ЛАБОРАТОРНАЯ РАБОТА «ОБЪЕКТНО-ОРИЕНТОРОВАННОЕ ПРОГРАММИРОВАНИЕ НА PYTHON»

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ.

Написать программы к задачам, указанным в вашем варианте. В каждой задаче даются разные предметные области. Нужно ОБЯЗАТЕЛЬНО РАСШИРИТЬ в задаче каждый класс путем добавления атрибутов, свойств, гетеров, сетеров, и дополнительных методов, а также придумать и переопределить операции над некоторыми объектами классов.

В результате каждая задача должна иметь законченный вид. Должны быть созданы объекты, введены данные, осуществлены какие-то действия над объектами, что должно подтверждаться выводом результатов на экран и зафиксировано в виде скриншотов выполнения программы.

Распределение задач по вариантам

№ варианта	Задача 1	Задача 2	Задача 3	Задача 4	Задача 5	Задача 6	Задача 7	Задача 8	Задача 9	Задача 10	Задача 11	Задача 12	Задача 13	Задача 14	Задача 15	Задача 16	Задача 17	Задача 18	Задача 19	Задача 20
1.	*							*				*							*	
2.		*							*				*					*		
3.			*					*		*				*						
4.				*					*		*				*					
5.	*				*							*				*				
6.		*				*							*				*			
7.			*				*							*				*		
8.				*				*							*				*	
9.					*				*							*				*
10.	*					*				*							*			
11.		*					*				*							*		
12.			*									*				*			*	
13.				*									*				*			*
14.					*					*				*				*		
15.						*					*				*				*	
16.					*		*									*				*
17.	*							*				*					*			
18.		*				*			*				*							
19.			*				*			*				*						
20.				*							*				*					*
21.	*				*				*					*						
22.		*				*				*					*					
23.			*				*				*					*				
24.				*				*				*					*			
25.					*				*				*					*		
26.						*				*				*					*	
27.		*					*											*		*
28.			*					*				*			*					
29.					*				*				*			*				
30.	*									*							*			*

СПИСОК ЗАДАЧ

Задача 1. Класс "Животные"

Создать класс `Animal` с методами `move()` и `speak()`. Создать дочерние классы `Dog`, `Cat`, `Bird`, переопределив методы родительского класса для каждого вида животных.

```
class Animal:
    def move(self):
        pass

    def speak(self):
        pass

class Dog(Animal):
    def move(self):
        return "Собака бежит"

    def speak(self):
        return "Гав-гав!"
```

Задача 2. Класс "Фигуры"

Создать базовый класс `Shape` с абстрактным методом `area()`. Реализовать подклассы `Circle`, `Rectangle`, `Triangle`.

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2
```

Задача 3. Полиморфизм в расчетах

Реализуйте функцию `calculate_area(shapes)`, которая принимает список объектов фигур (`Shape`) и возвращает общую площадь всех фигур.

```
def calculate_area(shapes):
    total_area = 0
    for shape in shapes:
        total_area += shape.area()
    return total_area
```

Задача 4. Класс "Транспорт"

Создать базовый класс `Vehicle` с атрибутами `speed` и `fuel`. Реализовать подклассы `Car`, `Bike`, `Truck`, каждый из которых имеет уникальный метод движения.

```
class Vehicle:
    def __init__(self, speed, fuel):
        self.speed = speed
```

```
        self.fuel = fuel

class Car(Vehicle):
    def drive(self):
        print("Машина едет")
```

Задача 5. Интерфейсы и полиморфизм

Создайте интерфейс `Movable` с методом `move()`. Реализуйте несколько классов, которые реализуют этот интерфейс, например, `Human`, `Robot`, `Animal`.

