Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических
систем и электроники института перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Плугатырев Владислав Алексеевич
3 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Доцент кафедры инфокоммуникаций:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Наследование и полиморфизм в языке Python.

Цель работы: : приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Ход работы

1. Создание репозитория.

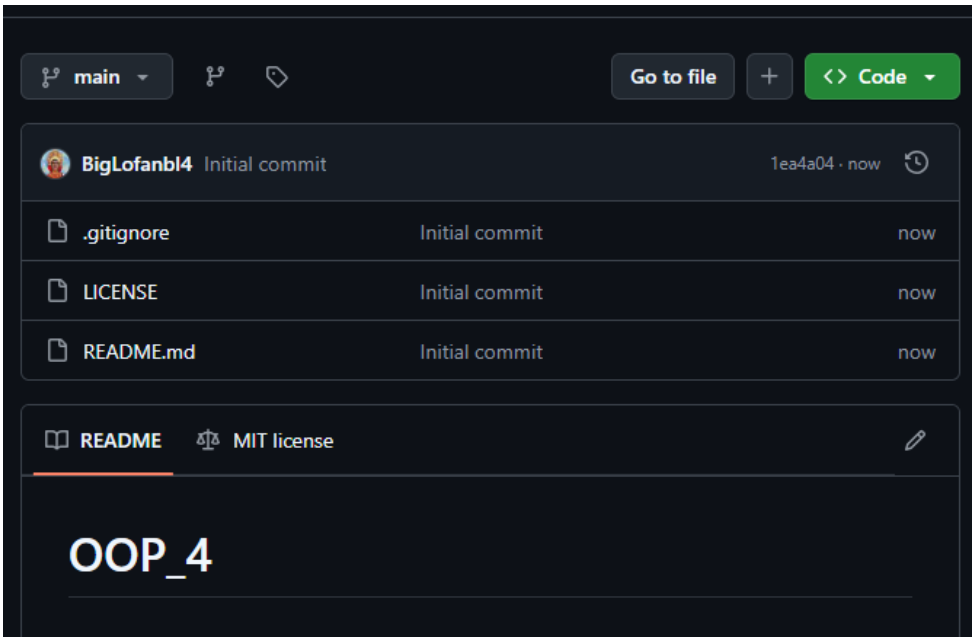

Рисунок 1.1 – Созданный репозиторий

2. Выполнение примеров из лабораторной работы.

```python
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Знаменатель не может быть нулем.")
        self.__numerator = a
        self.__denominator = b
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            while b:
                a, b = b, a % b
            return a

        c = gcd(self.__numerator, self.__denominator)
        self.__numerator //= c
        self.__denominator //= c

    @property
    def numerator(self):
        return self.__numerator

    @property
    def denominator(self):
        return self.__denominator
```

Рисунок 1.2 – Код 1 примера

```python
from abc import ABC, abstractmethod


class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass


class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")


class Pentagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")


class Hexagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")


class Quadrilateral(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")
```

Рисунок 1.3 – Код второго примера

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


class Human(Animal):
    def move(self):
        print("I can walk and run")


class Snake(Animal):
    def move(self):
        print("I can crawl")


class Dog(Animal):
    def move(self):
        print("I can bark")


class Lion(Animal):
    def move(self):
        print("I can roar")
```

Рисунок 1.4 – Код третьего примера

3. Выполнение первого задания.

```python
class Human:
    def __init__(self, id, team):
        self.id = id
        self.team = team


class Soldier(Human):
    def __init__(self, id, team):
        super().__init__(id, team)

    def follow_hero(self, hero):
        print(f"Soldier {self.id} follows {hero.id} team {self.team}")


class Hero(Human):
    def __init__(self, id, team):
        super().__init__(id, team)
        self.level = 1

    def increase_level(self):
        self.level += 1
```

Рисунок 1.5 – Код программы

```
Team blue has 23 soldiers
Team red has 27 soldiers
Soldier 1 follows red_hero team Red
red_hero 1
```

Рисунок 1.6 – Вывод программы

4. Выполнение первого индивидуального задания: создать класс Triad (тройка чисел); определить методы увеличения полей на 1 Определить производный класс Date с полями: год, месяц и день. Переопреде лить методы увеличения полей на 1 и определить метод увеличения даты на n дней.

