

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

**Отчет по лабораторной работе № 16  
«Наследование и полиморфизм в языке Python»**

**по дисциплине «Основы программной инженерии»**

Выполнила:

Первых Дарья Александровна,  
2 курс, группа ПИЖ-б-о-20-1

Проверил:

Доцент кафедры  
инфокоммуникаций, Воронкин Р.А.

Ставрополь, 2022 г.

## ВЫПОЛНЕНИЕ

```
class Figure:
    def __init__(self, color):
        self.__color = color

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, c):
        self.__color = c

class Rectangle(Figure):
```

main x

C:\Users\podar\study\anaconda\envs\16LR\  
10 20 green  
red

Рисунок 1 – Полиморфизм

```
def info(self):
    print("Rectangle")
    print("Color: " + self.color)
    print("Width: " + str(self.width))
    print("Height: " + str(self.height))
    print("Area: " + str(self.area()))

if __name__ == '__main__':
    fig = Figure("orange")
    fig.info()
    rect = Rectangle(10, 20, "green")
    rect.info()
```

if \_\_name\_\_ == '\_\_main\_\_'

main ×

C:\Users\podar\study\anaconda\envs\16LR\python.exe  
Figure  
Color: orange  
Rectangle  
Color: green  
Width: 10  
Height: 20  
Area: 200

Рисунок 2 – Полиморфизм

```
self.width = w
self.height = h

class DeskTable(Table):
    def square(self):
        return self.width * self.length

if __name__ == '__main__':
    t1 = Table(1.5, 1.8, 0.75)
    t2 = DeskTable(0.8, 0.6, 0.7)
    print(t2.square())

if __name__ == '__main__':
    main
```

C:\Users\podar\study\anaconda\envs\16LR\python.  
0.48

Рисунок 3 – Простое наследование методов родительского класса

```
class Table:
    def __init__(self, l, w, h):
        self.length = l
        self.width = w
        self.height = h

class KitchenTable(Table):
    def __init__(self, l, w, h, p):
        Table.__init__(self, l, w, h)
        self.places = p

if __name__ == '__main__':
```

Рисунок 4 – Полное переопределение метода надклассса

```
class parent:
    def geeks(self):
        pass

class child(parent):
    def geeks(self):
        print("child class")

if __name__ == '__main__':
    print(issubclass(child, parent))
    print(isinstance(child(), parent))
```

main x

C:\Users\podar\study\anaconda\envs\16LR\py  
True  
True

Рисунок 5 – Дополнение, оно же расширение, метода

```
10 class R(ABC):
11     def rk(self):
12         print("Abstract Base Class")
13
14
15 class K(R):
16     def rk(self):
17         super().rk()
18         print("subclass")
19
20
21 if __name__ == '__main__':
22     r = K()
23     r.rk()
```

K > rk()

main ×

C:\Users\podar\study\anaconda\envs\16LR\python

Abstract Base Class

subclass

Рисунок 6 – Параметры со значениями по умолчанию у родительского класса

```
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):
            while b:
                a, b = b, a % b
            return a
        g = gcd(self.__numerator, self.__denominator)
        self.__numerator //= g
        self.__denominator //= g

    def __str__(self):
        return f'{self.__numerator}/{self.__denominator}'

if __name__ == '__main__':
    r = Rational(3, 4)
    print(r)
    r = Rational(8, 10)
    print(r)
    r = Rational(31, 20)
    print(r)
    r = Rational(1, 20)
    print(r)
    r = Rational(3, 5)
    print(r)
    r = Rational(16, 15)
    print(r)
```

Рисунок 7 – Параметры со значениями по умолчанию у родительского класса

```
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")  
  
main ×  
C:\Users\podar\study\anaconda\envs\16LR\py  
I have 3 sides  
I have 4 sides  
I have 5 sides  
I have 6 sides
```

Рисунок 8 – Абстрактные классы в Python



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

main ×

C:\Users\podar\study\anaconda\en  
I can walk and run  
I can crawl  
I can bark  
I can roar

Рисунок 9 – Абстрактные классы в Python

## Индивидуальное задание №1

Составить программу с использованием иерархии классов. Номер варианта необходимо получить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанных классов.

Создать базовый класс `Car` (машина), характеризующийся торговой маркой (строка), числом цилиндров, мощностью. Определить методы переназначения и изменения мощности. Создать производный класс `Lorry` (грузовик), характеризующийся также грузоподъемностью кузова. Определить функции переназначения марки и изменения грузоподъемности.



```
def capacity(self):
    return self.__capacity

@capacity.setter
def capacity(self, inp):
    self.__capacity = inp

def main():
    truck = Lorry("Газель", 4, 119, 2500)
    passenger = Car("Лада", 8, 105)
    print(f"Марка {truck.mark}, количество цилиндров: {truck.cylinders}, "
          f"мощность: {truck.power} л.с, грузоподъемность: {truck.capacity} т")
    print(f"Марка {passenger.mark}, количество цилиндров: {passenger.cylinders}")

if __name__ == "__main__":
    main()
```

main x

C:\Users\podar\study\anaconda\envs\16LR\python.exe "C:/Users/podar/study/PyCharm Com  
Марка Газель, количество цилиндров: 4, мощность: 119 л.с, грузоподъемность: 2500 т  
Марка Лада, количество цилиндров: 8

Рисунок 10 – Индивидуальное задание №1

## Индивидуальное задание №2

Создать абстрактный базовый класс `Figure` с абстрактными методами вычисления площади и периметра. Создать производные классы: `Rectangle` (прямоугольник), `Circle` (круг), `Trapezium` (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие — в производных.

```
class Figure(ABC):
    @abstractmethod
    def square(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(Figure):
    def square(self, a, b):
        print("Площадь прямоугольника: ", a * b)
```

Rectangle > square()

main ×

C:\Users\podar\study\anaconda\envs\16LR\python.exe "C  
Площадь прямоугольника: 80  
Периметр прямоугольника: 18  
Площадь треугольника: 9.0  
Периметр треугольника: 18  
Площадь трапеции: 16.0  
Периметр трапеции: 19

Рисунок 11 – Индивидуальное задание №2

## ВОПРОСЫ

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

Синтаксически создание класса с указанием его родителя выглядит так:

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

super – это ключевое слово, которое используется для обращения к родительскому классу.

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

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

### 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 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` ?

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` либо экземпляром одного из потомков класса `Cls`.