

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет о лабораторной работе №4.3 по дисциплине основы программной
инженерии**

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

Выполнение:

```
#!/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()
```

Пример 1

```
3/4
Введите обыкновенную дробь: 2/121
2/121
371/484
355/484
3/242
8/363
```

Результат работы примера 1

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

# 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
if __name__ == '__main__':
```

```
R = Triangle()
R.noofsides()
K = Quadrilateral()
K.noofsides()
R = Pentagon()
R.noofsides()
K = Hexagon()
K.noofsides()
```

Пример 2

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

Результат работы примера 2

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

# 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")

if __name__ == '__main__':
    # Driver code
    R = Human()
    R.move()
    K = Snake()
    K.move()
    R = Dog()
    R.move()
```

```
K = Lion()
K.move()
```

Пример 3

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

Результат работы примера 3

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

from random import randint

class Unit:
    __number = 0

    def __init__(self, team):
        self.__identifier = self.__number
        self.team = team
        self.__class__.__number += 1

    @property
    def identifier(self):
        return self.__identifier

    @property
    def team(self):
        return self.__team

    @team.setter
    def team(self, value):
        self.__team = value

class Hero(Unit):

    def __init__(self, team):
        super().__init__(team)
        self.level = 0

    @property
    def level(self):
        return self.__level

    @level.setter
    def level(self, value):
        self.__level = value

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

    def display(self):
        print(f"Команда: {self.team}")
        print(f"ID: {self.identifier}")
        print(f"Уровень: {self.level}")
```

```

class Solider(Unit):

    def __init__(self, team):
        super().__init__(team)
        self.follows = False
        self.hero_id = -1

    def start_following(self, hero):
        if isinstance(hero, Hero):
            self.follows = True
            self.hero_id = hero.identifier
        else:
            raise ValueError

    @property
    def follows(self):
        return self.__follows

    @property
    def hero_id(self):
        return self.__hero_id

    @follows.setter
    def follows(self, value):
        self.__follows = value

    @hero_id.setter
    def hero_id(self, value):
        self.__hero_id = value

    def display(self):
        print(f"Команда: {self.team}")
        print(f"ID: {self.identifier}")
        print(f"Следует за героем: {self.follows}")
        if self.follows:
            print(f"ID героя: {self.hero_id}")

if __name__ == '__main__':
    h1 = Hero(1)
    h2 = Hero(2)
    team1 = []
    team2 = []
    for i in range(10):
        probability = randint(1, 2)
        if probability == 1:
            team1.append(Solider(1))
        else:
            team2.append(Solider(2))
    if len(team1) > len(team2):
        h1.increment_level()
    else:
        h2.increment_level()
    rand_unit = randint(0, len(team1) - 1)
    team1[rand_unit].start_following(h1)
    print("Герой №1:")
    h1.display()
    print("\nГерой №2:")
    h2.display()
    print("\nСолдат команды №2:")
    team2[0].display()

```

```
print("\nСолдат, следующий за героем №1:")
team1[rand_unit].display()
```

Решение первой индивидуальной задачи

