Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №29
дисциплины «Основы программной инженерии»

Выполнил:
Плугатырев Владислав Алексеевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Классы данных в Python.

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

Ход работы.

1. Создание репозитория.

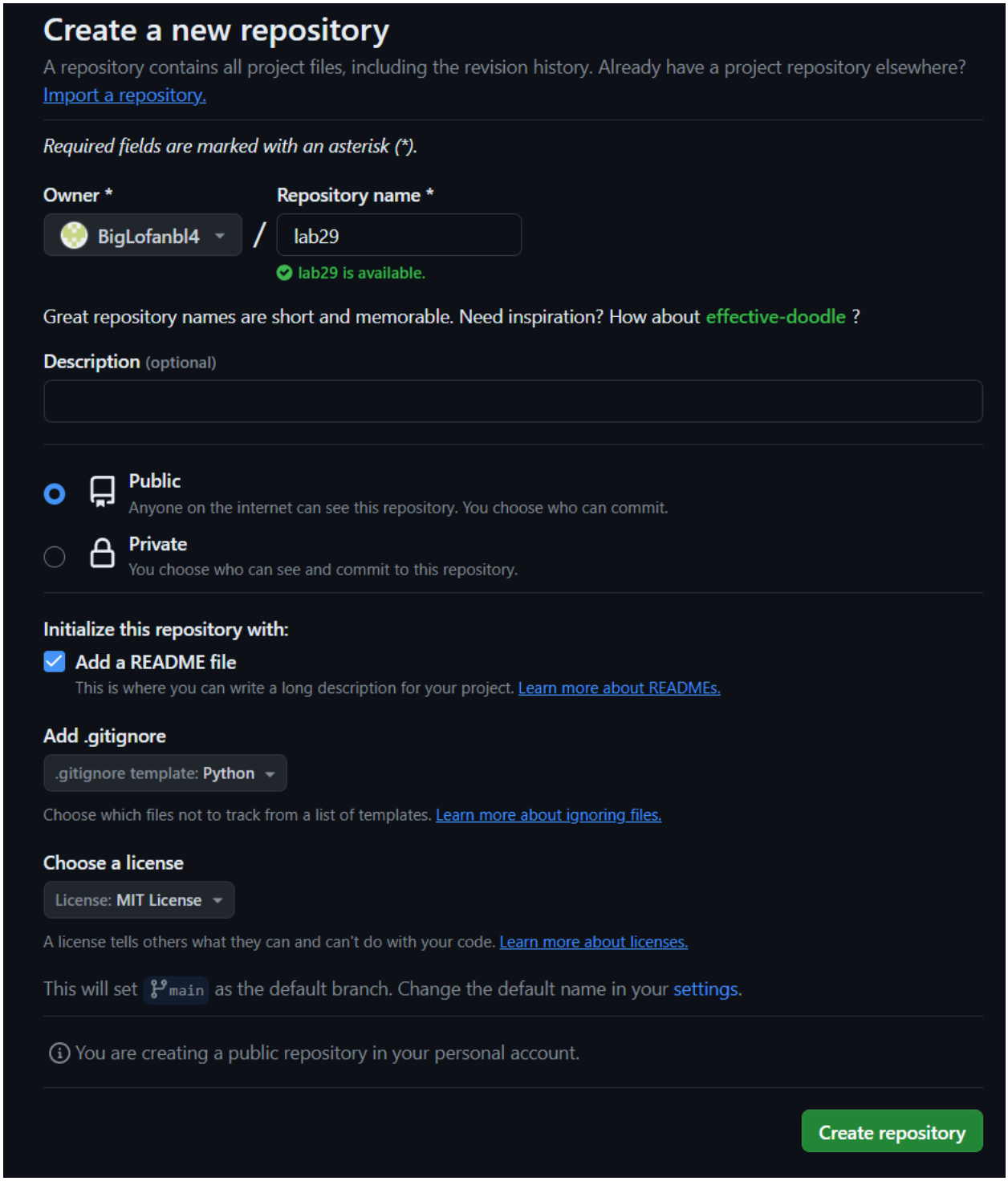

Рисунок 1.1 – Создание репозитория

2. Выполнение примеров из лабораторной работы.

```
from dataclasses import dataclass, field
from datetime import date
import sys
from typing import List
import xml.etree.ElementTree as ET


@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int


@dataclass
class Staff:
    workers: List[Worker] = field(default_factory=lambda: [])

    def add(self, name, post, year):
        self.workers.append(Worker(name=name, post=post, year=year))
        self.workers.sort(key=lambda worker: worker.name)

    def __str__(self):
        # Заголовок таблицы.
        table = []
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 20, "-" * 8
        )
        table.append(line)
        table.append(
            "| {:^4} | {:^30} | {:^20} | {:^8} |".format(
                "№", "Ф.И.О.", "Должность", "Год"
            )
        )
        table.append(line)
        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(self.workers, 1):
            table.append(
                "| {:>4} | {:<30} | {:<20} | {:>8} |".format(
                    idx, worker.name, worker.post, worker.year
                )
            )
        table.append(line)
        return "\n".join(table)

    def select(self, period):
        # Получить текущую дату.
        today = date.today()
        result = []
        for worker in self.workers:
            if today.year - worker.year >= period:
                result.append(worker)
        return result
```