```
class Movable(ABC):
    @abstractmethod
    def move(self):
        pass

class Human(Movable):
    def move(self):
        return "Человек идет"
```

Задача 6. Наследование и конструкторы

Создайте класс `Person` с атрибутами `name` и `age`. Затем создайте класс `Employee`, который наследует от `Person` и добавляет атрибут `job_title`.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Employee(Person):
    def __init__(self, name, age, job_title):
        super().__init__(name, age)
        self.job_title = job_title
```

Задача 7. Абстрактный класс "Форма"

Создайте абстрактный класс `Form` с методами `draw()` и `erase()`. Реализуйте подклассы `Square`, `Circle`, `Line`, где каждый реализует эти методы.

```
from abc import ABC, abstractmethod

class Form(ABC):
    @abstractmethod
    def draw(self):
        pass

    @abstractmethod
    def erase(self):
        pass

class Square(Form):
    def draw(self):
        print("Рисуем квадрат")

    def erase(self):
        print("Стираем квадрат")
```

Задача 8. Класс "Птицы"

Создайте класс `Bird` с методами `fly()` и `sing()`. Реализуйте подклассы `Eagle`, `Parrot`, `Penguin`, переопределяя методы полета и пения.

```
class Bird:
    def fly(self):
        pass

    def sing(self):
        pass

class Eagle(Bird):
    def fly(self):
        return "Орел летит высоко"

    def sing(self):
        return "Крик орла"
```

Задача 9. Класс "Банковский счет"

Создайте класс `Account` с методами `deposit()` и `withdraw()`. Реализуйте подкласс `SavingsAccount`, который добавляет процентную ставку на остаток счета.

```
class Account:
    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Недостаточно средств")

class SavingsAccount(Account):
    def __init__(self, balance=0, interest_rate=0.05):
        super().__init__(balance)
        self.interest_rate = interest_rate

    def apply_interest(self):
        self.balance *= (1 + self.interest_rate)
```

Задача 10. Класс "Геометрические фигуры"

Создайте базовый класс `GeometricFigure` с абстрактным методом `perimeter()`. Реализуйте подклассы `Rectangle`, `Circle`, `Triangle`.

```
from abc import ABC, abstractmethod
class GeometricFigure(ABC):
    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(GeometricFigure):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def perimeter(self):
        return 2 * (self.width + self.height)
```

Задача 11. Полиморфизм в обработке данных

Напишите функцию `process_data(objects)`, которая принимает список объектов различных типов и вызывает разные методы в зависимости от типа объекта.

```
def process_data(objects):
    for obj in objects:
        if isinstance(obj, Circle):
            obj.draw_circle()
        elif isinstance(obj, Rectangle):
            obj.draw_rectangle()
        else:
            print(f"Неизвестный тип объекта: {type(obj).__name__}")
```

Задача 12. Класс "Строки"

Создайте класс `StringProcessor` с методами `uppercase()`, `lowercase()`, `reverse()`. Реализуйте подкласс `AdvancedStringProcessor`, добавляющий метод `palindrome_check()`.

```
class StringProcessor:
    def __init__(self, string):
        self.string = string

    def uppercase(self):
        return self.string.upper()

    def lowercase(self):
        return self.string.lower()

    def reverse(self):
        return self.string[::-1]

class AdvancedStringProcessor(StringProcessor):
    def palindrome_check(self):
        reversed_string = self.reverse()
        return self.string == reversed_string
```

Задача 13. Класс "Транспортные средства"

Создайте базовый класс `Transportation` с методами `start_engine()` и `stop_engine()`. Реализуйте подклассы `Car`, `Motorcycle`, `Train`.

```
class Transportation:
    def start_engine(self):
        pass

    def stop_engine(self):
        pass

class Car(Transportation):
    def start_engine(self):
        print("Запуск двигателя автомобиля")

    def stop_engine(self):
        print("Остановка двигателя автомобиля")
```

Задача 14. Класс "Деревья"

Создайте класс `Tree` с методами `grow()` и `die()`. Реализуйте подклассы `Oak`, `Pine`, `Maple`, добавив уникальные свойства каждому виду деревьев.

```
class Tree:
    def grow(self):
        pass

    def die(self):
        pass

class Oak(Tree):
    def grow(self):
        print("Дуб растет медленно, но уверенно")

    def die(self):
        print("Дуб умирает, оставляя после себя мощные корни")
```

