МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 4.3

«Наследование и полиморфизм в языке Python»

по дисциплине «Основы программной инженерии»

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

« » мая 2023 г.

Подпись студента_____

Работа защищена

« »_____20__г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2023

Цель работы:

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

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: https://github.com/afk552/lab4.3

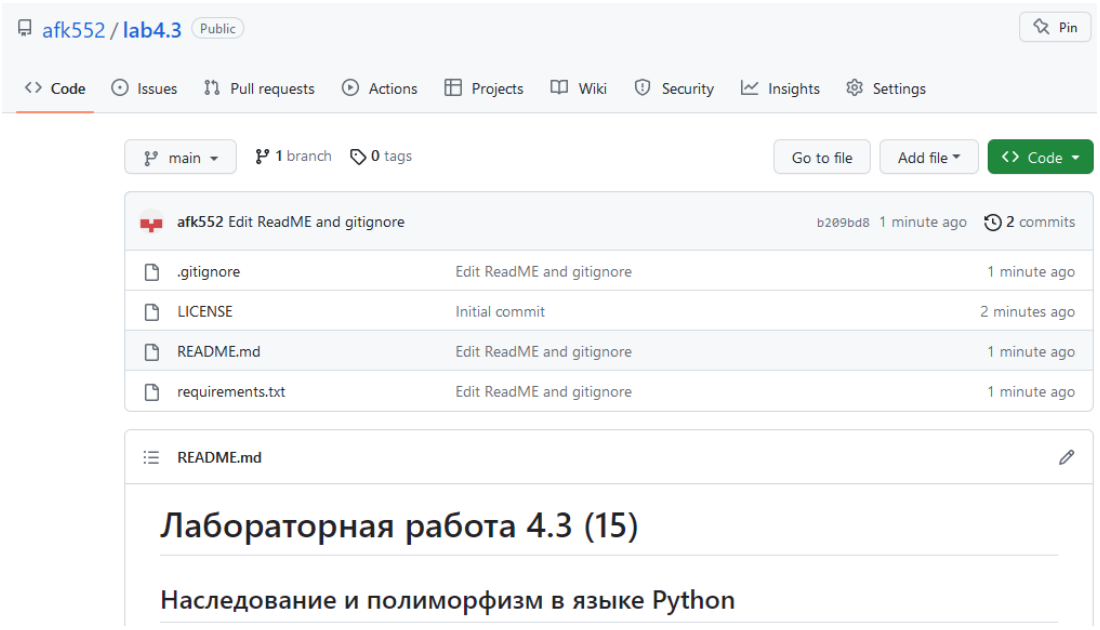

Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

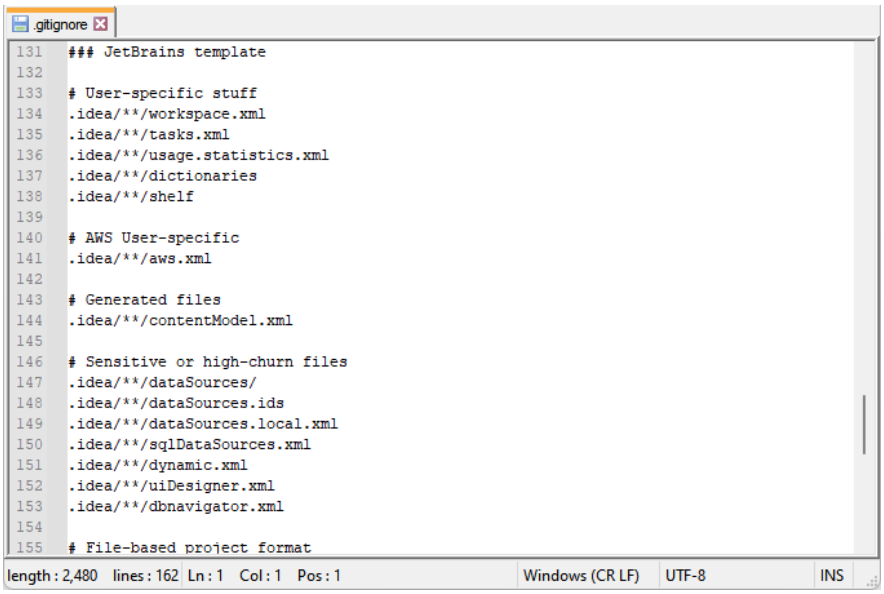

Рисунок 2 – Окно блокнота

Организуйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab4.3>git checkout -b develop
Switched to a new branch 'develop'

