

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

Институт перспективной инженерии

**ОТЧЕТ  
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3  
дисциплины «Объектно-ориентированное программирование»  
Вариант 7**

Выполнил:  
Мотовилов Вадим Борисович  
3 курс, группа ИВТ-б-о-23-2,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А. ктн доцент  
департамента перспективной  
инженерии

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты\_\_\_\_\_

Ставрополь, 2026 г.

**Тема:** Наследование и полиморфизм в языке Python.

**Цель работы:** приобретение навыков по иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

**Порядок выполнения работы:**

**Задание 1.**

7. Создать класс Triangle с полями-сторонами. Определить методы изменения сторон, вычисления углов, вычисления периметра. Создать производный класс Equilateral (равносторонний), имеющий поле площади. Определить метод вычисления площади.

Код программы:

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

import math

class Triangle:
    def __init__(self, a=3, b=4, c=5):
        if not self._is_valid_triangle(a, b, c):
            print("Ошибка: такие стороны не могут образовать треугольник")
            a, b, c = 3, 4, 5
        self.a = a
        self.b = b
        self.c = c

    def _is_valid_triangle(self, a, b, c):
        return (a > 0 and b > 0 and c > 0 and
                a + b > c and
                a + c > b and
                b + c > a)

    def read(self):
        while True:
            try:
                a = float(input("Введите сторону a: "))
                b = float(input("Введите сторону b: "))
                c = float(input("Введите сторону c: "))

                if self._is_valid_triangle(a, b, c):
                    self.a = a
                    self.b = b
                    self.c = c
                    break
            except ValueError:
                print("Ошибка: введите числа")

    def display(self):
```

```

print(f"Треугольник со сторонами: a={self.a:.2f}, b={self.b:.2f}, c={self.c:.2f}")
print(f"Периметр: {self.perimeter():.2f}")
angles = self.calculate_angles()
print(f"Углы: A={angles[0]:.1f}°, B={angles[1]:.1f}°, C={angles[2]:.1f}°")

def set_sides(self, a=None, b=None, c=None):
    new_a = a if a is not None else self.a
    new_b = b if b is not None else self.b
    new_c = c if c is not None else self.c

    if self._is_valid_triangle(new_a, new_b, new_c):
        self.a = new_a
        self.b = new_b
        self.c = new_c
    else:
        print("Ошибка: такие стороны не могут образовать треугольник")

def calculate_angles(self):
    try:
        # Угол A (противолежащий стороне a)
        cosA = (self.b**2 + self.c**2 - self.a**2) / (2 * self.b * self.c)
        angleA = math.degrees(math.acos(cosA))

        # Угол B (противолежащий стороне b)
        cosB = (self.a**2 + self.c**2 - self.b**2) / (2 * self.a * self.c)
        angleB = math.degrees(math.acos(cosB))

        # Угол C (противолежащий стороне c)
        cosC = (self.a**2 + self.b**2 - self.c**2) / (2 * self.a * self.b)
        angleC = math.degrees(math.acos(cosC))

        return angleA, angleB, angleC
    except ValueError:
        return 60.0, 60.0, 60.0

def perimeter(self):
    return self.a + self.b + self.c

class Equilateral(Triangle):
    def __init__(self, side=1):
        super().__init__(side, side, side)
        self._area = None

    def read(self):
        while True:
            try:
                side = float(input("Введите сторону равностороннего треугольника: "))
                if side > 0:
                    self.set_sides(side, side, side)
                    self._area = None
                    break
                else:

```

```

        print("Ошибка: сторона должна быть положительной")
    except ValueError:
        print("Ошибка: введите число")

def display(self):
    print(f"Равносторонний треугольник со стороной: {self.a:.2f}")
    print(f"Периметр: {self.perimeter():.2f}")
    print(f"Все углы: 60.0°")
    print(f"Площадь: {self.calculate_area():.2f}")

def set_sides(self, a=None, b=None, c=None):
    if a is not None:
        if a > 0:
            self.a = a
            self.b = a
            self.c = a
            self._area = None
        else:
            print("Ошибка: сторона должна быть положительной")
    else:
        super().set_sides(a, b, c)
        if not self._is_equilateral():
            print("Внимание: треугольник больше не равносторонний")

def _is_equilateral(self):
    return math.isclose(self.a, self.b) and math.isclose(self.b, self.c)

def calculate_area(self):
    if self._area is None or not self._is_equilateral():
        self._area = (math.sqrt(3) / 4) * self.a ** 2
    return self._area

def get_area(self):
    return self.calculate_area()

