

**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

Мальцев Н.А. « » _____ 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.

Составить программу с использованием иерархии классов. Номер варианта необходимо получить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':`. Создать класс `Triad` (тройка чисел); определить метод сравнения триад (см. задание 2). Определить производный класс `Time` с полями: час, минута и секунда. Определить полный набор методов сравнения моментов времени.

Код программы:

```

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

class Triad:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def compare(self, other):
        # Сравниваем значения полей троек чисел
        if self.a == other.a and self.b == other.b and self.c == other.c:

```

```

        return "Триады равны"
    elif (
        self.a > other.a
        or (self.a == other.a and self.b > other.b)
        or (self.a == other.a and self.b == other.b and self.c > other.c)
    ):
        return "Первая триада больше второй"
    else:
        return "Вторая триада больше первой"

class Time(Triad):
    def __init__(self, hour, minute, second):
        super().__init__(hour, minute, second)

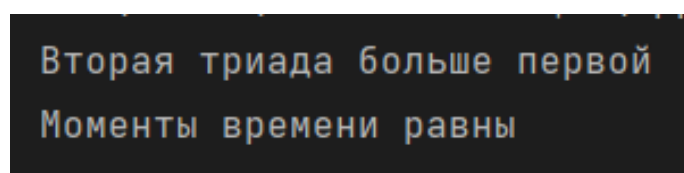
    def compare(self, other):
        # Сравниваем значения полей моментов времени
        if (
            self.a > other.a
            or (self.a == other.a and self.b > other.b)
            or (self.a == other.a and self.b == other.b and self.c > other.c)
        ):
            return "Первый момент времени больше второго"
        elif self.a == other.a and self.b == other.b and self.c == other.c:
            return "Моменты времени равны"
        else:
            return "Второй момент времени больше первого"

if __name__ == "__main__":
    # Создаем объекты класса Triad
    triad1 = Triad(1, 2, 3)
    triad2 = Triad(4, 5, 6)
    print(triad1.compare(triad2)) # Сравниваем тройки чисел

    # Создаем объекты класса Time
    time1 = Time(10, 20, 30)
    time2 = Time(10, 20, 30)
    print(time1.compare(time2)) # Сравниваем моменты времени

```

Результат работы программы:



```

Вторая триада больше первой
Моменты времени равны

```

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 Pair с виртуальными арифметическими операциями. Создать производные классы Money (деньги) и Fraction (дробное число).

Код программы:

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

from abc import ABC, abstractmethod

# Создаем абстрактный класс Pair
class Pair(ABC):
    @abstractmethod
    def __add__(self, other):
        pass

    @abstractmethod
    def __sub__(self, other):
        pass

    @abstractmethod
    def __mul__(self, other):
        pass

    @abstractmethod
    def __truediv__(self, other):
        pass

    @abstractmethod
    def __str__(self):
        pass

# Создаем класс Money, который наследуется от Pair
class Money(Pair):
    def __init__(self, amount):
        self.amount = amount

    # Переопределяем методы сложения, вычитания, умножения, деления и вывода
    # для класса Money
    def __add__(self, other):
        if isinstance(other, Money):
            return Money(self.amount + other.amount)
        else:
            raise TypeError("Unsupported operand type")

    def __sub__(self, other):
        if isinstance(other, Money):
            return Money(self.amount - other.amount)
        else:
            raise TypeError("Unsupported operand type")

    def __mul__(self, other):
        if isinstance(other, (int, float)):
            return Money(self.amount * other)
```

```

        else:
            raise TypeError("Unsupported operand type")

    def __truediv__(self, other):
        if isinstance(other, (int, float)):
            return Money(self.amount / other)
        else:
            raise TypeError("Unsupported operand type")

    def __str__(self):
        return str(self.amount)

# Создаем класс Fraction, который наследуется от Pair
class Fraction(Pair):
    def __init__(self, numerator, denominator):
        self.numerator = numerator
        self.denominator = denominator

    # Переопределяем методы сложения, вычитания, умножения, деления и вывода
    для класса Fraction
    def __add__(self, other):
        if isinstance(other, Fraction):
            common_denominator = self.denominator * other.denominator
            new_numerator = (self.numerator * other.denominator) + (
                other.numerator * self.denominator
            )
            return Fraction(new_numerator, common_denominator)
        else:
            raise TypeError("Unsupported operand type")

    def __sub__(self, other):
        if isinstance(other, Fraction):
            common_denominator = self.denominator * other.denominator
            new_numerator = (self.numerator * other.denominator) - (
                other.numerator * self.denominator
            )
            return Fraction(new_numerator, common_denominator)
        else:
            raise TypeError("Unsupported operand type")

    def __mul__(self, other):
        if isinstance(other, (int, float)):
            return Fraction(self.numerator * other, self.denominator)
        else:
            raise TypeError("Unsupported operand type")

    def __truediv__(self, other):
        if isinstance(other, (int, float)):
            return Fraction(self.numerator, self.denominator * other)
        else:
            raise TypeError("Unsupported operand type")

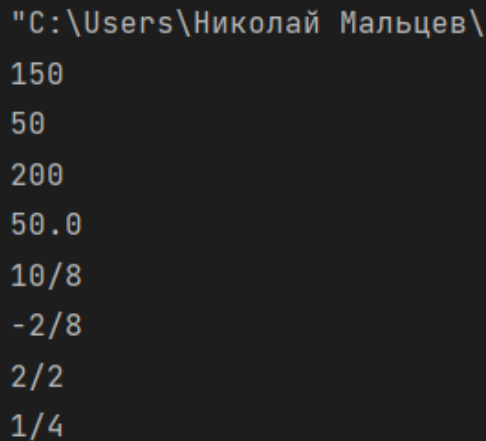
    def __str__(self):
        return f"{self.numerator}/{self.denominator}"

# Основной код программы
if __name__ == "__main__":
    money1 = Money(100)
    money2 = Money(50)
    print(money1 + money2)
    print(money1 - money2)
    print(money1 * 2)

```

```
print(money1 / 2)
fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 4)
print(fraction1 + fraction2)
print(fraction1 - fraction2)
print(fraction1 * 2)
print(fraction1 / 2)
```

Результат работы программы:



```
"C:\Users\Николай Мальцев\  
150  
50  
200  
50.0  
10/8  
-2/8  
2/2  
1/4
```

Рисунок 2. Результат работы программы

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 крикает 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.