C:\git\lab4.3>
```

Рисунок 3 – Окно командной строки

Примеры

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


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):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height


if __name__ == '__main__':
    rect = Rectangle(10, 20, "green")
    print(rect.width,
          rect.height,
          rect.color
          )
    rect.color = "red"
    print(rect.color)
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex1.py
10 20 green
red

Process finished with exit code 0
```

Рисунок 4 – Пример 1

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


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

    def info(self):
        print("Figure")
        print("Color: " + self.__color)


class Rectangle(Figure):
    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

    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()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex2.py
Figure
Color: orange
Rectangle
Color: green
Width: 10
Height: 20
Area: 200

Process finished with exit code 0
```

Рисунок 5 – Пример 2

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


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

DeskTable › square()

Run ex3

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex3.py
0.48

Process finished with exit code 0
```

Рисунок 6 – Пример 3

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

1 usage
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
"""


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


if __name__ == '__main__':
    t4 = KitchenTable(1.5, 2, 0.75, 6)
    print(t4.height)
```

```
Run  ex4
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex4.py
0.75

Process finished with exit code 0
```

Рисунок 7 – Пример 4

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


# Python program showing
# implementation of abstract
# class through subclassing
import abc


3 usages
class parent:
    def geeks(self):
        pass


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


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

```
Run  ex5
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex5.py
True
True

Process finished with exit code 0
```

Рисунок 8 – Пример 5

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


# Python program invoking a
# method using super()
from abc import ABC


1 usage
class R(ABC):
    1 usage
    def rk(self):
        print("Abstract Base Class")


1 usage
class K(R):
    1 usage
    def rk(self):
        super().rk()
        print("subclass")


if __name__ == '__main__':
    r = K()
    r.rk()
```

```
Run   ex6
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex6.py
Abstract Base Class
subclass
```

Рисунок 9 – Пример 6

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


13 usages
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

    2 usages
    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

    14 usages
    @property
    def numerator(self):
        return self.__numerator

    18 usages
    @property
    def denominator(self):
        return self.__denominator

    # Прочитать значение дроби с клавиатуры. Дробь вводится
    # как a/b.
    1 usage
    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
    6 usages
    def display(self):
        print(f"{self.__numerator}/{self.__denominator}")

    # Сложение обыкновенных дробей.
    1 usage
    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.
    1 usage
    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.
    1 usage
    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.
    1 usage
    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()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task1.py
3/4
Введите обыкновенную дробь: 3/2
3/2
9/4
3/4
9/8
2/1

Process finished with exit code 0
```

Рисунок 10 – Пример 7

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


# Python program showing
# abstract base class work
from abc import ABC, abstractmethod


4 usages
class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass


1 usage
class Triangle(Polygon):
    # overriding abstract method
    1 usage
    def noofsides(self):
        print("I have 3 sides")


1 usage
class Pentagon(Polygon):
    # overriding abstract method
    1 usage
    def noofsides(self):
        print("I have 5 sides")


1 usage
class Hexagon(Polygon):
    # overriding abstract method
    1 usage
    def noofsides(self):
        print("I have 6 sides")


1 usage
class Quadrilateral(Polygon):
    # overriding abstract method
    1 usage
    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()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task2.py
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0
```

Рисунок 11 – Пример 8

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


# Python program showing
# abstract base class work
from abc import ABC


4 usages
class Animal(ABC):
    def move(self):
        pass


1 usage
class Human(Animal):
    1 usage
    def move(self):
        print("I can walk and run")


1 usage
class Snake(Animal):
    1 usage
    def move(self):
        print("I can crawl")


1 usage
class Dog(Animal):
    1 usage
    def move(self):
        print("I can bark")


1 usage
class Lion(Animal):
    1 usage
    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()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task3.py
I can walk and run
I can crawl
I can bark
I can roar
```

Рисунок 12 – Пример 9

8, 9. Решите задачу, 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.

