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 №1
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: Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

https://github.com/AkselSukub/New_OOP1

Задание 1.

7. Поле first — дробное число, левая граница диапазона; поле second — дробное число, правая граница диапазона. Реализовать метод `rangecheck()` — проверку заданного числа на принадлежность диапазону.

Код программы:

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

class Pair:
    def __init__(self, first=0.0, second=0.0):
        if first > second:
            print("Ошибка: левая граница должна быть <= правой")
            exit()
        self.first = float(first)
        self.second = float(second)

    def read(self):
        self.first = float(input("Левая граница: "))
        self.second = float(input("Правая граница: "))
        if self.first > self.second:
            print("Ошибка: левая граница должна быть <= правой")
            exit()

    def display(self):
        print(f"[{self.first}, {self.second}]")

    def rangecheck(self, number):
        return self.first <= number <= self.second

def make_tim(first, second):
    if first > second:
        print("Ошибка: левая граница должна быть <= правой")
        exit()
    return Pair(first, second)

if __name__ == '__main__':
    print("\nСоздадим диапазон:")
    p2 = Pair()
```

```
p2.read()
print("Ваш диапазон: ", end="")
p2.display()

num = float(input("Проверить число: "))
print(f"Число {num} в диапазоне? {p2.rangecheck(num)}")
```

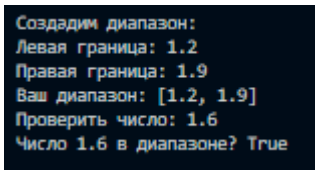

Рисунок 1. Пример работы кода

Задание 2.

7. Создать класс Date для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями типа unsigned int: для года, месяца и дня. Класс должен включать не менее трех функций инициализации: числами, строкой вида «год.месяц.день» (например, «2004.08.31») и датой. Обязательными операциями являются: вычисление даты через заданное количество дней, вычитание заданного количества дней из даты, определение високосности года, присвоение и получение отдельных частей (год, месяц, день), сравнение дат (равно, до, после), вычисление количества дней между датами.

Код программы:

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

class Date:
    def __init__(self, year=2000, month=1, day=1):
        if not self._is_valid_date(year, month, day):
            print("Ошибка: неверная дата")
            year, month, day = 2000, 1, 1
        self.year = year
        self.month = month
        self.day = day

    def _is_leap_year(self, year):
        return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)

    def _days_in_month(self, year, month):
        if month == 2:
            return 29 if self._is_leap_year(year) else 28
        if month in [4, 6, 9, 11]:
            return 30
        return 31

    def _is_valid_date(self, year, month, day):
        if month < 1 or month > 12:
            return False
        if day < 1 or day > self._days_in_month(year, month):
```

```
            return False
        return True

    @classmethod
    def from_string(cls, date_str):
        try:
            parts = date_str.split('.')
            if len(parts) != 3:
                raise ValueError
            year, month, day = map(int, parts)
            return cls(year, month, day)
        except:
            print("Ошибка формата строки")
            return cls()

    @classmethod
    def from_date(cls, other_date):
        return cls(other_date.year, other_date.month, other_date.day)

    def read(self):
        while True:
            try:
                date_str = input("Введите дату в формате 'год.месяц.день': ")
                parts = date_str.split('.')
                if len(parts) != 3:
                    print("Ошибка: используйте формат 'год.месяц.день'")
                    continue

                year, month, day = map(int, parts)
                if self._is_valid_date(year, month, day):
                    self.year, self.month, self.day = year, month, day
                    break
                else:
                    print("Ошибка: неверная дата")
            except:
                print("Ошибка ввода")

    def display(self):
        print(f"{self.year:04d}.{self.month:02d}.{self.day:02d}")

    def is_leap(self):
        return self._is_leap_year(self.year)

    def add_days(self, days):
        result = Date.from_date(self)
        result.day += days

        while result.day > result._days_in_month(result.year, result.month):
            result.day -= result._days_in_month(result.year, result.month)
            result.month += 1
            if result.month > 12:
                result.month = 1
```

```
            result.year += 1

        return result

    def subtract_days(self, days):
        result = Date.from_date(self)
        result.day -= days

        while result.day < 1:
            result.month -= 1
            if result.month < 1:
                result.month = 12
                result.year -= 1
            result.day += result._days_in_month(result.year, result.month)

        return result

    def set_date(self, year=None, month=None, day=None):
        if year is not None:
            self.year = year
        if month is not None:
            self.month = month
        if day is not None:
            self.day = day

        if not self._is_valid_date(self.year, self.month, self.day):
            print("Ошибка: неверная дата")

    def get_year(self):
        return self.year

    def get_month(self):
        return self.month

    def get_day(self):
        return self.day

    def __eq__(self, other):
        return (self.year, self.month, self.day) == (other.year, other.month, other.day)

    def __lt__(self, other):
        return (self.year, self.month, self.day) < (other.year, other.month, other.day)

    def __gt__(self, other):
        return (self.year, self.month, self.day) > (other.year, other.month, other.day)

    def days_between(self, other):
        date1 = self._to_days()
        date2 = other._to_days()
        return abs(date2 - date1)

    def _to_days(self):
```

```
        days = self.day
        for m in range(1, self.month):
            days += self._days_in_month(self.year, m)
        return days + self.year * 365 + (self.year // 4 - self.year // 100 + self.year // 400)

if __name__ == '__main__':
    print("Создаем первую дату:")
    date1 = Date()
    date1.read()

    print(f"\nГод {date1.year} - {'високосный' if date1.is_leap() else 'не високосный'}")

    print("\nОперации с днями:")
    days_to_add = int(input("Сколько дней добавить к первой дате? "))
    new_date = date1.add_days(days_to_add)
    print(f"Дата после добавления {days_to_add} дней: ", end="")
    new_date.display()

    days_to_subtract = int(input("Сколько дней вычесть из первой даты? "))
    new_date2 = date1.subtract_days(days_to_subtract)
    print(f"Дата после вычитания {days_to_subtract} дней: ", end="")
    new_date2.display()

    print("\nСоздаем вторую дату для сравнения:")
    date2 = Date()
    date2.read()

    print(f"\nГод {date2.year} - {'високосный' if date2.is_leap() else 'не високосный'}")

    print("\nСравнение дат:")
    print("Дата 1: ", end="")
    date1.display()
    print("Дата 2: ", end="")
    date2.display()

    if date1 == date2:
        print("Даты равны")
    elif date1 < date2:
        print("Дата 1 раньше даты 2")
    else:
        print("Дата 1 позже даты 2")

    days_diff = date1.days_between(date2)
    print(f"Количество дней между датами: {days_diff}")

    print("\nИзменение частей даты:")
    print("Текущая дата 1: ", end="")
    date1.display()

    change = input("Хотите изменить год, месяц или день? (г/м/д/нет): ").lower()

    if change == 'г':
```

```
    new_year = int(input("Новый год: "))
    date1.set_date(year=new_year)
elif change == 'м':
    new_month = int(input("Новый месяц (1-12): "))
    date1.set_date(month=new_month)
elif change == 'д':
    new_day = int(input("Новый день: "))
    date1.set_date(day=new_day)

print("Дата после изменения: ", end="")
date1.display()
```

