1.Create a class called `User`.
Every time a new user instance is created, store it in a class-level list (so that all users are tracked).
Add a **`@classmethod`** called `get_user_count()` that returns the total number of users created.

```
u1 = User("Alice")

u2 = User("Bob")

print(User.get_user_count())  # Output: 2
```

===

2. BankAccount Class

Create a class called **`BankAccount`** with a **`balance`** property.

- The balance should **never go below zero** (if a withdrawal would make it negative, prevent it or raise an error).
- Implement a **`@classmethod`** named `from_string(cls, data)` that takes a string like `"Ali,1500"` and returns a new `BankAccount` object with the given owner and balance.

===

3.Car Rental System

Design a class called Car with the following attributes and methods:

Attributes:

•car_id (string): representing the unique car ID
•model (string): representing the model of the car
•year (integer): representing the manufacturing year of the car
•is_available (boolean): representing whether the car is available for rental

Methods:

•__init__: Initialize the attributes (default availability should be True)
•rent_car: Mark the car as rented if available
•return_car: Mark the car as available
•display_info: Display the car ID, model, year, and availability status

Design a class called RentalAgency with the following attributes and methods:

Attributes:

•cars(list of Car objects): representing the cars in the rental agency

Methods:

•__init__: Initialize the attributes. If no cars are provided, create an empty list
•add_car: Add a car to the rental agency
•find_car: Find and return a car by car ID
•rent_car: Rent a car from the agency (update availability)
•display_available_cars: Display information of all available cars
==

4. Product Inventory
Design a class called Product with the following attributes and methods:
Attributes:
• name (string): representing the name of the product.
• price (float): representing the price of the product.
• quantity (integer): representing the quantity of the product in stock.

Methods:
• __init__: Initialize the attributes.
• sell: Update the quantity of the product after a sale.
• restock: Update the quantity of the product after restocking.
• display_info: Display the name, price, and quantity of the product.

Design a class called Inventory with the following attributes and methods:
Attributes:
• products (list of Product objects): representing the products in the inventory.
Methods:
• __init__: Initialize the attributes. If no products are provided, create an empty list.
• add_product: Add a product to the inventory.
• update_product: Update the price of a product in the inventory.
• delete_product: Delete a product from the inventory.
• read_inventory: Display the information of all the products in the inventory.

==
5. BankAccount and Bank
Design a class called BankAccount with the following attributes and methods:
Attributes:
• account_number (string): representing the account number.
• balance (float): representing the current balance of the account.
Methods:

• __init__: Initialize the attributes.
• deposit: Add the given amount to the account's balance.
• withdraw: Subtract the given amount from the account's balance.
• display_balance: Display the current balance of the account.

Design a class called Bank with the following attributes and methods:
Attributes:
• name (string): representing the name of the bank.
• accounts (list of BankAccounts): representing the bank accounts.

Methods:
• __init__(self, name): Initialize the attributes.
• create_account: Create a new bank account with the given account number and initial deposit.
• close_account: Close a bank account with the given account number.
• update_balance: Update the balance of a bank account by adding or subtracting the given amount.
• display_accounts: Display the information of all the bank accounts.

==
6. Magic Methods
Design a class called Book that represents a book. Implement the following magic methods to explore the concept of magic methods:
• __init__(self, title, author): Initialize the title and author attributes.
• __str__(self): Return a string representation of the book, displaying the title and author.
• __len__(self): Return the length of the book, representing the number of pages.
• __getitem__(self, index): Retrieve the page content at the given index.
• __setitem__(self, index, content): Set the page content at the given index.
• __delitem__(self, index): Delete the page at the given index.
• __contains__(self, keyword): Check if the book contains a specific keyword.

```
book = Book("The Catcher in the Rye", "J.D. Salinger")
book.pages = ["Page 1 content", "Page 2 content", "Page 3 content"]
print(book)
# Output: "Book: The Catcher in the Rye by J.D. Salinger"

print(len(book))
print(book[2])
# Output: 3
# Output: "Page 3 content"

book[2] = "Updated content for page 3"
print(book[2])
# Output: "Updated content for page 3"

del book[1]
print(len(book))
# Output: 2

print("Catcher" in book) # Output: False
print("content" in book) # Output: True
```

==
7. Magic Methods
Create a ShoppingCart class that represents a shopping cart with a collection of items. Item represents an individual item in the shopping cart with name and price
Magic Methods:
• __init__(self, name, price): Initializes the name and price attributes of the item.

• __str__(self): Returns a string representation of the item, displaying the name and price.
ShoppingCart represents a shopping cart with a collection of items.
Attributes:
items which is list to store the items in the shopping cart.
Magic Methods:
• __init__(self): Initializes the shopping cart with an empty list of items.
• __str__(self): Returns a string representation of the shopping cart, displaying the items in the cart.
• __len__(self): Returns the number of items in the shopping cart.
• __contains__(self, item): Checks if a specific item is present in the shopping cart.
• __add__(self, other_cart): Combines two shopping carts by adding the items from both carts.
• add_item(self, item): Adds an item to the shopping cart.
• remove_item(self, item): Removes a specific item from the shopping cart.
• total_price(self): Calculates and returns the total price of all items in the shopping cart.

==

8. Data Hiding

Design a class called Employee that represents an employee. Implement encapsulation and data hiding to protect the attributes of the employee. The class should include the following methods and attributes:
Attributes (private):
• __name (string): representing the name of the employee.
• __salary (float): representing the salary of the employee.
Methods:
• __init__(self, name, salary): Initialize the name and salary attributes.
• get_name(self): Return the name of the employee.
• get_salary(self): Return the salary of the employee.
• set_name(self, new_name): Update the name of the employee.
• set_salary(self, new_salary): Update the salary of the employee.

```
employee = Employee("John Doe", 5000.0)

print(employee.get_name())
# Output: "John Doe"

print(employee.get_salary())
# Output: 5000.0

employee.set_name("Jane Smith")
employee.set_salary(6000.0)
print(employee.get_name())
# Output: "Jane Smith"

print(employee.get_salary())
# Output: 6000.0
# Accessing the private attributes directly raises an AttributeError

print(employee.__name)
# Output: AttributeError: 'Employee' object has no attribute '__name'

print(employee.__salary)
# Output: AttributeError: 'Employee' object has no attribute '__salary'
```