```
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 Soldier(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__':
    # Создаем героев
    h1 = Hero(1)
    h2 = Hero(2)

    # Создаем команды
    team1 = []
    team2 = []
```

```
    # Рандомим и добавляем в команды
    amount = 10
    for i in range(amount):
        team = randint(1, 2)
        if team == 1:
            team1.append(Soldier(1))
        else:
            team2.append(Soldier(2))

    # Увеличения уровня героя в зависимости от длины списка
    if len(team1) > len(team2):
        h1.increment_level()
    else:
        h2.increment_level()

    # Следование за героем
    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()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\task8_9.py
Герой №1:
Команда: 1
ID: 0
Уровень: 0

Герой №2:
Команда: 2
ID: 1
Уровень: 1

Солдат команды №2:
Команда: 2
ID: 1
Следует за героем: False

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

Process finished with exit code 0
```

Рисунок 13 – Код программы заданий 8, 9

Индивидуальное задание.

Задание 1. Создать класс Man (человек), с полями: имя, возраст, пол и вес. Определить методы переназначения имени, изменения возраста и изменения веса. Создать производный класс Student, имеющий поле года обучения. Определить методы переназначения и увеличения года обучения.

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\indiv_1.py

Имя: Andrey
Пол: Male
Возраст: 18
Вес: 50

Имя: Andrey
Пол: Male
Возраст: 20
Вес: 43

Имя: Test
Пол: Female
Возраст: 23
Вес: 100
Год обучения: 2

Имя: Jean
Пол: Female
Возраст: 22
Вес: 100
Год обучения: 3

Process finished with exit code 0
```

Рисунок 14 – Результат выполнения программы индивидуального задания 1

```python
class Man:
    # Инициализация класса "Человек"
    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

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


# Производный класс "Человек" -> "Студент"
class Student(Man):
    def __init__(self, name, age, sex, weight, study_year):
        # Доступ к наследуемым элементам
        super().__init__(name, age, sex, weight)
        # Поле "Год обучения"
        self.study_year = study_year

    @property
    def study_year(self):
        return self.__study_year

    @study_year.setter
    def study_year(self, value):
        self.__study_year = value

    def set_study_year(self, value):
        self.study_year = value

    def increase_study_year(self):
        self.study_year += 1

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


if __name__ == "__main__":
    m1 = Man(name="Andrey", age=18, sex="Male", weight=50)
    m1.display()
    m1.set_age(20)
    m1.set_weight(43)
    m1.display()

    m2 = Student(name="Test", age=23, sex="Female", weight=100, study_year=2)
    m2.display()
    m2.increase_study_year()
    m2.set_name("Jean")
    m2.set_age(22)
    m2.display()
```

