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»
Вариант 13

Выполнил:
Мотовилов Вадим Борисович
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Воронкин Р.А.

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Наследование и полиморфизм в языке Python

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

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее. Ссылка на гит: https://github.com/AkselSukub/OOP_3
2. Выполнил индивидуальное задание №1:

13. Создать класс Triad (тройка чисел); определить методы увеличения полей на 1. Определить производный класс Date с полями: год, месяц и день. Переопределить методы увеличения полей на 1 и определить метод увеличения даты на n дней.

Код индивидуального задания №1:

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

class Triad:
    """
    Класс, представляющий тройку чисел.
    """

    def __init__(self, a, b, c):
        """
        Конструктор класса.

        :param a: Первое число.
        :param b: Второе число.
        :param c: Третье число.
        """
        self.a = a
        self.b = b
        self.c = c

    def __str__(self):
        """
        Возвращает строковое представление объекта.

        :return: Строка, представляющая тройку чисел.
        """
        return f"({self.a}, {self.b}, {self.c})"

    def increase_by_one(self):
        """
        Увеличивает каждое число в тройке на 1.
        """
        self.a += 1
        self.b += 1
        self.c += 1

class Date(Triad):
```

```
"""
Класс, представляющий дату (год, месяц, день), наследуется от Triad.
"""

def __init__(self, year, month, day):
    """
    Конструктор класса.

    :param year: Год.
    :param month: Месяц.
    :param day: День.
    """
    super().__init__(year, month, day)

def __str__(self):
    """
    Возвращает строковое представление объекта Date.

    :return: Строка, представляющая дату.
    """
    return f"{self.a:04d}-{self.b:02d}-{self.c:02d}"

def increase_by_one(self):
    """
    Увеличивает год, месяц и день на 1, с учетом допустимых значений.
    """
    self.c += 1  # Сначала увеличиваем день
    self.normalize_date()

def increase_date_by_n_days(self, n):
    """
    Увеличивает дату на n дней.

    :param n: Количество дней для увеличения.
    """
    self.c += n
    self.normalize_date()

def is_leap_year(self, year):
    """
    Проверяет, является ли год високосным.

    :param year: Год для проверки.
    :return: True, если год високосный, False в противном случае.
    """
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

def days_in_month(self, year, month):
    """
    Возвращает количество дней в заданном месяце и году.

    :param year: Год.
    :param month: Месяц.
    :return: Количество дней в месяце.
    """
    if month in [4, 6, 9, 11]:
        return 30
    elif month == 2:
        return 29 if self.is_leap_year(year) else 28
    else:
        return 31

def normalize_date(self):
```

```
        """
        Нормализует дату, учитывая количество дней в месяце и високосные годы.
        """
        while self.c > self.days_in_month(self.a, self.b):
            self.c -= self.days_in_month(self.a, self.b)
            self.b += 1
            if self.b > 12:
                self.b = 1
                self.a += 1

if __name__ == '__main__':
    # Демонстрация класса Triad
    triad = Triad(1, 2, 3)
    print("Исходная тройка:", triad)
    triad.increase_by_one()
    print("Тройка после увеличения на 1:", triad)

    # Демонстрация класса Date
    date = Date(2023, 12, 31)
    print("\nИсходная дата:", date)
    date.increase_by_one()
    print("Дата после увеличения на 1 день:", date)

    date.increase_date_by_n_days(30)
    print("Дата после увеличения на 30 дней:", date)

    date2 = Date(2024, 2, 28)
    print("\nИсходная дата:", date2)
    date2.increase_by_one()
    print("Дата после увеличения на 1 день (високосный год):", date2)

    date3 = Date(2023, 1, 1)
    date3.increase_date_by_n_days(365)
    print("\nДата после увеличения на 365 дней:", date3)

