

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.3
дисциплины «Объектно-ориентированное программирование»
Вариант 2

Выполнил:
Иващенко Олег Андреевич
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и
вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем»

(подпись)

Руководитель практики:
Воронкин Роман Александрович,
доцент департамента цифровых,
робототехнических систем и
электроники

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: «Наследование и полиморфизм в языке Python»

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

Порядок выполнения работы:

Пример 1. Реляционная (несократимая) дробь представляется парой целых чисел (a, b) , где a – числитель, b – знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции:

- сложения add: $(a, b) + (c, d) = (ad + bc, bd)$;
- вычитания sub: $(a, b) - (c, d) = (ad - bc, bd)$;
- умножения mul: $(a, b) \times (c, d) = (ac, bd)$;
- деления div: $(a, b) / (c, d) = (ad, bc)$;
- сравнения equal, greater, less.

Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

Листинг 1 – Код программы примера 1

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

class Rational:

    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)

        if b == 0:
            raise ValueError()

        self.__numerator = abs(a)
        self.__denominator = abs(b)

        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
```

```

def gcd(a, b):
    if a == 0:
        return b
    elif b == 0:
        return a
    elif a >= b:
        return gcd(a % b, b)
    else:
        return gcd(a, b % 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

# Прочитать значение дроби с клавиатуры. Дробь вводится
# как a/b.
def read(self, prompt=None):
    line = input() if prompt is None else input(prompt)
    parts = list(map(int, line.split('/', maxsplit=1)))

    if parts[1] == 0:
        raise ValueError()

    self.__numerator = abs(parts[0])
    self.__denominator = abs(parts[1])

    self.__reduce()

# Вывести дробь на экран
def display(self):
    print(f"{self.__numerator}/{self.__denominator}")

# Сложение обыкновенных дробей.
def add(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator

        return Rational(a, b)
    else:
        raise ValueError()

```

```

# Вычитание обыкновенных дробей.
def sub(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator

        return Rational(a, b)
    else:
        raise ValueError()

# Умножение обыкновенных дробей.
def mul(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError()

# Деление обыкновенных дробей.
def div(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator
        return Rational(a, b)
    else:
        raise ValueError()

# Отношение обыкновенных дробей.
def equals(self, rhs):
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
            (self.denominator == rhs.denominator)
    else:
        return False

def greater(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator
        return v1 > v2
    else:
        return False

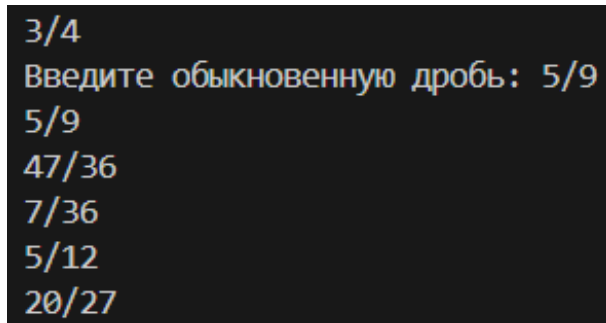
def less(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator
        return v1 < v2
    else:
        return False

```

```

if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()
    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()
    r3 = r2.add(r1)
    r3.display()
    r4 = r2.sub(r1)
    r4.display()
    r5 = r2.mul(r1)
    r5.display()
    r6 = r2.div(r1)
    r6.display()

```



3/4
 Введите обыкновенную дробь: 5/9
 5/9
 47/36
 7/36
 5/12
 20/27

Рисунок 1 – Вывод программы примера 1

Пример 2. Используя абстрактные методы, создать несколько классов, наследуемые от абстрактного.

Листинг 2 – Код программы примера 2

```

# Python program showing
# abstract base class work

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")
```

```
# Driver code
```

```
R = Triangle()
```

```
R.noofsides()
```

```
K = Quadrilateral()
```

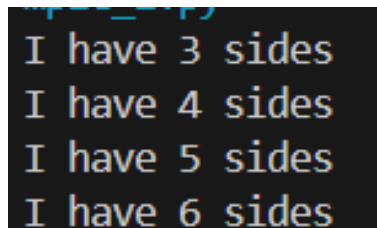
```
K.noofsides()
```

```
R = Pentagon()
```

```
R.noofsides()
```

```
K = Hexagon()
```

```
K.noofsides()
```



```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Рисунок 2 – Результат выполнения программы примера 2

Пример 3.

