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

Кафедра инфокоммуникаций

Объектно-ориентированное программирование

Отчет по лабораторной работе №4.3

Наследование и полиморфизм в языке Python

Выполнил студент группы

ИВТ-б-о-21-1

Харченко Б.Р. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил доцент
Кафедры инфокоммуникаций, старший преподаватель
Воронкин Р.А.

_____
(подпись)

Ставрополь 2023

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.

Порядок выполнения работы:

Задание.

Разработайте программу по следующему описанию. В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня. В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки. Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, увеличивается уровень. Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random


class Soldier:
    def __init__(self, number, team):
        self.number = number
        self.team = team

    def go_to_hero(self, hero):
        print(f"Солдат {self.number} идет за героем {hero.number}")


class Hero:
    def __init__(self, number):
        self.number = number
        self.level = 1

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


if __name__ == "__main__":
    hero1 = Hero(1)
```

```
hero2 = Hero(2)

soldiers_team1 = []
soldiers_team2 = []

for _ in range(10):
    number = random.randint(1, 100)
    team = random.choice([1, 2])
    soldier = Soldier(number, team)

    if soldier.team == 1:
        soldiers_team1.append(soldier)
    else:
        soldiers_team2.append(soldier)

if len(soldiers_team1) > len(soldiers_team2):
    hero1.increase_level()
else:
    hero2.increase_level()

soldier_to_follow = random.choice(soldiers_team1)
soldier_to_follow.go_to_hero(hero1)

print(f"Идентификационный номер солдата: {soldier_to_follow.number}")
print(f"Идентификационный номер героя: {hero1.number}")
```

Результат работы программы:

```
Солдат 57 идет за героем 1
Идентификационный номер солдата: 57
Идентификационный номер героя: 1
```

Рисунок 1. Результат работы программы

Задание 1.

Создать базовый класс Triad (тройка чисел) с операциями сложения с чис лом, умножения на число, проверки на равенство. Создать производный класс Vector3D, задаваемый тройкой координат. Должны быть реализованы: опера ция сложения векторов, скалярное произведение векторов.

Код программы:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Triad:
    """
    Базовый класс, представляющий тройку чисел.
    """
    def __init__(self, a=0, b=0, c=0):
        self.a = a
        self.b = b
        self.c = c

    def add(self, number):
        self.a += number
        self.b += number
        self.c += number

    def multiply(self, number):
        self.a *= number
        self.b *= number
        self.c *= number

    def equals(self, other):
        return self.a == other.a and self.b == other.b and self.c == other.c

    def display(self):
        print(f"Тройка чисел: {self.a}, {self.b}, {self.c}")
```

```python
class Vector3D(Triad):
    """
    Производный класс для представления трехмерного вектора.
    """
    2 usages (2 dynamic)
    def add_vector(self, other):
        self.a += other.a
        self.b += other.b
        self.c += other.c

    def dot_product(self, other):
        return self.a * other.a + self.b * other.b + self.c * other.c

    def display(self):
        print(f"Вектор: ({self.a}, {self.b}, {self.c})")

if __name__ == '__main__':
    triad1 = Triad( a: 1,  b: 2,  c: 3)
    triad2 = Triad( a: 4,  b: 5,  c: 6)

    triad1.display()
    triad1.add(10)
    triad1.display()
    triad1.multiply(2)
    triad1.display()
    print(f"Равенство triad1 и triad2: {triad1.equals(triad2)}")
```

```
vector1.display()
vector1.add_vector(vector2)
print("После сложения векторов:")
vector1.display()

print(f"Скалярное произведение vector1 и vector2: {vector1.dot_product(vector2)}")
```

Результат работы программы:

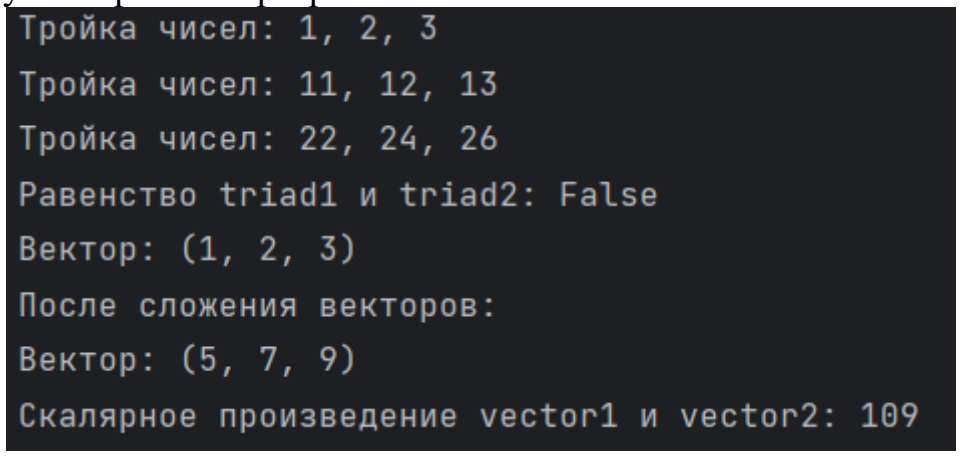

Рисунок 2. Результат работы программы

Задание 2.

