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 №3

Дисциплины «Объектно-ориентированное программирование»

Выполнил:

Пустяков Андрей Сергеевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента цифровых и робототехнических систем и электроники института перспективной инженерии

_____

(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Наследование и полиморфизм в языке Python.

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

Ход работы:

Проработка примеров лабораторной работы:

Пример 1.

Рациональная (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 + be, bd);
– вычитания sub, (a, b) – (c, d) = (ad – be, bd);
– умножения mul, (a, b) * (c, d) = (ac, bd);
– деления div, (a,b) / (c, d) = (ad, be);
– сравнения equal, greate, less.

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

Код программы задания 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()
```

Результаты работы кода примера 1 (в качестве дроби берем дробь 5/3) (рис. 1).

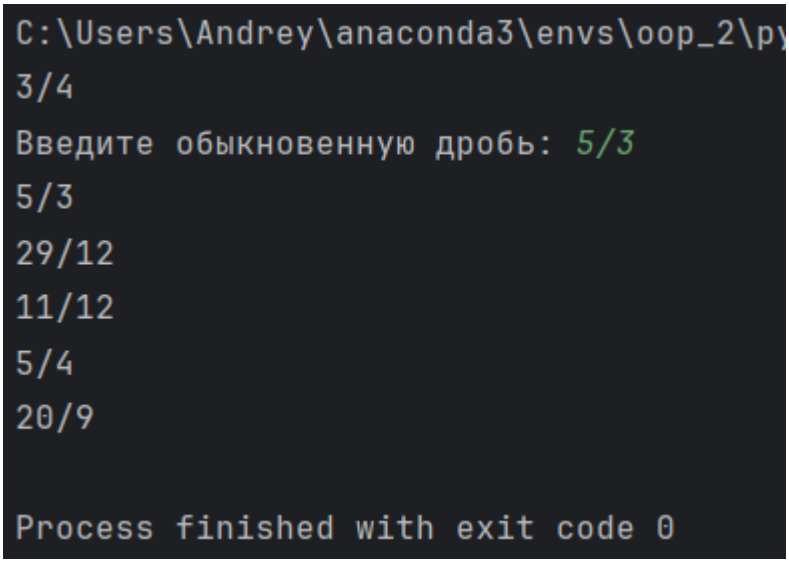

Рисунок 1 – Результаты работы программы примера 1

Пример 2.

Необходимо создать некоторый абстрактный класс как образец для других классов. Код программы реализации данного класса:

```
#!/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")


if __name__ == '__main__':
    # Driver code
    R = Triangle()
    R.noofsides()
    K = Quadrilateral()
    K.noofsides()
    R = Pentagon()
    R.noofsides()
    K = Hexagon()
    K.noofsides()
```

Результаты работы программы примера 2 (рис. 2).

```
C:\Users\Andrey\anaconda3\envs\oop_2\pythor
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0
```

Рисунок 2 – Результаты работы программы примера 2

Пример 3.

Необходимо создать абстрактный класс «Animal», который будет использоваться для других классов, отвечающих за других животных. Код примера 3 с реализациями данных подклассов:

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

# Python program showing
# abstract base class work

from abc import ABC, abstractmethod


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()
```

Результаты работы данной программы (рис. 3).

```
C:\Users\Andrey\anaconda3\envs\oop_2\python
I can walk and run
I can crawl
I can bark
I can roar

Process finished with exit code 0
```

Рисунок 3 – результаты работы методов классов животных

Задание 1.

Необходимо разработать программу по следующему описанию (реализовать в программе полиморфизм):

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.

Код программы, реализующей данные операции:

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


import random


# 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:
    """
    Базовый класс для всех юнитов игры.
    """

    def __init__(self, unique_id, species):
        """
```

```
        Конструктор класса Юнит.