if __name__ == '__main__':
    print("1. Работа с обычным треугольником:")
    t1 = Triangle(3, 4, 5)
    t1.display()

    print("\n Изменяем стороны на 5, 6, 7:")
    t1.set_sides(5, 6, 7)
    t1.display()

    print("\n2. Создание равностороннего треугольника:")
    eq1 = Equilateral(3)
    eq1.display()

    print("\n Изменяем сторону на 5:")
    eq1.set_sides(5)
    eq1.display()

```

```

print("\n3. Проверка наследования:")
print(f"  Периметр равностороннего треугольника: {eq1.perimeter():.2f}")
print(f"  Углы равностороннего треугольника: {eq1.calculate_angles()}\n")

print("\n4. Ввод обычного треугольника с клавиатуры:")
t2 = Triangle()
t2.read()
t2.display()

print("\n5. Ввод равностороннего треугольника с клавиатуры:")
eq2 = Equilateral()
eq2.read()
eq2.display()

print("\n6. Демонстрация полиморфизма (список треугольников):")
shapes = [
    Triangle(3, 4, 5),
    Equilateral(4),
    Triangle(5, 5, 8),
    Equilateral(6)
]

for i, shape in enumerate(shapes, 1):
    print(f"\n  Фигура {i}:")
    shape.display()
    if isinstance(shape, Equilateral):
        print(f"  Площадь (специальный метод): {shape.get_area():.2f}\n")

print("\n7. Проверка обработки ошибок:")

print("  Попытка создать недопустимый треугольник:")
t3 = Triangle(1, 2, 10)

print("\n  Попытка создать равносторонний треугольник с отрицательной стороной:")
eq3 = Equilateral(-5)

print("\n  Попытка изменить равносторонний треугольник на неравносторонний:")
eq4 = Equilateral(5)
print("  Исходный треугольник:")
eq4.display()
print("  Пытаемся изменить только одну сторону:")
eq4.set_sides(a=10)
eq4.display()

```

```

1. Работа с обычным треугольником:
Треугольник со сторонами: a=3.00, b=4.00, c=5.00
Периметр: 12.00
Углы: A=36.9°, B=53.1°, C=90.0°

Изменяем стороны на 5, 6, 7:
Треугольник со сторонами: a=5.00, b=6.00, c=7.00
Периметр: 18.00
Углы: A=44.4°, B=57.1°, C=78.5°

2. Создание равностороннего треугольника:
Равносторонний треугольник со стороной: 3.00
Периметр: 9.00
Все углы: 60.0°
Площадь: 3.90

Изменяем сторону на 5:
Равносторонний треугольник со стороной: 5.00
Периметр: 15.00
Все углы: 60.0°
Площадь: 10.83

3. Проверка наследования:
Периметр равностороннего треугольника: 15.00
Углы равностороннего треугольника: (60.000000000001, 60.000000000001, 60.000000000001)

4. Ввод обычного треугольника с клавиатуры:
Введите сторону a: 3
Введите сторону b: 4
Введите сторону c: 5
Треугольник со сторонами: a=3.00, b=4.00, c=5.00
Периметр: 12.00
Углы: A=36.9°, B=53.1°, C=90.0°

5. Ввод равностороннего треугольника с клавиатуры:
Введите сторону равностороннего треугольника: 4
Равносторонний треугольник со стороной: 4.00
Периметр: 12.00
Все углы: 60.0°
Площадь: 6.93

```

Рисунок 1. Пример работы кода

## Задание 2.

7. Создать абстрактный базовый класс Root (корень) с виртуальными методами вычисления корней и вывода результата на экран. Определить производные классы Linear (линейное уравнение) и Square (квадратное уравнение) с собственными методами вычисления корней и вывода на экран.

Код программы:

```

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

from abc import ABC, abstractmethod
import math

class Reader:
    """Вспомогательный класс для ввода данных с клавиатуры"""

    @staticmethod
    def read_float(prompt):
        """Чтение вещественного числа"""
        while True:
            try:
                return float(input(prompt))
            except ValueError:
                print("Ошибка: введите число")

```

```

@staticmethod
def read_int(prompt):
    """Чтение целого числа"""
    while True:
        try:
            return int(input(prompt))
        except ValueError:
            print("Ошибка: введите целое число")

class Root(ABC):
    """Абстрактный базовый класс для корней уравнений"""

    @abstractmethod
    def calculate_roots(self):
        """Абстрактный метод вычисления корней"""
        pass

    @abstractmethod
    def display(self):
        """Абстрактный метод вывода результата"""
        pass

