# Задание 1. Класс Автомобиль  
class Car:  
def *init*(self):  
self.wheels = 0  
self.model = ""  
self.speed = 0  
  
def set\_parameters(self, wheels, model, speed):  
self.wheels = wheels  
self.model = model  
self.speed = speed  
  
def display\_info(self):  
print(f"Model: {self.model}, Wheels: {self.wheels}, Speed: {self.speed} km/h")  
  
car1 = Car()  
car1.set\_parameters(4, "Sedan", 120)  
car1.display\_info()  
  
car2 = Car()  
car2.set\_parameters(2, "Motorcycle", 150)  
car2.display\_info()  
  
# Задание 2. Создание полей класса  
class EmptyClass:  
pass  
  
obj1 = EmptyClass()  
obj2 = EmptyClass()  
obj2.newValue = 5  
  
# Задание 3. Небольшой класс  
class SimpleClass:  
def *init*(self, text):  
self.text = text  
  
def display\_text(self, parameter):  
print(self.text + parameter)  
  
obj3 = SimpleClass("Hello, ")  
obj3.display\_text("world")  
  
# Простые задачи  
# 1. Класс Точка  
class Point:  
def *init*(self, x, y):  
self.x = x  
self.y = y  
  
def display\_coordinates(self):  
print(f"Coordinates: ({self.x}, {self.y})")  
  
point = Point(3, 5)  
point.display\_coordinates()  
  
# 2. Класс Прямоугольник  
class Rectangle:  
def *init*(self, width, height):  
self.width = width  
self.height = height  
  
def calculate\_area(self):  
return self.width \* self.height  
  
rectangle = Rectangle(4, 6)  
print("Rectangle Area:", rectangle.calculate\_area())  
  
# 3. Класс Студент  
class Student:  
def *init*(self, name, surname, year\_of\_study):  
self.name = name  
self.surname = surname  
self.year\_of\_study = year\_of\_study  
  
def display\_info(self):  
print(f"Student: {self.name} {self.surname}, Year of Study: {self.year\_of\_study}")  
  
student = Student("John", "Doe", 2023)  
student.display\_info()  
  
# Средние задачи  
# 1. Класс Автомобиль с наследованием  
class Automobile:  
def *init*(self, brand):  
self.brand = brand  
  
class Truck(Automobile):  
def *init*(self, brand, cargo\_capacity):  
super().*init*(brand)  
self.cargo\_capacity = cargo\_capacity  
  
class Sedan(Automobile):  
def *init*(self, brand, passengers\_capacity):  
super().*init*(brand)  
self.passengers\_capacity = passengers\_capacity  
  
truck = Truck("Volvo", 5000)  
sedan = Sedan("Toyota", 5)  
  
# 2. Класс Счетчик  
class Counter:  
def *init*(self, initial\_value):  
self.value = initial\_value  
  
def increment(self):  
self.value += 1  
  
def decrement(self):  
self.value -= 1  
  
def get\_value(self):  
return self.value  
  
counter = Counter(0)  
counter.increment()  
counter.decrement()  
print("Counter Value:", counter.get\_value())  
  
# 3. Класс Телефонная книга  
class PhoneBook:  
def *init*(self):  
self.contacts = {}  
  
def add\_contact(self, name, phone\_number):  
self.contacts[name] = phone\_number  
  
def remove\_contact(self, name):  
if name in self.contacts:  
del self.contacts[name]  
  
def search\_contact(self, name):  
return self.contacts.get(name, "Contact not found")  
  
phone\_book = PhoneBook()  
phone\_book.add\_contact("John", "123-456-7890")  
phone\_book.add\_contact("Jane", "987-654-3210")  
phone\_book.remove\_contact("Jane")  
print("John's Phone Number:", phone\_book.search\_contact("John"))  
  
# Сложные задачи  
# 1. Класс Банковский счет с инкапсуляцией  
class BankAccount:  
def *init*(self):  
self.\_\_balance = 0  
  
def deposit(self, amount):  
self.\_\_balance += amount  
  
def withdraw(self, amount):  
if amount <= self.\_\_balance:  
self.\_\_balance -= amount  
else:  
print("Insufficient funds")  
  
def check\_balance(self):  
return self.\_\_balance  
  
account = BankAccount()  
account.deposit(1000)  
account.withdraw(500)  
print("Account Balance:", account.check\_balance())  
  
# 2. Класс Книжная библиотека  
class Book:  
def *init*(self, title, author):  
self.title = title  
self.author = author  
self.is\_available = True

class Reader:  
def *init*(self, name):  
self.name = name  
self.borrowed\_books = []  
  
class Library:  
def *init*(self):  
self.books = []  
  
def register\_book(self, title, author):  
book = Book(title, author)  
self.books.append(book)  
  
def lend\_book(self, book, reader):  
if book.is\_available:  
book.is\_available = False  
reader.borrowed\_books.append(book)  
print(f"{reader.name} borrowed {book.title} by {book.author}")  
else:  
print(f"{book.title} by {book.author} is not available")  
  
def return\_book(self, book, reader):  
if book in reader.borrowed\_books:  
book.is\_available = True  
reader.borrowed\_books.remove(book)  
print(f"{reader.name} returned {book.title} by {book.author}")  
else:  
print(f"{reader.name} did not borrow {book.title}")  
  
library = Library()  
library.register\_book("The Great Gatsby", "F. Scott Fitzgerald")  
library.register\_book("To Kill a Mockingbird", "Harper Lee")  
  
reader = Reader("Alice")  
library.lend\_book(library.books[0], reader)  
library.return\_book(library.books[0], reader)  
  
# 3. Класс Геометрические фигуры с полиморфизмом  
from abc import ABC, abstractmethod  
  
class Shape(ABC):  
@abstractmethod  
def calculate\_area(self):  
pass  
  
class Circle(Shape):  
def *init*(self, radius):  
self.radius = radius  
  
def calculate\_area(self):  
return 3.14 \* self.radius \*\* 2  
  
class Rectangle(Shape):  
def *init*(self, width, height):  
self.width = width  
self.height = height  
  
def calculate\_area(self):  
return self.width \* self.height  
  
class Triangle(Shape):  
def *init*(self, base, height):  
self.base = base  
self.height = height  
  
def calculate\_area(self):  
return 0.5 \* self.base \* self.height  
  
circle = Circle(5)  
rectangle = Rectangle