        :param unique_id: ID юнита, должен быть уникальным и для солдат и для
героев.
        :param species: Принадлежность юнита команде (имя команды).
        """
        self.unique_id = unique_id
        self.team = species


class Hero(Unit):
    """
    Класс Герой, который наследует класс Unit(Юнит), так как сам тоже
является юнитом.
    """

    def __init__(self, unique_id, species):
        """
        Конструктор класса герой.

        :param unique_id: ID героя.
        :param species: Принадлежность героя команде (имя команды).
        """

        super().__init__(unique_id, species)
        self.level = 1  # Начальный уровень героя

    def increase_level(self):
        """
        Метод, отвечающий за повышения уровня героя.
        """

        self.level += 1
        print(f"Герой {self.unique_id}, принадлежащий расе {self.team}
повышается до {self.level} уровня.")


class Soldier(Unit):
    """
    Класс Солдат, стандартный юнит игры, поэтому тоже наследует класс
Unit(Юнит).
    Конструктор класса полностью наследуется из класса Unit.
    """

    def follow_the_hero(self, hero):
        """
        Метод, отвечающий за следование зе героем.

        :param hero: Герой, за которым нужно следовать.
        """

        print(f"Солдат {self.unique_id} идет за героем {hero.unique_id} расы
{hero.team}.")


if __name__ == "__main__":
    # Создаем героев для двух команд
    hero1 = Hero(unique_id=1, species="Альянс")
    hero2 = Hero(unique_id=2, species="Нежить")

    # Списки солдат для команд
    team_alliance_soldiers = []
    team_undead_soldiers = []
```

```
    # Генерация солдат
    for i in range(1, 201):  # пусть будет 200 солдат (по 100 единиц для
каждой команды в идеале)
        team = random.choice(["Альянс", "Нежить"])
        soldier = Soldier(unique_id=i, species=team)
        if team == "Альянс":
            team_alliance_soldiers.append(soldier)
        else:
            team_undead_soldiers.append(soldier)

    # Вывод численности солдат команд
    print(f"Раса Альянс, численность: {len(team_alliance_soldiers)} солдат.")
    print(f"Раса Нежить, численность: {len(team_undead_soldiers)} солдат.")

    # Определение команды с более длинным списком
    if len(team_alliance_soldiers) > len(team_undead_soldiers):
        hero1.increase_level()
    elif len(team_undead_soldiers) > len(team_alliance_soldiers):
        hero2.increase_level()
    else:
        print("Количество солдат обеих рас одинаково, уровни героев не
увеличиваются.")

    # Отправляем одного из солдат первого героя следовать за ним
    if team_alliance_soldiers:  # Проверяем, есть ли солдаты у расы Альянс
        soldier = team_alliance_soldiers[0]
        soldier.follow_the_hero(hero1)
        print(f"Идентификаторы: Солдат с id={soldier.unique_id}, "
              f"идет за героем с id={hero1.unique_id} расы {hero1.team}.")
    else:
        print("У расы Альянс нет солдат для выполнения команды.")
```

В данном коде класс Hero и класс Soldier наследуют класс Union, а именно такие атрибуты как id и принадлежность команде (предполагается, что и герои, и солдаты являются юнитами).

Результаты работы данного кода (вывод информации о повышении уровня и герое с солдатом первой команды) (рис. 4).

```
C:\Users\Andrey\anaconda3\envs\oop_2\python.exe C:\Users\Andrey\De
Раса Альянс, численность: 93 солдат.
Раса Нежить, численность: 107 солдат.
Герой 2, принадлежащий расе Нежить повышается до 2 уровня.
Солдат 7 идет за героем 1 расы Альянс.
Идентификаторы: Солдат с id=7, идет за героем с id=1 расы Альянс.