    date4 = Date(2024, 1, 1)
    date4.increase_date_by_n_days(365)
    print("Дата после увеличения на 365 дней (високосный год):", date4)
```

3. Выполнил индивидуальное задание №2:

13. Создать абстрактный базовый класс Series (прогрессия) с виртуальными функциями вычисления j-го элемента прогрессии и суммы прогрессии. Определить производные классы: Linear (арифметическая) и Exponential (геометрическая). (Арифметическая прогрессия $a_j = a_0 + j \times d$, $j = 0, 1, 2, \ldots$ Сумма арифметической прогрессии: $S_n = (n + 1)(a_0 + a_n)/2$. Геометрическая прогрессия: $a_j = a_0 \times r^j$, $j = 0, 1, 2, \ldots$ Сумма геометрической прогрессии: $S_n = (a_0 - a_n r)/(1 - r)$).

Код индивидуального задания №2:

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

from abc import ABC, abstractmethod

class Series(ABC):
    """
    Абстрактный базовый класс для прогрессий.
    """
```

```
    @abstractmethod
    def __init__(self, a0):
        """
        Конструктор абстрактного класса.
        :param a0: Первый элемент прогрессии.
        """
        self._a0 = a0

    @abstractmethod
    def get_element(self, j):
        """
        Абстрактный метод для вычисления j-го элемента прогрессии.
        :param j: Индекс элемента (начиная с 0).
        :return: Значение j-го элемента.
        """
        pass

    @abstractmethod
    def get_sum(self, n):
        """
        Абстрактный метод для вычисления суммы первых n+1 элементов прогрессии (от 0 до n).
        :param n: Количество элементов для суммирования (сумма до n-го элемента включительно).
        :return: Сумма первых n+1 элементов.
        """
        pass

    @abstractmethod
    def input_data(self):
         """
         Абстрактный метод для ввода данных прогрессии.
         """
         pass

    @abstractmethod
    def output_data(self):
        """
        Абстрактный метод для вывода данных прогрессии.
        """
        pass

    @property
    def a0(self):
        """
        Свойство для доступа к первому элементу прогрессии.
        """
        return self._a0

    @a0.setter
    def a0(self, value):
        """
        Свойство для установки первого элемента прогрессии.
        """
        self._a0 = value

class Linear(Series):
    """
    Класс для арифметической прогрессии.
    """

    def __init__(self, a0, d):
        """
        Конструктор класса.
        :param a0: Первый элемент прогрессии.
```

```
        :param d: Разность арифметической прогрессии.
        """
        super().__init__(a0)
        self._d = d

    def get_element(self, j):
        """
        Вычисляет j-й элемент арифметической прогрессии.
        :param j: Индекс элемента.
        :return: Значение j-го элемента.
        """
        return self._a0 + j * self._d

    def get_sum(self, n):
        """
        Вычисляет сумму первых n+1 элементов арифметической прогрессии.
        :param n: Количество элементов для суммирования (сумма до n-го элемента включительно).
        :return: Сумма первых n+1 элементов.
        """
        an = self.get_element(n)
        return (n + 1) * (self._a0 + an) / 2

    def input_data(self):
        """
        Ввод данных для арифметической прогрессии.
        """
        try:
            self._a0 = float(input("Введите первый элемент (a0): "))
            self._d = float(input("Введите разность (d): "))
        except ValueError:
            print("Ошибка: Некорректный ввод. Введите числовые значения.")

    def output_data(self):
        """
        Вывод данных об арифметической прогрессии.
        """
        print("Арифметическая прогрессия:")
        print(f"  a0 = {self._a0}")
        print(f"  d = {self._d}")

    @property
    def d(self):
        """
        Свойство для доступа к разности арифметической прогрессии.
        """
        return self._d

    @d.setter
    def d(self, value):
        """
        Свойство для установки разности арифметической прогрессии.
        """
        self._d = value

class Exponential(Series):
    """
    Класс для геометрической прогрессии.
    """

    def __init__(self, a0, r):
        """
        Конструктор класса.
        :param a0: Первый элемент прогрессии.
```

```
        :param r: Знаменатель геометрической прогрессии.
        """
        super().__init__(a0)
        self._r = r

    def get_element(self, j):
        """
        Вычисляет j-й элемент геометрической прогрессии.
        :param j: Индекс элемента.
        :return: Значение j-го элемента.
        """
        return self._a0 * (self._r ** j)

    def get_sum(self, n):
        """
        Вычисляет сумму первых n+1 элементов геометрической прогрессии.
        :param n: Количество элементов для суммирования (сумма до n-го элемента включительно).
        :return: Сумма первых n+1 элементов.
        """
        an = self.get_element(n)
        if self._r == 1:
          return (n + 1) * self._a0
        return (self._a0 - an * self._r) / (1 - self._r)

    def input_data(self):
        """
        Ввод данных для геометрической прогрессии.
        """
        try:
            self._a0 = float(input("Введите первый элемент (a0): "))
            self._r = float(input("Введите знаменатель (r): "))
        except ValueError:
            print("Ошибка: Некорректный ввод. Введите числовые значения.")

    def output_data(self):
        """
        Вывод данных о геометрической прогрессии.
        """
        print("Геометрическая прогрессия:")
        print(f"  a0 = {self._a0}")
        print(f"  r = {self._r}")

    @property
    def r(self):
        """
        Свойство для доступа к знаменателю геометрической прогрессии.
        """
        return self._r

    @r.setter
    def r(self, value):
        """
        Свойство для установки знаменателя геометрической прогрессии.
        """
        self._r = value

def print_series_info(series, n):
    """
    Функция для вывода информации о прогрессии, используя виртуальные вызовы.
    :param series: Объект класса Series (или его потомка).
    :param n: Количество элементов для суммирования.
    """
    print("\nИнформация о прогрессии:")
```

```
    series.output_data()  # Виртуальный вызов output_data
    print(f"  {n}-й элемент: {series.get_element(n)}") # Виртуальный вызов get_element
    print(f"  Сумма первых {n+1} элементов: {series.get_sum(n)}") # Виртуальный вызов get_sum

if __name__ == '__main__':
    # Демонстрация работы классов
    linear = Linear(1, 2)
    exponential = Exponential(2, 3)

    # Демонстрация виртуального вызова через функцию print_series_info
    print_series_info(linear, 5)
    print_series_info(exponential, 5)

    # Демонстрация ввода данных
    print("\nВвод данных для арифметической прогрессии:")
    linear.input_data()
    print_series_info(linear, 5)

    print("\nВвод данных для геометрической прогрессии:")
    exponential.input_data()
    print_series_info(exponential, 5)

    # Демонстрация изменения свойств
    linear.a0 = 5
    linear.d = 3
    print("\nАрифметическая прогрессия после изменения a0 и d:")
    print_series_info(linear, 5)

    exponential.a0 = 1
    exponential.r = 2
    print("\nГеометрическая прогрессия после изменения a0 и r:")
    print_series_info(exponential, 5)
```

Ответы на 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.