```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.available = True  # Default availability is True

    def borrow(self):
        if self.available:
            self.available = False
            print(f"You have borrowed '{self.title}'.")
        else:
            print(f"Sorry, '{self.title}' is already borrowed.")

    def return_book(self):
        if not self.available:
            self.available = True
            print(f"You have returned '{self.title}'.")
        else:
            print(f"'{self.title}' was not borrowed.")

# Creating book objects
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald")
book2 = Book("1984", "George Orwell")

# Demonstration of borrowing and returning
book1.borrow()  # Borrow "The Great Gatsby"
book1.borrow()  # Attempt to borrow again (should show it's already borrowed)
book1.return_book()  # Return the book
book1.return_book()  # Attempt to return again (should show it's not borrowed)

print("\n")  # Line break for clarity

book2.borrow()  # Borrow "1984"
book2.return_book()  # Return "1984"
```

```
You have borrowed 'The Great Gatsby'.
Sorry, 'The Great Gatsby' is already borrowed.
You have returned 'The Great Gatsby'.
'The Great Gatsby' was not borrowed.


You have borrowed '1984'.
You have returned '1984'.
```

```
class Student:
    university = "ABC University"  # Class variable (shared by all instances)

    def __init__(self, name, age):
        self.name = name  # Instance variable
        self.age = age  # Instance variable

    @classmethod
    def update_university(cls, new_university):
        cls.university = new_university  # Updates class variable

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}, University: {self.university}")

# Creating student objects
student1 = Student("Alice", 20)
student2 = Student("Bob", 22)

# Displaying student details before changing the university
print("Before updating university:")
student1.display_info()
student2.display_info()
```

```
# Updating the university name for all students
Student.update_university("XYZ University")

# Displaying student details after changing the university
print("\nAfter updating university:")
student1.display_info()
student2.display_info()
```

```
Before updating university:
Name: Alice, Age: 20, University: ABC University
Name: Bob, Age: 22, University: ABC University

After updating university:
Name: Alice, Age: 20, University: XYZ University
Name: Bob, Age: 22, University: XYZ University
```

```
class Employee:
    employee_count = 0  # Class variable to track total employees

    def __init__(self, name, salary):
        self.name = name  # Instance variable
        self.salary = salary  # Instance variable
        Employee.employee_count += 1  # Increment employee count when a new employee is created

    @classmethod
    def total_employees(cls):
        return cls.employee_count  # Return total employee count

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

# Creating multiple employee objects
employee1 = Employee("Alice", 50000)
employee2 = Employee("Bob", 60000)
employee3 = Employee("Charlie", 55000)

# Displaying employee details
employee1.display_info()
employee2.display_info()
employee3.display_info()

# Displaying the total number of employees
print(f"\nTotal Employees: {Employee.total_employees()}")
```

```
Name: Alice, Salary: $50000
Name: Bob, Salary: $60000
Name: Charlie, Salary: $55000

Total Employees: 3
```

```
class MathOperations:
    @staticmethod
    def add_numbers(a, b):
        return a + b  # Returns the sum of two numbers

# Calling the static method using the class name
result1 = MathOperations.add_numbers(5, 10)
print(f"Sum using class name: {result1}")

# Creating an object and calling the static method
math_util = MathOperations()
result2 = math_util.add_numbers(7, 3)
print(f"Sum using object: {result2}")
```

```
Sum using class name: 15
Sum using object: 10
```

```
# Base class
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")

# Derived class (inherits from Vehicle)
class Car(Vehicle):
    def __init__(self, brand, model, number_of_doors):
        super().__init__(brand, model)  # Call parent constructor
        self.number_of_doors = number_of_doors

    def display_info(self):
        super().display_info()  # Call parent method
        print(f"Number of Doors: {self.number_of_doors}")

# Further derived class (inherits from Car)
class SportsCar(Car):
    def __init__(self, brand, model, number_of_doors, top_speed):
        super().__init__(brand, model, number_of_doors)  # Call Car constructor
        self.top_speed = top_speed

    def display_info(self):
        super().display_info()  # Call Car's display_info
        print(f"Top Speed: {self.top_speed} km/h")

# Demonstration: Creating a SportsCar object
sports_car = SportsCar("Ferrari", "488 GTB", 2, 330)

# Displaying sports car details
sports_car.display_info()
```

```
Brand: Ferrari, Model: 488 GTB
Number of Doors: 2
Top Speed: 330 km/h
```

```
class BankAccount:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance  # Private variable to store balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: ${amount:.2f}")
```

```
            else:
                print("Deposit amount must be positive.")