Process finished with exit code 0
```

Рисунок 4 – Результаты работы программы

Выполнение индивидуальных заданий:

Вариант 25

Задание 1.

Необходимо создать программу с использованием иерархии классов.

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 (Вариант 25 (5)).

Код программы с реализацией данных классов:

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

# 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.
# (Вариант 25 (5)).


class Man:
    def __init__(self, name, age, gender, weight):
        """
        Конструктор основного класса Man.

        :param name: имя человека;
        :param age: возраст человека;
        :param gender: пол человека;
        :param weight: вес человека.
        """
        self.name = name
        self.age = age
        self.gender = gender
        self.weight = weight

    def rename(self, new_name):
        """Изменяет имя человека."""
        self.name = new_name

    def change_age(self, new_age):
        """Изменяет возраст человека."""
        if new_age > 0:
            self.age = new_age
        else:
            print("Возраст должен быть положительным числом!")

    def change_weight(self, new_weight):
        """Изменяет вес человека."""
        if new_weight > 0:
            self.weight = new_weight
        else:
            print("Вес должен быть положительным числом!")


class Student(Man):
    def __init__(self, name, age, gender, weight, year_of_study):
```

```
        """
        Конструктор класса Student (наследует класс Man).

        :param name: имя;
        :param age: возраст;
        :param gender: пол;
        :param weight: вес;
        :param year_of_study: год обучения.
        """
        super().__init__(name, age, gender, weight)  # то, что унаследовали
от Man
        self.year_of_study = year_of_study

    def change_year_of_study(self, new_year):
        """Изменяет год обучения студента."""
        if new_year > 0:
            self.year_of_study = new_year
        else:
            print("Год обучения должен быть положительным числом!")

    def increment_year_of_study(self):
        """Увеличивает год обучения на 1."""
        self.year_of_study += 1


if __name__ == "__main__":
    # Для примера создадим объект man
    man = Man("Иван", 30, "мужской", 75)
    print(f"Человек: {man.name}, {man.age} лет, {man.gender}, его вес:
{man.weight} кг")

    man.rename("Алексей")  # Смена имени человека
    man.change_age(35)     # Смена возраста человека
    man.change_weight(80)  # Изменение веса человека
    print(f"Изменения: {man.name}, {man.age} лет, вес {man.weight} кг")

    # Для примера создадим объект класса Student
    student = Student("Ольга", 20, "женский", 55, 1)
    print(f"Студент: {student.name}, {student.age} лет, {student.gender}, вес
{student.weight} кг, год обучения: "
          f"{student.year_of_study}")

    student.increment_year_of_study()
    print(f"Увеличение года обучения: Студент: {student.name}, {student.age}
лет, {student.gender}, вес "
          f"{student.weight} кг, год обучения: {student.year_of_study}")

    student.change_year_of_study(4)
    print(f"Изменения года обучения: Студент: {student.name}, {student.age}
лет, {student.gender}, вес "
          f"{student.weight} кг, год обучения: {student.year_of_study}")
```

Результаты работы программы с использованием иерархии классов (рис. 5). Для демонстрации возможностей данных классов был создан объект класса Man и объект класса Student, также все операции по изменению атрибутов для обоих объектов.

```
C:\Users\Andrey\anaconda3\envs\oop_2\python.exe C:\Users\Andrey\Desktop\ООП\Лабораторн
Человек: Иван, 30 лет, мужской, его вес: 75 кг
Изменения: Алексей, 35 лет, вес 80 кг
Студент: Ольга, 20 лет, женский, вес 55 кг, год обучения: 1
Увеличение года обучения: Студент: Ольга, 20 лет, женский, вес 55 кг, год обучения: 2
Изменения года обучения: Студент: Ольга, 20 лет, женский, вес 55 кг, год обучения: 4

Process finished with exit code 0
```

Рисунок 5 – Результаты работы программы

Задание 2.

Необходимо реализовать абстрактный класс, определив в нем 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 переопределенных абстрактных методов. Необходимо написать функцию вывода, получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов.

Создать абстрактный базовый класс Function (функция) с виртуальными методами вычисления значения функции $y = f(x)$ в заданной точке x и вывода результата на экран. Определить производные классы Ellipse (эллипс), Hyperbola (гипербола) с собственными функциями вычисления y в зависимости от входного параметра x. Уравнение эллипса: $x^2 / a^2 + y^2 / b^2 = 1$; гиперболы: $x^2 / a^2 – y^2 / b^2 = 1$ (Вариант 25 (8)).