В следующих заданиях требуется реализовать абстрактный базовый класс, определив в нем абстрактные методы и свойства. Эти методы определяются в производных классах. В базовых классах должны быть объявлены абстрактные методы ввода/вывода, которые реализуются в производных классах. Вызывающая программа должна продемонстрировать все варианты вызова переопределенных абстрактных методов. Написать

функцию вывода, получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов.

Создать абстрактный базовый класс Number с абстрактными методами — арифметическими операциями. Создать производные классы Integer (целое) и Real(действительное).

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from abc import ABC, abstractmethod


class Number(ABC):
    """
    Абстрактный базовый класс для числа.
    """

    @abstractmethod
    def add(self, other):
        pass

    @abstractmethod
    def subtract(self, other):
        pass

    @abstractmethod
    def multiply(self, other):
        pass

    @abstractmethod
    def divide(self, other):
        pass
```

```python
class Integer(Number):
    """
    Класс для представления целого числа.
    """

    def __init__(self, value):
        self.value = value

    def add(self, other):
        if isinstance(other, (Integer, Real)):
            return Integer(self.value + other.value)

    def subtract(self, other):
        if isinstance(other, (Integer, Real)):
            return Integer(self.value - other.value)

    def multiply(self, other):
        if isinstance(other, (Integer, Real)):
            return Integer(self.value * other.value)

    def divide(self, other):
        if isinstance(other, (Integer, Real)) and other.value != 0:
            return Real(self.value / other.value)
        else:
            raise ValueError("Division by zero is not allowed.")

    def display(self):
        print(f"Целое число: {self.value}")
```

```python
class Real(Number):
    """
    Класс для представления действительного числа.
    """

    def __init__(self, value):
        self.value = value

    def add(self, other):
        if isinstance(other, (Integer, Real)):
            return Real(self.value + other.value)

    def subtract(self, other):
        if isinstance(other, (Integer, Real)):
            return Real(self.value - other.value)

    def multiply(self, other):
        if isinstance(other, (Integer, Real)):
            return Real(self.value * other.value)

    def divide(self, other):
        if isinstance(other, (Integer, Real)) and other.value != 0:
            return Real(self.value / other.value)
        else:
            raise ValueError("Division by zero is not allowed.")

    def display(self):
        print(f"Действительное число: {self.value}")
```

```python
def demonstrate_virtual_call(number):
    """
    Функция для демонстрации работы с числами.
    """
    number.display()


if __name__ == "__main__":
    int_number = Integer(10)
    real_number = Real(5.5)

    result = int_number.add(real_number)
    result.display()

    result = int_number.subtract(real_number)
    result.display()

    result = int_number.multiply(real_number)
    result.display()

    result = int_number.divide(real_number)
    result.display()
```

Результат работы программы:

```
C:\Users\harch\venv\Scripts\python.exe "C:\Users\harch\OneDrive\Рабочий стол\ООП\ООП_ЛР_3\ЛР_3\ind_3.2.py"
Целое число: 15.5
Целое число: 4.5
Целое число: 55.0
Действительное число: 1.8181818181818181
```

Ответы на вопросы:

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

Наследование — это когда один класс (подкласс) получает свойства и методы другого класса (суперкласса). Подкласс может наследовать все публичные атрибуты и методы своего суперкласса и добавлять свои собственные. В языке Python наследование реализуется с помощью ключевого слова class. Для создания подкласса нужно указать имя суперкласса в скобках после имени подкласса. Подкласс получает все атрибуты и методы суперкласса, их можно использовать напрямую или переопределить.

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

Полиморфизм — это возможность объектов разных классов иметь одно и то же имя метода, но каждый класс может предоставить свою собственную реализацию этого метода. Это позволяет использовать одинаковое имя метода для объектов различных классов, что упрощает программирование и повышает гибкость кода. В языке Python полиморфизм реализуется через наследование и переопределение методов. Если в подклассе метод с тем же именем переопределяется, то при вызове этого метода на объекте подкласса будет использоваться его реализация, а не реализация суперкласса. Это

позволяет использовать одинаковые методы с разным поведением для разных классов.

3. Что такое «утиная» типизация в языке Python?

«Утиная» типизация (англ. duck typing) — это концепция в языке программирования Python, основанная на философии «если она выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, и есть утка». В контексте Python утиная типизация означает, что тип объекта определяется по его возможностям и методам, а не по его явно заданному типу. Иными словами, если объект обладает определенными методами, то мы можем использовать его как экземпляр нужного типа, не задумываясь о его фактическом классе или интерфейсе.

4. Каково назначение модуля abc языка Python?

Модуль abc (аббревиатура от "Abstract Base Classes") является частью стандартной библиотеки языка Python и предоставляет средства для определения абстрактных базовых классов.

5. Как сделать некоторый метод класса абстрактным?

Необходимо декорировать его методы как абстрактные, а реализацию выносить в классы-наследники.

6. Как сделать некоторое свойство класса абстрактным?

Можно потребовать атрибут в конкретных классах, определив их с помощью @abstractproperty.

7. Каково назначение функции isinstance?

Функция isinstance() проверяет, является ли объект экземпляром указанного класса или его подкласса.

Вывод: в ходе работы были приобретены навыки по созданию иерархии классов при написании программ с использованием языка программирования Python версии 3.10.