Magic Methods & Operator Overloading

1. **Exercise**: Create a class `Matrix` that supports addition, subtraction, and multiplication with other matrices.
    - Implement methods `__add__`, `__sub__`, and `__mul__`.
    - Ensure that the operations only work with matrices of the same dimensions.
    - Include a method to display the matrix in a readable format.
2. **Exercise**: Create a class `Money` that supports addition, subtraction, and comparison with other Money objects.
    - Implement methods `__add__`, `__sub__`, `__eq__`, `__lt__`, and `__gt__`.
    - Include a method to display the money amount in a readable format.
    - Ensure that operations handle different currencies appropriately.
3. **Exercise**: Create a class `RationalNumber` that supports addition, subtraction, multiplication, and division with other rational numbers.
    - Implement methods `__add__`, `__sub__`, `__mul__`, and `__truediv__`.
    - Include a method to display the rational number in a readable format.
    - Ensure that operations handle division by zero appropriately.

Object Lifecycle

4. **Exercise**: Create a class `TemporaryFile` that creates a file during initialization and deletes it when the object is deleted.
    - Implement the `__init__` method to create a file and print a message.
    - Implement the `__del__` method to delete the file and print a message.
    - Demonstrate the lifecycle of the `TemporaryFile` object in a test case.
5. **Exercise**: Create a class `Cache` that initializes a cache during object creation and clears it when the object is deleted.
    - Implement the `__init__` method to create the cache and print a message.
    - Implement the `__del__` method to clear the cache and print a message.
    - Demonstrate the lifecycle of the `Cache` object in a test case.

Data Hiding

6. **Exercise**: Create a class `CredentialsManager` with private attributes for storing usernames and passwords securely.
    - Use private attributes to store the credentials.
    - Implement methods to set and get the credentials with validation.
    - Include a method to display masked passwords.
7. **Exercise**: Create a class `EmailAccount` with private attributes for the email address and password, and methods to safely access and modify them.
    - Use private attributes to store the email and password.
    - Implement getter and setter methods with validation.
    - Include a method to display the email address and masked password.
8. **Exercise**: Create a class `MedicalRecord` with private attributes for patient information, and methods to safely access and modify it.

    - Use private attributes to store patient information.
    - Implement getter and setter methods with validation.
    - Include a method to display masked patient information.
9. **Exercise**: Create a class `BankAccount` with private attributes for account number and balance, and methods to safely access and modify them.
    - Use private attributes to store the account number and balance.
    - Implement getter and setter methods with validation.
    - Include a method to display the account details.

Multi-Inheritance & MRO

10. **Exercise**: Create a class `TeachingAssistant` that inherits from both `Student` and `Teacher`. Demonstrate method resolution order.
    - Define the `Student` class with a method `study`.
    - Define the `Teacher` class with a method `teach`.
    - Create the `TeachingAssistant` class and demonstrate how methods from both parent classes are resolved.
    - Print the method resolution order.
11. **Exercise**: Create a class `HybridCar` that inherits from both `ElectricCar` and `GasCar`. Demonstrate method resolution order.
    - Define the `ElectricCar` class with a method `charge`.
    - Define the `GasCar` class with a method `refuel`.
    - Create the `HybridCar` class and demonstrate how methods from both parent classes are resolved.
    - Print the method resolution order.
12. **Exercise**: Create a class `SmartWatch` that inherits from both `Watch` and `FitnessTracker`. Demonstrate method resolution order.
    - Define the `Watch` class with a method `show_time`.
    - Define the `FitnessTracker` class with a method `track_steps`.
    - Create the `SmartWatch` class and demonstrate how methods from both parent classes are resolved.
    - Print the method resolution order.

Composition vs Inheritance

13. **Exercise**: Create a class `House` that uses composition with a class `Room`.
    - Define the `Room` class with attributes for dimensions and type of room (e.g., kitchen, bedroom).
    - Define the `House` class with a list of rooms.
    - Implement methods in `House` to add rooms and display details of all rooms.
14. **Exercise**: Create a class `Computer` that uses composition with classes `CPU`, `Memory`, and `Storage`.
    - Define the `CPU` class with attributes for brand and speed.
    - Define the `Memory` class with attributes for size and type.
    - Define the `Storage` class with attributes for size and type.

