be used as an index to keep track of our position in the `students` list.
- `while i < len(students):`: This is a `while` loop. It will continue to execute the code inside it as long as the value of `i` is less than the total number of students in the `students` list.
- `student = students[i]`: Inside the loop, this line gets the dictionary for the student at the current index `i` from the `students` list and assigns it to the variable `student`.
- `if student["score"] > 75:`: This line checks if the value associated with the key `"score"` in the current `student` dictionary is greater than 75.
- `print(student["name"])`: If the condition in the previous line is true (the score is greater than 75), this line prints the value associated with the key `"name"` from the current `student` dictionary.
- `i += 1`: This line increments the value of `i` by 1. This is important to move to the next student in the list in the next iteration of the loop and eventually stop the loop.

TASK-5:

Prompt:

#write a python program to create a class named ShoppingCart with empty list. Check methods like add_item, remove_item using loop to calculate the total bill using conditional discounts.

Code:



```
assignment-6.4 task-5.py ☆
File Edit View Insert Runtime Tools Help

mands | + Code + Text | ▶ Run all ▼

Notebook Gemini

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

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

    def remove_item(self, item_name):
        """Removes an item from the shopping cart."""
        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."""
        total_bill = 0
        for item in self.items:
            total_bill += item["price"] * item["quantity"]

        # Apply conditional discounts
        if total_bill > 100:
            total_bill *= 0.9 # 10% discount for total over 100
            print("Applied 10% discount.")
        elif total_bill > 50:
            total_bill *= 0.95 # 5% discount for total over 50
            print("Applied 5% discount.")
        return total_bill
```

```
# Example usage:
cart = ShoppingCart()
cart.add_item("Laptop", 1000, 1)
cart.add_item("Mouse", 25, 2)
cart.add_item("Keyboard", 75, 1)

print("\nItems in cart:")
for item in cart.items:
    print(f"- {item['name']}: ${item['price']} x {item['quantity']}")

total = cart.calculate_total()
print(f"\nTotal bill: ${total:.2f}")

cart.remove_item("Mouse")

print("\nItems in cart after removal:")
for item in cart.items:
    print(f"- {item['name']}: ${item['price']} x {item['quantity']}")

total = cart.calculate_total()
print(f"\nTotal bill after removal: ${total:.2f}")
```

Output:

```
➦ Added 1 of Laptop to the cart.
Added 2 of Mouse to the cart.
Added 1 of Keyboard to the cart.

Items in cart:
- Laptop: $1000 x 1
- Mouse: $25 x 2
- Keyboard: $75 x 1
Applied 10% discount.

Total bill: $1012.50
Removed Mouse from the cart.

Items in cart after removal:
- Laptop: $1000 x 1
- Keyboard: $75 x 1
Applied 10% discount.

Total bill after removal: $967.50
```

Explanation:

The code defines a Python class called `ShoppingCart` to simulate a simple shopping cart.

- `__init__(self)`: This is the constructor of the class. When you create a `ShoppingCart` object, it initializes an empty list called `self.items`. This list will store the items added to the cart.
- `add_item(self, item_name, price, quantity=1)`: This method adds an item to the `self.items` list. It takes the `item_name`, `price`, and an optional `quantity` (defaulting to 1) as input. It creates a dictionary for the item and appends it to the `self.items` list. It also prints a message confirming the item was added.
- `remove_item(self, item_name)`: This method removes an item from the `self.items` list based on its name. It iterates through the `self.items` list and if it finds a dictionary with a matching `item_name`, it removes that dictionary from the list and prints a confirmation message. If the item is not found, it prints a message indicating that.
- `calculate_total(self)`: This method calculates the total bill for the items in the cart. It iterates through the `self.items` list, multiplies the `price` by the `quantity` for each item, and adds it to `total_bill`. After calculating the initial total, it applies conditional discounts:

- If `total_bill` is greater than 100, a 10% discount is applied (`total_bill *= 0.9`).
- If `total_bill` is greater than 50 (but not greater than 100), a 5% discount is applied (`total_bill *= 0.95`). It also prints a message indicating if a discount was applied. Finally, it returns the calculated `total_bill`.

The example usage at the end of the code demonstrates how to create a `ShoppingCart` object, add items, calculate and print the total, remove an item, and then recalculate and print the total again.