    @abstractmethod
    def __str__(self):
        """Строковое представление уравнения"""
        pass

    def __repr__(self):
        """Представление для отладки"""
        return str(self)

class Linear(Root):
    """Класс для линейного уравнения: a*x + b = 0"""

    def __init__(self, a=1, b=0):
        if a == 0:
            raise ValueError(
                "Коэффициент a не может быть равен 0 для линейного уравнения"
            )
        self.a = a
        self.b = b
        self.roots = []
        self.calculate_roots() # вычисляем корни сразу при создании

    @classmethod
    def read_from_keyboard(cls):
        """Создание объекта через ввод с клавиатуры"""
        print("Введите коэффициенты линейного уравнения a*x + b = 0")
        a = Reader.read_float("a = ")
        b = Reader.read_float("b = ")

```

```

while a == 0:
    print("Коэффициент а не может быть равен 0 для линейного уравнения")
    a = Reader.read_float("a = ")

return cls(a, b)

def calculate_roots(self):
    """Вычисление корня линейного уравнения: x = -b/a"""
    try:
        root = -self.b / self.a
        self.roots = [root]
        return self.roots
    except ZeroDivisionError:
        self.roots = []
        return self.roots

def display(self):
    """Вывод уравнения и его корней"""
    print(f"Уравнение: {self}")
    if self.roots:
        print(f"Корень: x = {self.roots[0]:.2f}")
    else:
        print("Корней нет")

def __str__(self):
    """Строковое представление линейного уравнения"""
    b_sign = "+" if self.b >= 0 else "-"
    b_abs = abs(self.b)
    return f'{self.a:.2f}*x {b_sign} {b_abs:.2f} = 0'

class Square(Root):
    """Класс для квадратного уравнения: a*x^2 + b*x + c = 0"""

    def __init__(self, a=1, b=0, c=0):
        if a == 0:
            raise ValueError(
                "Коэффициент а не может быть равен 0 для квадратного уравнения"
            )
        self.a = a
        self.b = b
        self.c = c
        self.roots = []
        self.calculate_roots() # вычисляем корни сразу при создании

    @classmethod
    def read_from_keyboard(cls):
        """Создание объекта через ввод с клавиатуры"""
        print("Введите коэффициенты квадратного уравнения a*x^2 + b*x + c = 0")
        a = Reader.read_float("a = ")
        b = Reader.read_float("b = ")
        c = Reader.read_float("c = ")

```

```

while a == 0:
    print("Коэффициент a не может быть равен 0 для квадратного уравнения")
    a = Reader.read_float("a = ")

return cls(a, b, c)

def calculate_roots(self):
    """Вычисление корней квадратного уравнения"""
    try:
        discriminant = self.b**2 - 4 * self.a * self.c

        if discriminant > 0:
            # Два различных корня
            x1 = (-self.b + math.sqrt(discriminant)) / (2 * self.a)
            x2 = (-self.b - math.sqrt(discriminant)) / (2 * self.a)
            self.roots = [x1, x2]
        elif discriminant == 0:
            # Один корень (кратности 2)
            x = -self.b / (2 * self.a)
            self.roots = [x]
        else:
            # Действительных корней нет
            self.roots = []

        return self.roots
    except ZeroDivisionError:
        self.roots = []
        return self.roots

def display(self):
    """Вывод уравнения и его корней"""
    print(f"Уравнение: {self}")

    discriminant = self.b**2 - 4 * self.a * self.c

    if discriminant > 0:
        print(f"Дискриминант: {discriminant:.2f} > 0")
        print(
            f"Два различных корня: x1 = {self.roots[0]:.2f}, x2 = {self.roots[1]:.2f}"
        )
    elif discriminant == 0:
        print(f"Дискриминант: {discriminant:.2f} = 0")
        print(f"Один корень (кратности 2): x = {self.roots[0]:.2f}")
    else:
        print(f"Дискриминант: {discriminant:.2f} < 0")
        print("Действительных корней нет")

def __str__(self):
    """Строковое представление квадратного уравнения"""
    b_sign = "+" if self.b >= 0 else "-"
    c_sign = "+" if self.c >= 0 else "-"
    b_abs = abs(self.b)

```

```

c_abs = abs(self.c)
return f'{self.a:.2f}*x^2 {b_sign} {b_abs:.2f}*x {c_sign} {c_abs:.2f} = 0"

def demonstrate_virtual_call(root_obj):
    """Функция, демонстрирующая виртуальный вызов"""
    print("\n" + "=" * 50)
    print("Демонстрация виртуального вызова:")
    print("Тип объекта:", type(root_obj).__name__)
    print("-" * 30)