Задача 15. Класс "Школьники"

Создайте класс `Student` с методами `study()` и `play()`. Реализуйте подклассы `ElementaryStudent`, `HighSchoolStudent`, `CollegeStudent`, переопределяя методы в каждом классе.

```
class Student:
    def study(self):
        pass

    def play(self):
        pass

class ElementaryStudent(Student):
    def study(self):
        print("Ученик начальной школы учится читать и писать")

    def play(self):
        print("Ученик начальной школы играет в подвижные игры")
```

Задача 16. Класс "Магазин"

Создайте класс `Store` с методами `add_item()`, `remove_item()`, `display_inventory()`. Реализуйте подклассы `GroceryStore`, `ClothingStore`, `ElectronicsStore`.

```
class Store:
    def add_item(self, item):
        pass

    def remove_item(self, item):
        pass

    def display_inventory(self):
        pass

class GroceryStore(Store):
    def add_item(self, item):
        print(f"Добавлен продукт питания: {item}")
```

Задача 17. Класс "Спортсмены"

Создайте класс `Athlete` с методами `train()` и `compete()`. Реализуйте подклассы `Sprinter`, `MarathonRunner`, `Swimmer`.

```
class Athlete:
    def train(self):
        pass
```

```
    def compete(self):
        pass

class Sprinter(Athlete):
    def train(self):
        print("Спринтер тренируется в беге на короткие дистанции")

    def compete(self):
        print("Спринтер участвует в соревнованиях по бегу на 100 метров")
```

Задача 18.Класс "Финансовая операция"

Создайте базовый класс FinancialOperation с методами execute() и revert(). Реализуйте подклассы Deposit, Withdrawal, Transfer, каждый из которых представляет различные типы финансовых операций.

```
class FinancialOperation:
    def execute(self):
        pass

    def revert(self):
        pass

class Deposit(FinancialOperation):
    def __init__(self, account, amount):
        self.account = account
        self.amount = amount

    def execute(self):
        self.account.deposit(self.amount)

    def revert(self):
        self.account.withdraw(self.amount)
```

Задача 19.Класс "Компьютерная игра"

Создайте класс Game с методами start_game(), save_progress(), load_progress(). Реализуйте подклассы RPGGame, FPSGame, StrategyGame, где каждый класс реализует уникальные особенности жанра игр.

```
class Game:
    def start_game(self):
        pass

    def save_progress(self):
        pass

    def load_progress(self):
        pass

class RPGGame(Game):
    def start_game(self):
        print("Начало ролевой игры")

    def save_progress(self):
        print("Сохранение прогресса в ролевой игре")

    def load_progress(self):
        print("Загрузка сохраненного прогресса в ролевой игре")
```

Задача 20. Класс "Электронные устройства"

Создайте класс ElectronicDevice с методами turn_on(), turn_off(), charge(). Реализуйте подклассы Smartphone, Laptop, Tablet, добавляя специфичные функции для каждого устройства.

```
class ElectronicDevice:
    def turn_on(self):
        pass

    def turn_off(self):
        pass

    def charge(self):
        pass

class Smartphone(ElectronicDevice):
    def take_photo(self):
        print("Сделан снимок на смартфон")

    def make_call(self):
        print("Звонок через смартфон")
```

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ ДОЛЖЕН ВКЛЮЧАТЬ

1. Титульный лист по форме с номером варианта.
2. Для каждой задачи из списка задач по вариантам (4 задачи по вариантам):
    a. Условие задачи.
    b. Программный код решения этой задачи (листинг).
    c. Скриншоты выполнения программы.

Внимание! Отчет должен быть набран шрифтом **Times New Roman** и отформатирован: поля: левое – 3,5; правое – 1,5; нижнее и верхнее – 2 см; красная строка (отступ) - 1 см; межстрочный интервал – одинарный; правый край выровнен по ширине; рисунки сопровождены надписями.