```python
class Triad:
    def __init__(self, num_1, num_2, num_3):
        self.num_1 = num_1
        self.num_2 = num_2
        self.num_3 = num_3

    def increment_num_1(self):
        self.num_1 += 1

    def increment_num_2(self):
        self.num_2 += 1

    def increment_num_3(self):
        self.num_3 += 1

    def __repr__(self):
        return f"({self.num_1}, {self.num_2}, {self.num_3})"


class Date(Triad):
    def __init__(self, year, month, day):
        super().__init__(year, month, day)

    # увеличение года на 1
    def increment_num_1(self):
        self.num_1 += 1

    # увеличение месяца на 1
    def increment_num_2(self):
        self.num_2 += 1
        if self.num_2 > 12:
            self.num_2 = 1
            self.increment_num_1()

    # увеличение дней на 1
    def increment_num_3(self):
        self.add_days(1)

    # увеличение на n дней
    def add_days(self, n):
        new_date = datetime(self.num_1, self.num_2, self.num_3) + timedelta(
            days=n
        )
        self.num_1 = new_date.year
        self.num_2 = new_date.month
        self.num_3 = new_date.day
```

Рисунок 1.7 – Код первого индивидуального задания

```
(2022, 1, 11)
(2022, 1, 16)
```

Рисунок 1.8 - Вывод

5. Выполнение второго индивидуального задания: Создать абстрактный базовый класс Series (прогрессия) с виртуальными функциями вычисления -го элемента прогрессии и суммы прогрессии. Определить производные классы: Linear (арифметическая) и Exponential (геометрическая).

```python
class Linear(Series):
    def __init__(self, a1, d):
        self.a1 = a1
        self.d = d

    def get_element(self, n):
        return self.a1 + (n - 1) * self.d

    def get_sum(self, n):
        return n * (self.a1 + self.get_element(n)) / 2


class Exponential(Series):
    def __init__(self, a1, q):
        self.a1 = a1
        self.q = q

    def get_element(self, n):
        return self.a1 * (self.q ** (n - 1))

    def get_sum(self, n):
        if self.q == 1:
            return n * self.a1
        else:
            return self.a1 * (1 - self.q**n) / (1 - self.q)
```

Рисунок 1.9 – Код второго индивидуального задания

```
16
235.0
54
59048.0
```

Рисунок 2.1 – Вывод

Ответы на контрольные вопросы

1. Наследование в Python: Это механизм, позволяющий новому классу (наследнику или подклассу) перенимать свойства и методы другого класса (родительского или суперкласса). Подклассы могут переопределять или расширять функциональность родительского класса. В Python наследование

реализовано указанием родительского класса в скобках при определении нового класса.

2. Полиморфизм в Python: Это идиома, использующаяся для реализации функций или методов, способных обрабатывать данные различных типов. В Python полиморфизм проявляется в способности работать с объектами различных классов с использованием общего интерфейса. Полиморфизм реализуется за счет того, что функции и объекты не строго типизированы.

3. “Утиная” типизация (Duck Typing) в Python: Этот термин основан на фразе "если что-то ходит как утка и крякает как утка, то это, вероятно, утка". В контексте программирования это означает, что если объект реализует необходимый интерфейс (методы и свойства), то его можно использовать в качестве такового, независимо от того, каким классом он является. Python полагается на "утиную" типизацию, позволяя различным объектам использоваться в одном и том же контексте, если у них есть соответствующие методы и свойства.

4. Модуль abc (Abstract Base Classes) в Python: Этот модуль предоставляет инфраструктуру для определения абстрактных базовых классов, которые могут определять общие интерфейсы для набора подклассов. Это помогает в создании более надежного и легкого в использовании кода за счет предоставления классов-заготовок, которые требуют определенных методов у всех подклассов.

5. Создание абстрактного метода класса: Для создания абстрактного метода используются модуль «abc» и декоратор «abstractmethod». Абстрактный метод — это метод класса, который определен, но не реализован и должен быть переопределен в неабстрактных подклассах.

6. Создание абстрактного свойства класса: Механизм создания абстрактного свойства схож с созданием абстрактного метода, но используется декоратор «abstractproperty».

7. Назначение функции «isinstance»: Функция «isinstance» используется для проверки принадлежности объекта к классу или кортежу классов (для

поддержки наследования). Это полезно, когда вы хотите проверить, является ли объект экземпляром определенного класса или его подкласса.