Листинг 3 – Код программы примера 3

```
# Python program showing
```

```
# abstract base class work

from abc import ABC

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")

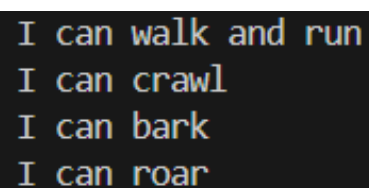
class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()
K = Snake()
K.move()
R = Dog()
R.move()
K = Lion()
K.move()
```



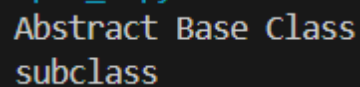
```
I can walk and run
I can crawl
I can bark
I can roar
```

Рисунок 3 – Результат выполнения программы примера 3

Пример 4. Рассмотрим пример вызова метода, используя super().

Листинг 4 – Код программы примера 4

```
# Python program invoking a  
# method using super()  
  
from abc import ABC  
  
class R(ABC):  
  
    def rk(self):  
        print("Abstract Base Class")  
  
class K(R):  
  
    def rk(self):  
        super().rk()  
        print("subclass")  
  
# Driver code  
r = K()  
r.rk()
```



```
Abstract Base Class  
subclass
```

Рисунок 4 – Результат выполнения программы примера 4

Общее задание. 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 создаётся 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.

Листинг 5 – Код программы общего задания

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

import abc
import random

class Unit:

    @abc.abstractmethod
    def __init__(self, id, command_id):
        """Метод инициализации"""
        pass

    @abc.abstractmethod
    def action(self, *args):
        """Метод действия"""
        pass

class Solder(Unit):

    def __init__(self, id, command_id):
        """Метод инициализации солдата"""
        self.id = id
        self.command_id = command_id

    def action(self, hero):
        """Метод действия солдата"""
        if isinstance(hero, Hero):
            print("Солдат: Следую за Героем!")
        else:
            TypeError

class Hero(Unit):

    def __init__(self, id, command_id):
        """Метод инициализации героя"""
        self.id = id
```

```

        self.command_id = command_id
        self.level = 1

    def action(self):
        """Метод действия героя"""
        self.level += 1
        print(f"Уровень Героя {self.id} повышен до уровня {self.level}!")

if __name__ == '__main__':
    """Основной метод"""
    hero1 = Hero(1, 0) # Создание героя 1
    hero2 = Hero(2, 1) # Создание героя 2

    # Создание список для распределения солдат
    firstTeamUnitList = []
    secondTeamUnitList = []

    # Цикл для распределения солдат по командам
    for i in range(30):
        team = random.randint(0, 1)
        if team == 0:
            firstTeamUnitList.append(Solder(i + 3, 0))
        else:
            secondTeamUnitList.append(Solder(i + 3, 1))

    firstCount = firstTeamUnitList.__len__()
    secondCount = secondTeamUnitList.__len__()

    # Сравнение количества солдат в каждой команде
    if firstCount > secondCount:
        hero1.action()
    elif secondCount > firstCount:
        hero2.action()
    else:
        print("Результат битвы: Ничья")

    # Вывод детальной информации по командам
    print("-----" * 4, "[Команда 1]", "-----" * 4)
    print(f"Уровень героя: {hero1.level}")
    print(f"Количество юнитов в команде: {firstTeamUnitList.__len__()}\\n")
    print("-----" * 4, "[Команда 2]", "-----" * 4)
    print(f"Уровень героя: {hero2.level}")
    print(f"Количество юнитов в команде: {secondTeamUnitList.__len__()}\\n")

    randomSolder = firstTeamUnitList[random.randint(0, firstCount - 1)]
    randomSolder.action(hero1)
    print(f"ID Героя 1: {hero1.id}")
    print("ID случайного солдата Героя 1: "
          f"{randomSolder.id}")

```

```

Уровень Героя 2 повышен до уровня 2!
----- [Команда 1] -----
Уровень героя: 1
Количество юнитов в команде: 11

----- [Команда 2] -----
Уровень героя: 2
Количество юнитов в команде: 19

Солдат: Следую за Героем!
ID Героя 1: 1
ID случайного солдата Героя 1: 8

```

Рисунок 5 – Результат выполнения программы общего задания

Индивидуальное задание 1. Создать класс Pair (пара чисел); определить методы изменения полей и вычисления произведения чисел. Определить производный класс RightAngled с полями-катетами. Определить методы вычисления гипотенузы и площади треугольника.

Листинг 6 – Код программы индивидуального задания 1