Код программы с реализацией абстрактного класса и производных классов:

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


from abc import ABC, abstractmethod
import math


# Создать абстрактный базовый класс Function (функция) с виртуальными
методами вычисления
# значения функции y = f(x) в заданной точке x и вывода результата на экран.
# Определить производные классы Ellipse (эллипс), Hyperbola (гипербола)
# с собственными функциями вычисления y в зависимости от входного параметра
x.
# Уравнение эллипса: x^2 / a^2 + y^2 / b^2 = 1;
```

```python
# гиперболы: x^2 / a^2 - y^2 / b^2 = 1.
# (Вариант 25 (8)).


class Function(ABC):
    """Абстрактный базовый класс Function."""

    @abstractmethod
    def calculate_y(self, x):
        """
        Абстрактный метод для вычисления y = f(x).
        Он должен быть переопределен в производных классах.
        """
        pass

    @abstractmethod
    def display_result(self, x):
        """
        Абстрактный метод для вывода результата на экран.
        Он также должен быть переопределен в производных классах.
        """
        pass


class Ellipse(Function):
    """Класс для вычисления значения функции по уравнению эллипса."""

    def __init__(self, a, b):
        """
        Инициализация эллипса с полуосями a и b (коэффициенты).
        """
        self.a = a
        self.b = b

    def calculate_y(self, x):
        """
        Вычисление y на основе уравнения эллипса:
        x^2 / a^2 + y^2 / b^2 = 1.
        Метод, который переопределяет метод из абстрактного класса.
        """

        if abs(x) > self.a:
            raise ValueError("Значение x выходит за границы эллипса (|x| <=
a).")
        return self.b * math.sqrt(1 - (x ** 2) / (self.a ** 2))

    def display_result(self, x):
        """Вывод результата на экран, переопределение метода вывода
абстрактного класса."""
        try:
            y = self.calculate_y(x)
            print(f"Эллипс: при x = {x}, y = ±{y:.2f}")
        except ValueError as e:
            print(f"Ошибка: {e}")


class Hyperbola(Function):
    """Класс для вычисления значения функции по уравнению гиперболы."""

    def __init__(self, a, b):
        """
        Инициализация гиперболы с полуосями a и b.
        """
        self.a = a
```

```
        self.b = b

    def calculate_y(self, x):
        """
        Вычисление y на основе уравнения гиперболы:
        x^2 / a^2 - y^2 / b^2 = 1.
        Метод, который переопределяет метод из абстрактного класса.
        """
        if abs(x) < self.a:
            raise ValueError("Значение x должно быть больше или равно a (|x|
>= a).")
        return self.b * math.sqrt((x ** 2) / (self.a ** 2) - 1)

    def display_result(self, x):
        """Вывод результата на экран."""
        try:
            y = self.calculate_y(x)
            print(f"Гипербола: при x = {x}, y = ±{y:.2f}")
        except ValueError as e:
            print(f"Ошибка: {e}")


def display_function_result(function_obj, x):
    """
    Функция для демонстрации виртуального вызова.
    Получает объект базового класса Function по ссылке
    и вызывает его методы.
    """
    function_obj.display_result(x)


if __name__ == "__main__":
    # Создаем объекты классов и устанавлюваем в качестве атрибутов значения
коэффициентов
    ellipse = Ellipse(a=5, b=3)
    hyperbola = Hyperbola(a=4, b=2)

    # Вызов методов напрямую (невиртуально)
    print("Прямой вызов методов:")
    ellipse.display_result(3)
    hyperbola.display_result(5)

    # Виртуальный вызов методов (через функцию display_function_result)
    print("\nВиртуальный вызов через базовый класс:")
    display_function_result(ellipse, 3)
    display_function_result(hyperbola, 5)

    # Примеры ошибок
    print("\nОбработка ошибок:")
    ellipse.display_result(6)        # Выход за границы эллипса
    hyperbola.display_result(3)      # Выход за границы гиперболы
    # abstract_function = Function()  # Попытка создать объект абстрактного
класса Function, вызовет ошибку
```

Результаты работы данного кода при создании объектов производных классов и нахождении значений функции для них (рис. 6).

