

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №3
Наследование и полиморфизм в языке Python

по дисциплине «Объектно-ориентированное программирование»

Выполнил студент группы ИВТ-б-о-21-1

Павленко М.С. « »_____20__г.

Подпись студента_____

Работа защищена« »_____20__г.

Проверил Воронкин Р.А. _____

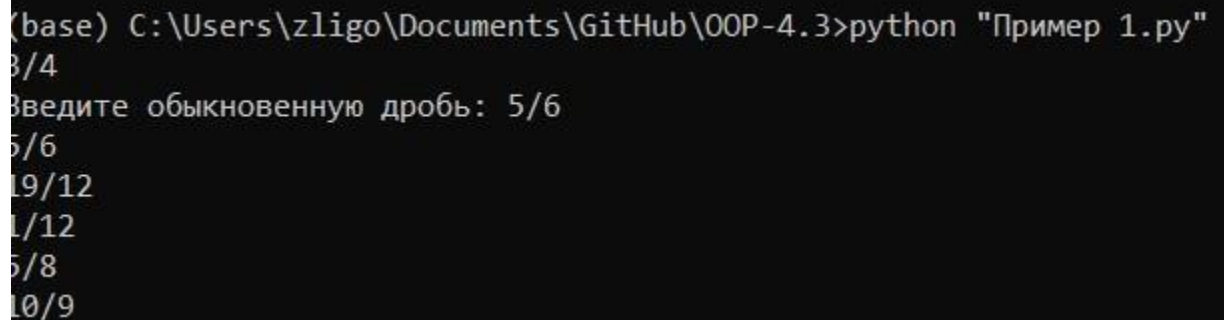
(подпись)

Ставрополь 2023

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

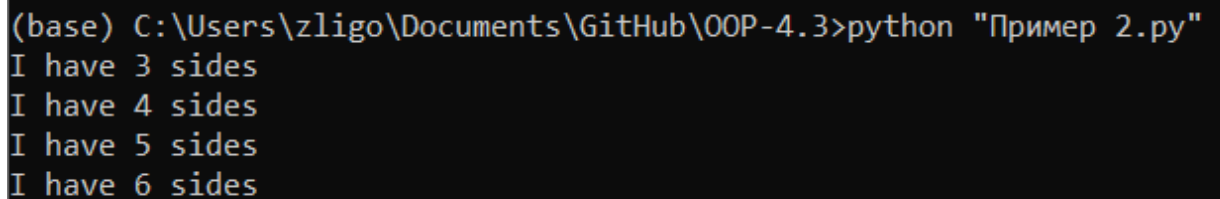
Ход работы

1. Изучив методические указания, приступил к разбору примеров.



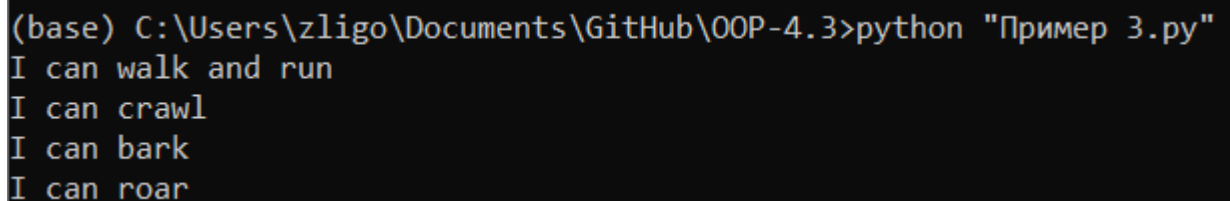
```
(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Пример 1.py"
3/4
Введите обыкновенную дробь: 5/6
5/6
19/12
1/12
5/8
10/9
```

Рисунок 3.1 – Разбор работы кода первого примера



```
(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Пример 2.py"
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Рисунок 3.2 – Разбор работы кода второго примера



```
(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Пример 3.py"
I can walk and run
I can crawl
I can bark
I can roar
```

Рисунок 3.3 – Разбор работы кода третьего примера

2. Затем приступил к выполнению общего задания, используя изученные по теоретическим документам и примерам новые методы работы с классами и объектами.