Рисунок 15 – Код программы индивидуального задания 1

Задание 2. Создать абстрактный базовый класс Triangle для представления треугольника с виртуальными функциями вычисления площади и периметра. Поля данных должны включать две стороны и угол между ними. Определить классы-наследники: прямоугольный треугольник, равнобедренный треугольник, равносторонний треугольник со своими функциями вычисления площади и периметра.

```python
from math import pi, sqrt, cos, sin
from abc import ABC, abstractmethod


# Базовый абстрактный класс "Треугольник"
3 usages
class Triangle(ABC):
    @abstractmethod
    def __init__(self):
        pass

    1 usage
    @abstractmethod
    def square(self):
        pass

    1 usage
    @abstractmethod
    def perimeter(self):
        pass

    # Функция вывода
    3 usages
    def print_info(self):
        print(f"Площадь треугольника = {self.square()}")
        print(f"Периметр треугольника = {self.perimeter()}")


# Класс-наследник "Прямоугольный треугольник"
1 usage
class RightTriangle(Triangle):
    def __init__(self, side1, side2, angle):
        self.__side1 = side1
        self.__side2 = side2
        self.__angle = angle * pi

    1 usage
    def square(self):
        return 0.5 * self.__side1 * self.__side2

    1 usage
    def perimeter(self):
        return (
            self.__side1
            + self.__side2
            + sqrt(
                self.__side1**2
                + self.__side2**2
                - 2 * self.__side1 * self.__side2 * cos(self.__angle)
            )
        )
```

```python
class IsoscelesTriangle(Triangle):
    def __init__(self, side1, side2, angle):
        self.__side1 = side1
        self.__side2 = side2
        self.__angle = angle * pi

    1 usage
    def square(self):
        return 0.5 * self.__side1 * self.__side2 * sin(self.__angle)

    1 usage
    def perimeter(self):
        return (
            self.__side1
            + self.__side2
            + sqrt(
                self.__side1**2
                + self.__side2**2
                - 2 * self.__side1 * self.__side2 * cos(self.__angle)
            )
        )


# Класс-наследник "Равнобедренный треугольник"
1 usage
class EquilateralTriangle(Triangle):
    def __init__(self, side1, side2, angle):
        # Если неверно заданы стороны или угол -> ошибка
        if side1 != side2 or angle != 60:
            raise ValueError("Не равнобедренный треугольник!")
        else:
            self.__side1 = side1
            self.__side2 = side2
            self.__angle = angle * pi

    1 usage
    def square(self):
        return (sqrt(3.0) / 4) * self.__side1**2

    1 usage
    def perimeter(self):
        return 3 * self.__side1
```

```python
if __name__ == "__main__":
    # Проверка классов, задание значений, расчет
    tr1 = RightTriangle(3, 4, 30)
    print(f"Площадь прямоугольного треугольника: {tr1.square()}")
    print(f"Периметр прямоугольного треугольника: {tr1.perimeter()}")
    tr2 = IsoscelesTriangle(2, 2, 45)
    print(f"Площадь равнобедренного треугольника: {tr2.square()}")
    print(f"Периметр равнобедренного треугольника: {tr2.perimeter()}")
    tr3 = EquilateralTriangle(5, 5, 60)
    print(f"Площадь равностороннего треугольника: {tr3.square()}")
    print(f"Периметр равностороннего треугольника: {tr3.perimeter()}")
    print("-" * 20)
    # Проверка функции вывода посчитанных значений
    tr1.print_info()
    tr2.print_info()
    tr3.print_info()
```

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\indiv_2.py
Площадь прямоугольного треугольника: 6.0
Периметр прямоугольного треугольника: 8.0
Площадь равнобедренного треугольника: 3.9163938347251765e-15
Периметр равнобедренного треугольника: 8.0
Площадь равностороннего треугольника: 10.825317547305483
Периметр равностороннего треугольника: 15
--------------------
Площадь треугольника = 6.0
Периметр треугольника = 8.0
Площадь треугольника = 3.9163938347251765e-15
Периметр треугольника = 8.0
Площадь треугольника = 10.825317547305483
Периметр треугольника = 15

Process finished with exit code 0
```

Рисунок 16 – Индивидуальное задание 2 (код, результат)

Вывод: Были приобретены и применены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

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 нужно. 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 - это возможность объектов с одинаковой сигнатурой методов вызывать разные реализации этого метода в зависимости от текущего типа объекта. В Python полиморфизм реализуется через вызов методов класса объекта без необходимости указывать явно тип объекта.

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

"Утиная" типизация - это стиль программирования, при котором проверка на соответствие типу объекта происходит во время выполнения, а не на этапе компиляции. В Python все объекты имеют общий тип object, и проверка соответствия типу может быть выполнена с помощью ключевого слова isinstance.

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

5. Как сделать некоторый метод класса абстрактным?

Для того чтобы сделать метод класса абстрактным, нужно создать абстрактный метод в базовом классе с помощью декоратора @abstractmethod. Этот метод не должен иметь реализации в базовом классе, и должен быть переопределен в каждом наследнике.

6. Как сделать некоторое свойство класса абстрактным?

Для того чтобы сделать свойство класса абстрактным, нужно создать абстрактное свойство в базовом классе с помощью декоратора @abstractmethod. Это свойство не должно иметь реализации в базовом классе, и должно быть переопределено в каждом наследнике.

7. Каково назначение функции isinstance?

Функция isinstance используется для проверки соответствия типа объекта указанному классу или его наследнику. Она принимает два аргумента: объект, тип которого нужно проверить, и класс или кортеж классов, с которым нужно сравнить тип объекта. Если объект является экземпляром указанного класса или его наследника, то функция возвращает True, в противном случае - False.