```
Прямой вызов методов:
Эллипс: при x = 3, y = ±2.40
Гипербола: при x = 5, y = ±1.50

Виртуальный вызов через базовый класс:
Эллипс: при x = 3, y = ±2.40
Гипербола: при x = 5, y = ±1.50

Обработка ошибок:
Ошибка: Значение x выходит за границы эллипса (|x| <= a).
Ошибка: Значение x должно быть больше или равно a (|x| >= a).
```

Рисунок 6 – Результаты работы программы задания 2

Если попытаться создать объект абстрактного класса это вызовет ошибку (рис. 7).

```
C:\Users\Andrey\anaconda3\envs\oop_2\python.exe C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_3\Object-Or
Traceback (most recent call last):
  File "C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_3\Object-Oriented_Programming_laboratory_work_3\ind
    abstract_function = Function()  # Попытка создать объект абстрактного класса Function, вызовет ошибку
TypeError: Can't instantiate abstract class Function with abstract methods calculate_y, display_result
```

Рисунок 7 – Ошибка создания объекта абстрактного класса

Ссылка на репозиторий данной лабораторной работы: https://github.com/AndreyPust/Object-Oriented_Programming_laboratory_work_3.git

Ответы на контрольные вопросы:

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

Наследование – это принцип объектно-ориентированного программирования, который позволяет создать новый класс (производный, или дочерний) на основе существующего класса (базового, или родительского). Производный класс наследует методы и свойства базового класса, что позволяет избежать дублирования кода. В Python наследование реализуется указанием базового класса в скобках при объявлении нового класса.

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

Полиморфизм – это способность объекта обрабатывать методы с одинаковыми именами, но с различным поведением, в зависимости от типа данных или класса. В Python полиморфизм реализован через переопределение методов: дочерние классы могут переопределять методы, унаследованные от родительских классов, для создания специализированного поведения.

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

«Утиная» типизация (duck typing) — это концепция, при которой в языке программирования проверяется не тип объекта, а его поведение. Если объект имеет нужные методы или свойства, он может использоваться в данном контексте, независимо от его конкретного типа. В Python «утиная» типизация позволяет применять объекты любого класса, если они поддерживают необходимые методы.

4. Каково назначение модуля «abc» языка программирования Python?

Модуль «abc» (Abstract Base Classes) предоставляет средства для определения абстрактных базовых классов и создания абстрактных методов. Абстрактный класс – это класс, который нельзя создать напрямую; он служит как основа для других классов. Модуль «abc» позволяет реализовывать абстракции в Python, обеспечивая, чтобы дочерние классы реализовали обязательные методы.

5. Как сделать некоторый метод класса абстрактным?

Чтобы сделать метод класса абстрактным, нужно импортировать «ABC» и «abstractmethod» из модуля «abc» и аннотировать метод с помощью декоратора @abstractmethod. Абстрактный метод – это метод, который обязан быть реализован в дочерних классах. Класс, содержащий абстрактные методы, не может быть создан напрямую.

6. Как сделать некоторое свойство класса абстрактным?

Чтобы сделать свойство класса абстрактным, в модуле «abc» используется декоратор @property совместно с @abstractmethod. Это позволяет заставить дочерние классы реализовывать конкретное свойство.

7. Каково назначение функции «isinstance»?

Функция «isinstance» проверяет, является ли объект экземпляром указанного класса или кортежа классов. Она возвращает True, если объект принадлежит указанному классу (или одному из классов в кортеже), и False в противном случае. Эта функция полезна для проверки типа данных, особенно в полиморфных структурах.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по созданию иерархии классов, а также изучено понятие наследования и полиморфизма при написании программ с помощью языка программирования Python версии 3.x.