```
Герой №1:
Команда: 1
ID: 0
Уровень: 0

Герой №2:
Команда: 2
ID: 1
Уровень: 1

Солдат команды №2:
Команда: 2
ID: 3
Следует за героем: False

Солдат, следующий за героем №1:
Команда: 1
ID: 0
Следует за героем: True
ID героя: 0
```

Результата работы программы

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

class Man:

    def __init__(self, name, age, sex, weight):
        self.name = name
        self.age = age
        self.sex = sex
        self.weight = weight

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

    @property
    def age(self):
        return self.__age

    @property
    def sex(self):
        return self.__sex

    @property
```



```

def weight(self):
    return self.__weight

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

@age.setter
def age(self, value):
    self.__age = value

@sex.setter
def sex(self, value):
    self.__sex = value

@weight.setter
def weight(self, value):
    self.__weight = value

def set_name(self, name):
    self.name = name

def set_weight(self, weight):
    self.weight = weight

def set_age(self, age):
    self.age = age

def display(self):
    print()
    print(f"Имя: {self.name}")
    print(f"Пол: {self.sex}")
    print(f"Возраст: {self.age}")
    print(f"Вес: {self.weight}")

class Student(Man):

    def __init__(self, name, age, sex, weight, grade):
        super().__init__(name, age, sex, weight)
        self.grade = grade

    @property
    def grade(self):
        return self.__grade

    @grade.setter
    def grade(self, value):
        self.__grade = value

    def set_grade(self, value):
        self.grade = value

    def increase_grade(self):
        self.grade += 1

    def display(self):
        super(Student, self).display()
        print(f"Год обучения: {self.grade}")

if __name__ == "__main__":
    m1 = Man(
        name="Ibragim",

```

```

        age=16,
        sex='Male',
        weight=62
    )
    m1.display()
    m1.set_age(17)
    m1.display()

    m2 = Student(
        name="Vladimir",
        age=18,
        sex="Male",
        weight=75,
        grade=2
    )
    m2.display()
    m2.increase_grade()
    m2.set_name("Dima")
    m2.set_age(19)
    m2.display()

```

Решение второй индивидуальной задачи

```

Имя: Ibragim
Пол: Male
Возраст: 16
Вес: 62

Имя: Ibragim
Пол: Male
Возраст: 17
Вес: 62

Имя: Vladimir
Пол: Male
Возраст: 18
Вес: 75
Год обучения: 2

Имя: Dima
Пол: Male
Возраст: 19
Вес: 75
Год обучения: 3

```

Результат работы программы

```

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

from math import sqrt
from abc import ABC, abstractmethod

```

```

class Function(ABC):
    @abstractmethod
    def __init__(self):
        pass

    @abstractmethod
    def calculate(self, x):
        pass

    def display(self, x):
        print(f"Вычисленное значение: {self.calculate(x)}")

class Ellipse(Function):

    def __init__(self, a, b):
        if a == 0 or b == 0:
            raise ValueError
        if a < b:
            raise ValueError
        self.__a = a
        self.__b = b

    def calculate(self, x):
        temp = (x ** 2) / (self.__a ** 2)
        y = sqrt((1 - temp) * (self.__b ** 2))
        return y

class Hyperbola(Function):

    def __init__(self, a, b):
        if a == 0 or b == 0:
            raise ValueError
        self.__a = a
        self.__b = b

    def calculate(self, x):
        temp = (x ** 2) / (self.__a ** 2)
        y = sqrt((1 + temp) * (self.__b ** 2))
        return y

if __name__ == '__main__':
    el = Ellipse(4, 2)
    print("Эллипс")
    print("Значение в точке x = 3: ")
    el.display(3)
    print()
    print("Гипербола")
    print("Значение в точке x = 2.5: ")
    hy = Hyperbola(1, 2)
    hy.display(2.5)

```

Решение третьей индивидуальной задачи

```
Эллипс
Значение в точке x = 3:
Вычисленное значение: 1.3228756555322954

Гипербола
Значение в точке x = 2.5:
Вычисленное значение: 5.385164807134504
```

Результат работы программы

Ответы на 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.

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, как правило, используется с позиции переопределения методов базового класса в классе наследнике.

Таким образом, класс наследник может расширять функционал класса родителя.

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

Утиная типизация заключается в том, что вместо проверки типа чего-либо в Python мы склонны проверять, какое поведение оно поддерживает, зачастую пытаюсь использовать это поведение и перехватывая исключение, если оно не работает.

4. Каково назначение модуля abc языка программирования Python?

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?

@abstractmethod.

6. Как сделать некоторое свойство класса абстрактным?

@abstractproperty

7. Каково назначение функции isinstance ?

Встроенная функция `isinstance(obj, Cls)`, используемая при реализации методов арифметических операций и операций отношения, позволяет узнать что некоторый объект `obj` является либо экземпляром класса `Cls`.