    # Виртуальный вызов абстрактных методов
    print("Результат str():", str(root_obj))
    print("Корни:", root_obj.calculate_roots())
    print("Вызов display():")
    root_obj.display()
    print("=" * 50)

if __name__ == "__main__":
    print("1. Линейные уравнения:")
    print("-" * 30)

    # Через ввод с клавиатуры
    print("\nСоздадим линейное уравнение через ввод:")
    lin2 = Linear.read_from_keyboard()
    demonstrate_virtual_call(lin2)

    print("\n\n2. Квадратные уравнения:")
    print("-" * 30)

    # Через ввод с клавиатуры
    print("\nСоздадим квадратное уравнение через ввод:")
    sq4 = Square.read_from_keyboard()
    demonstrate_virtual_call(sq4)

    print("\n\n3. Демонстрация полиморфизма (список уравнений):")
    print("-" * 50)

equations = [
    Linear(4, -8), # 4x - 8 = 0, корень: 2
    Square(1, -3, 2), # x^2 - 3x + 2 = 0, корни: 1 и 2
    Linear(-2, 10), # -2x + 10 = 0, корень: 5
    Square(2, 0, -8), # 2x^2 - 8 = 0, корни: 2 и -2
    Square(1, 2, 5), # x^2 + 2x + 5 = 0, корней нет
]

for i, eq in enumerate(equations, 1):
    print(f"\nУравнение {i}:")
    eq.display()
    print(f"Тип: {type(eq).__name__}")
    print("-" * 30)

```

```
1. Линейные уравнения:  
-----  
  
Создадим линейное уравнение через ввод:  
Введите коэффициенты линейного уравнения a*x + b = 0  
a = 2  
b = 4  
  
=====  
Демонстрация виртуального вызова:  
Тип объекта: Linear  
  
Результат str(): 2.00*x + 4.00 = 0  
Корни: [-2.0]  
Вызов display():  
Уравнение: 2.00*x + 4.00 = 0  
Корень: x = -2.00  
=====  
  
2. Квадратные уравнения:  
-----  
  
Создадим квадратное уравнение через ввод:  
Введите коэффициенты квадратного уравнения a*x^2 + b*x + c = 0  
a = 2  
b = 5  
c = 7  
  
=====  
Демонстрация виртуального вызова:  
Тип объекта: Square  
  
Результат str(): 2.00*x^2 + 5.00*x + 7.00 = 0  
Корни: []  
Вызов display():  
Уравнение: 2.00*x^2 + 5.00*x + 7.00 = 0  
Дискриминант: -31.00 < 0  
Действительных корней нет  
=====
```

Рисунок 2. Пример работы кода

### Контрольные вопросы:

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

Наследование – это механизм, позволяющий создать новый класс на основе существующего, унаследовав его атрибуты и методы. В Python наследование реализуется путем указания родительского класса в скобках после имени нового класса: `class ChildClass(ParentClass)`. Python поддерживает как одиночное, так и множественное наследование.

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

Полиморфизм – это концепция в объектно-ориентированном программировании, которая позволяет методам с одинаковыми именами работать с объектами разных классов, обеспечивая при этом уникальное поведение для каждого класса. В Python полиморфизм достигается за счет переопределения методов в дочерних классах.

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

Утиная типизация – это принцип, когда тип объекта определяется не его принадлежностью к классу, а наличием нужных методов и атрибутов. В Python этот принцип выражается в том, что методы могут работать с объектами разных классов, если у них есть требуемые методы или атрибуты.

4. Каково назначение модуля abc языка программирования Python?

Модуль abc предоставляет инструменты для создания абстрактных базовых классов, которые содержат объявления методов, не имеющие реализации. Этот модуль позволяет определять методы, которые обязаны реализовать дочерние классы.

5. Как сделать некоторый метод класса абстрактным? Чтобы сделать метод класса абстрактным, используется декоратор `@abstractmethod` из модуля abc.

6. Как сделать некоторое свойство класса абстрактным? Для создания абстрактного свойства класса применяется декоратор `@property` вместе с `@abstractmethod`.

7. Каково назначение функции `isinstance`? Функция `isinstance()` проверяет, является ли объект экземпляром указанного класса или его подкласса. Например, вызов `isinstance(dog, Animal)` вернет True, если объект `dog` является экземпляром класса `Animal` или его наследников.

**Вывод:** в ходе выполнения работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.