```

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

import math

class Pair:

    def __init__(self, x, y):
        """Инициализация объекта с указанием двух значений переменных"""
        self.x = x
        self.y = y

    def set_values(self, x, y):
        """Установка значений переменных x и y отдельным методом"""
        self.x = x
        self.y = y

    def mul(self):
        """Возвращает произведение значений переменных x и y"""
        return self.x * self.y

class RightAngled(Pair):

```

```

def __init__(self, cath1, cath2):
    """Инициализация с наследованием от класса Pair"""
    super().__init__(cath1, cath2)

def hypotenuse(self):
    """Вычисление гипотенузы по заданным катетам
    Значения x и y не объявлялись, т.к. наследованы от класса Pair"""
    return math.sqrt(self.x ** 2 + self.y ** 2)

def square(self):
    """Вычисление площади треугольника по катетам
    Значения x и y не объявлялись, т.к. наследованы от класса Pair"""
    return (self.x * self.y) / 2

if __name__ == "__main__":
    # Основной метод программы
    pair = Pair(0, 0) # Создание нового объекта класса Pair
    pair.set_values(5, 10) # Отдельная установка значений x и y
    # Вывод значений x и y, вычисление произведения
    print(f"Произведения чисел {pair.x} и {pair.y} равно {pair.mul()}")

    triangle = RightAngled(4, 12) # Создание нового объекта класса RightAngled
    # Определение гипотенузы треугольника с помощью метода hypotenuse по
    # установленным в предыдущей строке значениям, вывод этих значений и
    # результата расчёта.
    print(f"\nГипотенуза треугольника с катетами {triangle.x} и"
          f" {triangle.y} равна {round(triangle.hypotenuse(), 4)}")
    # Определение площади треугольника с помощью метода square
    print(f"Площадь треугольника равна {triangle.square()}")

```

```

Произведения чисел 5 и 10 равно 50
Гипотенуза треугольника с катетами 4 и 12 равна 12.6491
Площадь треугольника равна 24.0

```

Рисунок 6 – Результат выполнения программы индивидуального задания 1

Индивидуальное задание 2. В следующих задачах требуется реализовать абстрактный базовый класс, определив в нём 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 переопределённых абстрактных методов. Написать функцию вывода,

получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов.

Создать абстрактный базовый класс `Pair` с виртуальными арифметическими операциями. Реализовать производные классы `Complex` (комплексной числа) и `Rational` (рациональное число).

Листинг 7 – Код программы индивидуального задания 2

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

import abc

class Pair:

    @abc.abstractmethod
    def __init__(self, x, y):
        """Метод инициализации"""
        pass

    @abc.abstractmethod
    def set_values(self, x, y):
        """Метод смены значений"""
        pass

    @abc.abstractmethod
    def sum(self, x, y):
        """Метод суммирования"""
        pass

    @abc.abstractmethod
    def subtract(self, x, y):
        """Метод вычитания"""
        pass

    @abc.abstractmethod
    def divide(self, x, y):
        """Метод деления"""
        pass

    @abc.abstractmethod
    def multiply(self):
        """Метод произведения"""
        pass

    @abc.abstractmethod
    def display(self):
        """Метод вывода значений"""
        pass
```

```
class Complex(Pair):
```

```
    def __init__(self, real, imagine):
```

```
        """Метод инициализации комплексных чисел"""
```

```
        self.real = real
```

```
        self.imagine = imagine
```

```
    def sum(self, other):
```

```
        """Метод суммирования комплексных чисел"""
```

```
        return Complex(self.real + other.real, self.imagine + other.imagine)
```

```
    def subtract(self, other):
```

```
        """Метод вычитания комплексных чисел"""
```

```
        return Complex(self.real - other.real, self.imagine - other.imagine)
```

```
    def multiply(self, other):
```

```
        """Метод произведения комплексных чисел"""
```

```
        z1 = self.real * other.real - self.imagine * other.imagine
```

```
        z2 = self.real * other.imagine + self.imagine * other.real
```

```
        return Complex(z1, z2)
```

```
    def divide(self, other):
```

```
        """Метод деления комплексных чисел"""
```

```
        z1 = self.real * other.real + self.imagine * other.imagine
```

```
        z2 = other.real ** 2 + other.imagine ** 2
```

```
        z3 = other.real * self.imagine - self.real * other.imagine
```

```
        return Complex(round(z1 / z2, 4), round(z3 / z2, 4))
```

```
    def display(self):
```

```
        """Метод вывода комплексных чисел"""
```

```
        return f'{self.real} + {self.imagine}i'
```

```
class Rational:
```

```
    def __init__(self, x, y):
```

```
        """Метод инициализации рационального числа"""
```

```
        self.x = x
```

```
        self.y = y
```

```
    def sum(self, other):
```

```
        """Метод суммирования рациональных чисел"""
```

```
        if isinstance(other, Rational):
```

```
            return Rational(self.x * other.y + other.x * self.y,
```

```
                            self.y * other.y)
```

```
    def subtract(self, other):
```

```
        """Метод вычитания рациональных чисел"""
```

```
        if isinstance(other, Rational):
```

```
            return Rational(self.x * other.y - other.x * self.y,
```

```

        self.y * other.y)

def multiply(self, other):
    """Метод произведения рациональных чисел"""
    if isinstance(other, Rational):
        return Rational(self.x * other.x, self.y * other.y)

def divide(self, other):
    """Метод деления рациональных чисел"""
    if isinstance(other, Rational):
        return Rational(self.x * other.y, other.x * self.y)

def display(self):
    """Метод вывода рационального числа"""
    return f'{self.x}/{self.y}'

if __name__ == "__main__":
    """Основной методу"""

    print('-----[ Комплексные числа ]-----')
    c1 = Complex(5, 3)
    c2 = Complex(10, 2)
    print(f'Первое комплексное число: {c1.display()}')
    print(f'Второе комплексное число: {c2.display()}')
    print(f'z = ({c1.display()}) + ({c2.display()}) = {c1.sum(c2).display()}')
    print(f'z = ({c1.display()}) - ({c2.display()}) ')
    f = {c1.subtract(c2).display()}
    print(f'z = ({c1.display()}) × ({c2.display()}) ')
    f = {c1.multiply(c2).display()}
    print(f'z = ({c1.display()}) ÷ ({c2.display()}) ')
    f = {c1.divide(c2).display()}

    print("\n-----[ Рациональные числа ]-----")
    r1 = Rational(4, 5)
    r2 = Rational(6, 9)
    print(f'{r1.display()} + {r2.display()} = {r1.sum(r2).display()}')
    print(f'{r1.display()} - {r2.display()} = {r1.subtract(r2).display()}')
    print(f'{r1.display()} × {r2.display()} = {r1.multiply(r2).display()}')
    print(f'{r1.display()} ÷ {r2.display()} = {r1.divide(r2).display()}')