```

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

class Hero(Human):
    def __init__(self, team, id):
        super().__init__(team, id)
        self.lvl = 0
        self.detachment = []

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

    def move(self, hero):
        hero.detachment.append(self.id)

def main():
    hero1 = Hero(0, 0)
    hero2 = Hero(1, 1)
    teams = {
        hero1.team: {
            "id Героя": hero1.id,
            "id Солдат": []
        },
        hero2.team: {
            "id Героя": hero2.id,
            "id Солдат": []
        }
    }
    list_sold = {}
    for i in range(2, 12):
        team = random.randint(0, 1)
        list_sold[i] = Soldier(team, i)
        teams[team]["id Солдат"].append(i)

```

Рисунок 3.4 – Код общего задания

```
(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Задание 1.py"
Уровень героя первой команды: 0
Уровень героя второй команды: 1
Следуют за первым героем: [2]
Следуют за вторым героем: []
```

Рисунок 3.5 – Проверка кода общего задания

3. После чего приступил к выполнению индивидуальных заданий, закрепляя полученные знания.

```
class Liquid:
    def __init__(self, name, density):
        self.__name = name
        self.__density = density

    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, inp):
        self.__name = inp

    @property
    def density(self):
        return self.__density

    @density.setter
    def density(self, inp):
        self.__density = inp

class Alcohol(Liquid):
    def __init__(self, name, density, strength):
        super().__init__(name, density)
        self.__strength = strength

    @property
    def strength(self):
        return self.__strength

    @strength.setter
    def strength(self, inp):
        self.__strength = inp
```

Рисунок 3.6 – Код первого задания

```
(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Задание 2.py"
Водка 949 30
Вода 1000
Водка 949 60
```

Рисунок 3.7 – Проверка кода первого задания

```

class Body(ABC):

    @abstractmethod
    def square(self):
        pass

    @abstractmethod
    def volume(self):
        pass

class Ball(Body):
    def square(self, R):
        print("Площадь поверхности шара: ", 4*pi*R**2)

    def volume(self, R):
        print("Объём шара: ", 4/3*pi*R**3)

class Parallelepiped(Body):
    def square(self, a, b, c):
        print("Площадь поверхности параллелепипеда: ", 2*(a*b + b*c + a*c))

    def volume(self, a, b, c):
        print("Объём параллелепипеда: ", a*b*c)

```

Рисунок 3.8 – Код второго задания

```

(base) C:\Users\zligo\Documents\GitHub\OOP-4.3>python "Задание 3.py"
Площадь поверхности шара: 12.566370614359172
Площадь поверхности параллелепипеда: 6
Объём шара: 4.1887902047863905
Объём параллелепипеда: 1

```

Рисунок 3.9 – Результат выполнения кода

Контрольные вопросы

1. Что такое наследование как оно реализовано в языке Python?

Наследование — это возможность расширения (наследования) ранее написанного программного кода класса с целью дополнения, усовершенствования или привязки под новые требования.

Синтаксически создание класса с указанием его родителя выглядит так:

```
class имя_класса(имя_родителя1, [имя_родителя2,..., имя_родителя_n])
```

2. Что такое полиморфизм и как он реализован в языке Python?

Полиморфизм - это способность выполнять действие над объектом независимо от его типа. Это обычно реализуется путем создания базового класса и наличия двух или более подклассов, которые все реализуют методы с одинаковой сигнатурой.

3. Что такое "утиная" типизация в языке программирования Python?

Эта концепция адаптирована из следующего абдуктивного умозаключения:

Если что-то выглядит как утка, плавает как утка и крикает как утка, это наверняка и есть утка.

Концепция утиной типизации в основном принята в языках программирования, поддерживающих динамическую типизацию, таких как Python и JavaScript. Общей особенностью этих языков является возможность объявления переменных без указания их типа.

При использовании пользовательских типов для определённых целей, реализация связанных функций важнее, чем точные типы данных.

Утиная типизация подчёркивает реализацию связанных выполняемых функций, а конкретные типы данных менее важны

4. Каково назначение модуля `abc` языка программирования Python?

Начиная с версии языка 2.6 в стандартную библиотеку включается модуль `abc`, добавляющий в язык абстрактные базовые классы (далее АБК).

АБК позволяют определить класс, указав при этом, какие методы или свойства обязательно переопределить в классах-наследниках.

5. Как сделать некоторый метод класса абстрактным?

Перед методом класса необходимо добавить декоратор модуля `abc`:
`@abstractmethod`.

6. Как сделать некоторое свойство класса абстрактным?

Абстрактные классы включают в себя атрибуты в дополнение к методам, вы можете потребовать атрибуты в конкретных классах, определив их с помощью `@abstractproperty`.

7. Каково назначение функции `isinstance` ?

Функция `isinstance ()` в Python используется для проверки, является ли объект экземпляром указанного класса или нет.

Вывод: в ходе выполнения лабораторной работы были приобретены простейшие навыки по работе с наследованием и абстрактными методами классов в языке программирования Python.