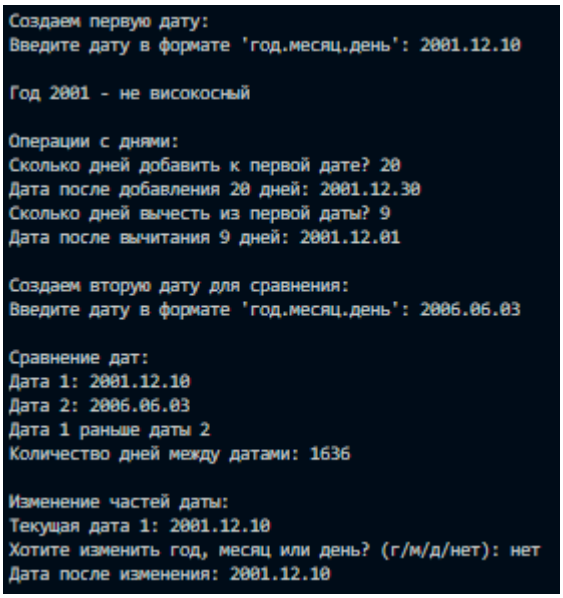

Рисунок 2. Пример работы кода

Контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Классы объявляются с помощью ключевого слова class и имени класса:

class syntax

class MyClass:

var = ... # некоторая переменная

def do_smt(self):

какой-то метод

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибут класса - это атрибут, общий для всех экземпляров класса. Атрибуты класса определены внутри класса, но вне каких-либо методов. Их значения одинаковы для всех экземпляров этого класса. Так что вы можете

рассматривать их как тип значений по умолчанию для всех наших объектов. Атрибуты экземпляра определяются в методах и хранят информацию, специфичную для экземпляра.

3. Каково назначение методов класса?

Классы позволяют определять данные и поведение похожих объектов. Поведение описывается методами. Метод похож на функцию тем, что это блок кода, который имеет имя и выполняет определенное действие. Методы, однако, не являются независимыми, поскольку они определены внутрикласса.

4. Для чего предназначен метод init () класса?

Метод init является конструктором. Конструкторы - это концепция объектно-ориентированного программирования. Класс может иметь один и только один конструктор. Если init определен внутри класса, он автоматически вызывается при создании нового экземпляра класса. Метод init указывает, какие атрибуты будут у экземпляров нашего класса.

5. Каково назначение self?

Аргумент self представляет конкретный экземпляр класса и позволяет нам получить доступ к его атрибутам и методам. Важно использовать параметр self внутри метода, если мы хотим сохранить значения экземпляра для последующего использования. В большинстве случаев нам также необходимо использовать параметр self в других методах, потому что при вызове метода первым аргументом, который ему передается, является сам объект.

6. Как добавить атрибуты в класс?

Например, мы хотим видеть информацию о всех видах наших питомцев. Мы могли бы записать ее в самом классе с самого начала или создать переменную следующим образом:

Pet.all_specs = [tom.spec, avocado.spec, ben.spec]

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

Хорошим тоном считается, что для чтения/изменения какого-то атрибута должны использоваться специальные методы, которые называются getter/setter, их можно реализовать, но ничего не помешает изменить атрибут напрямую. При этом есть соглашение, что метод или атрибут, который начинается с нижнего подчеркивания, является скрытым, и снаружи класса трогать его не нужно (хотя сделать это можно).

8. 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.

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.