```

```

-----[ Комплексные числа ]-----
Первое комплексное число: 5 + 3i
Второе комплексное число: 10 + 2i
z = (5 + 3i) + (10 + 2i) = 15 + 5i
z = (5 + 3i) - (10 + 2i) = -5 + 1i
z = (5 + 3i) x (10 + 2i) = 45 + 40i
z = (5 + 3i) ÷ (10 + 2i) = 0.5385 + 0.1923i

-----[ Рациональные числа ]-----
4/5 + 6/9 = 66/45
4/5 - 6/9 = 6/45
4/5 x 6/9 = 24/45
4/5 ÷ 6/9 = 36/30

```

Рисунок 7 – Вывод программы индивидуального задания 2

Ответы на контрольные вопросы:

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

В организации наследования участвуют как минимум два класса: класс родитель и класс потомок. При этом возможно множественное наследование, в этом случае у класса потомка может быть несколько родителей. Не все языки программирования поддерживают множественное наследование, но в Python можно его использовать. По умолчанию все классы в Python являются наследниками от object, явно этот факт указывать не надо.

Синтаксически создание класса с указанием его родителя выглядит так:

```

class <имя_класса>(имя_родителя, [имя_родителя_2, ...,
имя_родителя_n])

```

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

Полиморфизм, как правило, используется с позиции переопределения методов базового класса в классе наследнике. Т.е. метод, указанный в родительском классе, может иметь такое же имя, но другой функционал в классе-наследнике.

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

Утиная типизация – это концепция, характерная для языков программирования с динамической типизацией, согласно которой конкретный тип или класс объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает. Другими словами, при работе с объектом его тип не проверяется, вместо этого проверяются свойства и методы этого объекта. Такой подход добавляет гибкости коду, позволяет полиморфно работать с объектами, которые никак не связаны друг с другом и могут быть объектами разных классов. Единственное условие, чтобы все эти объекты поддерживали необходимый набор свойств и методов.

Пример:

```
>>> class Meter:
...     def __len__(self):
...         return 1_000
>>> len([1, 2, 3])
3
>>> len("Duck typing...")
14
>>> len(Meter())
1000
```

В примере функции len не важен тип аргумента, а важно лишь то, что у объекта можно вызвать метод `__len__()`.

4. Каково назначение модуля abc языка программирования Python?

По умолчанию Python не предоставляет абстрактных классов. Python поставляется с модулем, который обеспечивает основу для определения абстрактных базовых классов (ABC), и имя этого модуля – ABC. ABC работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы.

5. Как сделать некоторый метод класса абстрактным?

Метод становится абстрактным, если он украшен ключевым словом `@abstractmethod`.

6. Как сделать некоторое свойство класса абстрактным?

Абстрактные свойства включают в себя атрибуты в дополнение к методам. Определяются с помощью декоратора `@abstractproperty`.

7. Каково назначение функции `isinstance`?

Функция `isinstance` нужна для сравнения объекта с классом. Пример:

```
isinstance(<объект>, <класс>)
```

```
isinstance(5, int) # true
```

```
isinstance('Hello, world!', int) # false
```

Выводы: В процессе выполнения лабораторной работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x, проработаны пример, выполнено общее задание и 2 индивидуальных задания.