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

        def get_balance(self):
            return self.__balance  # Returns the current balance

    # Demonstration
    account = BankAccount(100)  # Starting with $100

    # Performing transactions
    account.deposit(50)         # Deposit $50
    account.withdraw(30)        # Withdraw $30
    account.withdraw(200)       # Attempt to withdraw more than the balance
    account.deposit(-20)        # Invalid deposit attempt

    # Checking balance
    print(f"Final Balance: ${account.get_balance():.2f}")
```

```
Deposited: $50.00
Withdrawn: $30.00
Insufficient funds!
Deposit amount must be positive.
Final Balance: $120.00
```

```
    import time

    # Timer decorator
    def timer(func):
        def wrapper():
            start_time = time.time()  # Record start time
            func()  # Execute the function
            end_time = time.time()  # Record end time
            print(f"Execution time: {end_time - start_time:.2f} seconds")
        return wrapper

    # Applying the decorator
    @timer
    def slow_function():
        print("Starting slow function...")
        time.sleep(2)  # Simulating a delay of 2 seconds
        print("Slow function finished.")

    # Calling the function
    slow_function()
```

```
Starting slow function...
Slow function finished.
Execution time: 2.00 seconds
```

```
    def generate_numbers(n):
        for i in range(1, n + 1):  # Loop from 1 to n
            yield i  # Yield each number one by one

    # Iterating over the generator without using a list
    for num in generate_numbers(10):  # Example with n = 10
```

```
    print(num)
```

```
1
2
3
4
5
6
7
8
9
10
```

```
def read_file(filename):
    file = None  # Initialize file variable
    try:
        file = open(filename, "r")  # Open the file in read mode
        content = file.read()  # Read the file content
        print("File Contents:\n", content)
    except FileNotFoundError:
        print("Error: File not found!")
    finally:
        if file:  # Ensure the file is closed if it was opened
            file.close()
            print("File closed.")

# Testing with an existing file
read_file("example.txt")  # Replace with a valid filename

# Testing with a non-existent file
read_file("missing.txt")  # This file does not exist
```

```
Error: File not found!
Error: File not found!
```

```
import pickle

# User class with username and password
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password  # In a real system, use hashing!

    def display_user(self):
        print(f"Username: {self.username}, Password: {self.password}")

# Function to save user data (Serialization)
def save_user(user, filename):
    with open(filename, "wb") as file:
        pickle.dump(user, file)  # Serialize and save

# Function to load user data (Deserialization)
def load_user(filename):
    try:
        with open(filename, "rb") as file:
            return pickle.load(file)  # Deserialize and return user object
    except FileNotFoundError:
        print("Error: File not found!")
        return None

# Creating a user object
user1 = User("john_doe", "secure123")

# Saving the user object
save_user(user1, "user_data.pkl")
```

```
print("User data saved successfully!\n")

# Loading the user object
loaded_user = load_user("user_data.pkl")

# Displaying user details
if loaded_user:
    print("Loaded User Details:")
    loaded_user.display_user()
```

```
User data saved successfully!

 Loaded User Details:
 Username: john_doe, Password: secure123
```

```
import matplotlib.pyplot as plt

# Function to copy an image file
def copy_image(source, destination):
    try:
        with open(source, "rb") as src_file:  # Read in binary mode
            image_data = src_file.read()  # Read the entire file

        with open(destination, "wb") as dest_file:  # Write in binary mode
            dest_file.write(image_data)  # Write data to new file

        print(f"Image copied successfully to {destination}")
    except FileNotFoundError:
        print("Error: Source file not found!")

# Function to display images using matplotlib
def display_images(original, copied):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))

    # Read and display the original image
    img1 = plt.imread(original)
    axes[0].imshow(img1)
    axes[0].set_title("Original Image")
    axes[0].axis("off")

    # Read and display the copied image
    img2 = plt.imread(copied)
    axes[1].imshow(img2)
    axes[1].set_title("Copied Image")
    axes[1].axis("off")

    plt.show()

# Specify file paths
original_image = "sample.jpg"  # Replace with your actual image file
copied_image = "copy_sample.jpg"

# Copy the image
copy_image(original_image, copied_image)

# Display both images
display_images(original_image, copied_image)
```

```
Error: Source file not found!
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-11-245bda36fc62> in <cell line: 0>()
     40
     41 # Display both images
---> 42 display_images(original_image, copied_image)

 3 frames
/usr/local/lib/python3.11/dist-packages/PIL/Image.py in open(fp, mode, formats)
   3463
   3464     if filename:
-> 3465         fp = builtins.open(filename, "rb")
   3466         exclusive_fp = True
   3467     else:

FileNotFoundError: [Errno 2] No such file or directory: 'sample.jpg'
```