    - Define the `Computer` class that includes instances of `CPU`, `Memory`, and `Storage`.
    - Implement methods in `Computer` to display details of all components.
15. **Exercise**: Create a class `Robot` that uses composition with classes `Arm`, `Leg`, and `Head`.
    - Define the `Arm` class with attributes for length and strength.
    - Define the `Leg` class with attributes for length and strength.
    - Define the `Head` class with attributes for size and sensors.
    - Define the `Robot` class that includes instances of `Arm`, `Leg`, and `Head`.
    - Implement methods in `Robot` to display details of all components.
16. **Exercise**: Create a class `Car` that uses composition with classes `Engine`, `Transmission`, and `Wheel`.
    - Define the `Engine` class with attributes for horsepower and type.
    - Define the `Transmission` class with attributes for type and number of gears.
    - Define the `Wheel` class with attributes for size and type.
    - Define the `Car` class that includes instances of `Engine`, `Transmission`, and `Wheel`.
    - Implement methods in `Car` to display details of all components.

Properties

17. **Exercise**: Create a class `Rectangle` with properties for width and height, and calculated properties for area and perimeter.
    - Use properties to ensure the width and height are positive numbers.
    - Implement methods to calculate and return the area and perimeter.
    - Include test cases to demonstrate the functionality.
18. **Exercise**: Create a class `Cylinder` with properties for radius and height, and calculated properties for volume and surface area.
    - Use properties to ensure the radius and height are positive numbers.
    - Implement methods to calculate and return the volume and surface area.
    - Include test cases to demonstrate the functionality.
19. **Exercise**: Create a class `Parallelogram` with properties for base, height, and side length, and calculated properties for area and perimeter.
    - Use properties to ensure the base, height, and side length are positive numbers.
    - Implement methods to calculate and return the area and perimeter.
    - Include test cases to demonstrate the functionality.

Abstraction

20. **Exercise**: Create an abstract class `Shape` with abstract methods for calculating the area and the perimeter.
    - Define concrete subclasses `Circle`, `Rectangle`, and `Triangle` that implement the abstract methods.
    - Demonstrate the use of these methods in the subclasses.
21. **Exercise**: Create an abstract class `PaymentMethod` with abstract methods for making a payment and checking the balance.

    - Define concrete subclasses `CreditCard`, `DebitCard`, and `PayPal` that implement the abstract methods.
    - Demonstrate the use of these methods in the subclasses.
22. **Exercise**: Create an abstract class `Animal` with abstract methods for making sound and moving.
    - Define concrete subclasses `Dog` and `Bird` that implement the abstract methods.
    - Demonstrate the use of these methods in the subclasses.
23. **Exercise**: Create an abstract class `Vehicle` with abstract methods for starting, stopping, and accelerating.
    - Define concrete subclasses `Car`, `Bike`, and `Truck` that implement the abstract methods.
    - Demonstrate the use of these methods in the subclasses.

Mixin

24. **Exercise**: Create a mixin class `TimestampMixin` that adds created and updated timestamps to other classes.
    - Implement methods `set_created_timestamp` and `set_updated_timestamp` in the mixin.
    - Use the mixin in a class `Document` and demonstrate timestamp functionality.
25. **Exercise**: Create a mixin class `AuditableMixin` that adds audit trail functionality to other classes.
    - Implement a method `audit` in the mixin to log changes.
    - Use the mixin in a class `Transaction` and demonstrate audit trail functionality.
26. **Exercise**: Create a mixin class `LoggableMixin` that adds logging functionality to other classes.
    - Implement a method `log` in the mixin to log messages.
    - Use the mixin in a class `Service` and demonstrate logging functionality.
27. **Exercise**: Create a mixin class `SerializableMixin` that adds serialization and deserialization functionality to other classes.
    - Implement methods `serialize` and `deserialize` in the mixin.
    - Use the mixin in a class `User` and demonstrate serialization and deserialization functionality.
28. **Exercise**: Create a mixin class `EncryptableMixin` that adds encryption and decryption functionality to other classes.
    - Implement methods `encrypt` and `decrypt` in the mixin.
    - Use the mixin in a class `Message` and demonstrate encryption and decryption functionality.
29. **Exercise**: Create a mixin class `VersionableMixin` that adds version control functionality to other classes.
    - Implement methods `save_version` and `get_version` in the mixin.
    - Use the mixin in a class `File` and demonstrate version control functionality.