Рисунок 2.1 – Код примера

3. Выполнение индивидуального задания: Выполнить индивидуальное задание лабораторной работы 4.5, использовав классы данных, а также загрузку и сохранение данных в формат XML.

```
@dataclass(frozen=True)
class Person:
    surname: str
    name: str
    zodiac: str
    birthday: list

@dataclass
class People:
    people: List[dict] = field(default_factory=list)

    def add(self, surname: str, name: str, zodiac: str, birthday: str):
        self.people.append(
            Person(
                surname,
                name,
                zodiac,
                birthday.split(".")
            )
        )

        self.people.sort(
            key=lambda x: datetime.strptime(
                ".".join(x.birthday), "%d.%m.%Y"
            )
        )

    def __str__(self):
        table = []
        line = "+-{}-+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 30, "-" * 20, "-" * 20
        )
        table.append(line)
        table.append(
            "| {:^4} | {:^30} | {:^30} | {:^20} | {:^20} |".format(
                "№", "Фамилия", "Имя", "Знак зодиака", "Дата рождения"
            )
        )
        table.append(line)

        for idx, person in enumerate(self.people, 1):
            print(
                "| {:>4} | {:<30} | {:<30} | {:<20} | {:>20} |".format(
                    idx,
                    person.get("surname", ""),
                    person.get("name", ""),
                    person.get("zodiac", ""),
                    ".".join(person.get("birthday", "")),
                )
            )
        )
```

Рисунок 3.1 – Код программа

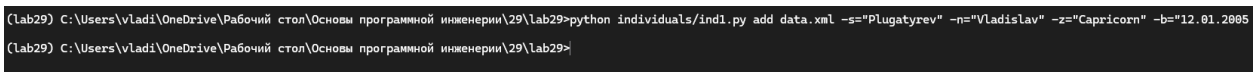

Рисунок 3.2 – Запуск программы

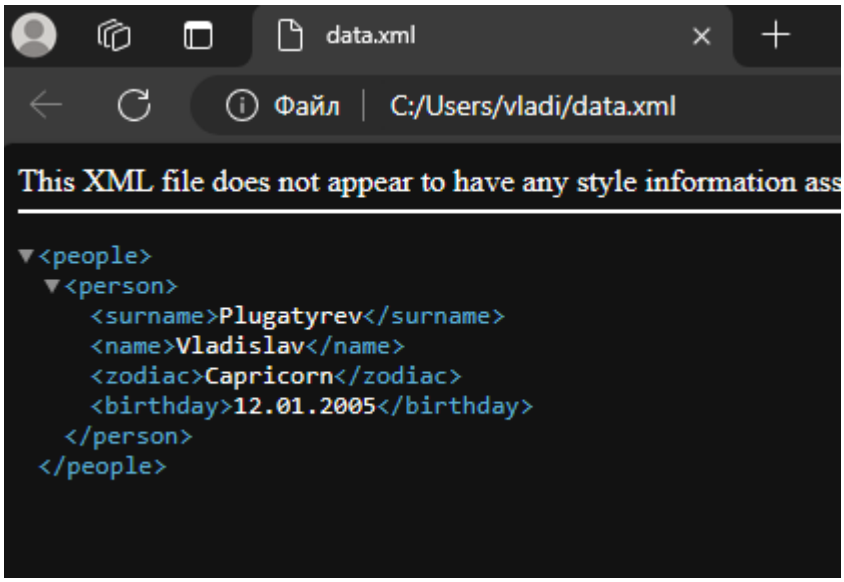

Рисунок 3.3 – XML-файл

Ответы на контрольные вопросы

1. Создание класса данных в Python:

В Python классы данных могут быть легко созданы при помощи декоратора «@dataclass» из модуля «dataclasses», который появился в Python 3.7+. Для создания класса данных достаточно пометить класс декоратором «@dataclass» и определить атрибуты.

2. Методы по умолчанию, реализуемые классом данных:

Класс данных, созданный с помощью декоратора «@dataclass», автоматически реализует несколько методов:

- «__init__»: конструктор класса для инициализации экземпляров,

- «__repr__»: возвращает форматированную строку, которая представляет объект (полезно для отладки),

- «__eq__»: метод сравнения объектов на равенство,

- «__lt__», «__le__», «__gt__», и «__ge__»: методы для сравнения, если параметр «dataclass» «order» установлен в «True»,

- «__hash__»: метод для вычисления хэш-функции, если экземпляр класса данных предполагается использовать в качестве ключа в словарях или элемента в множествах, и если в «@dataclass» установлен флаг «frozen=True».

Некоторые из этих методов можно опустить, установив соответствующие параметры в «@dataclass».

3. Создание неизменяемого класса данных:

Чтобы сделать класс данных неизменяемым (immutable), нужно использовать параметр «frozen» в декораторе «@dataclass».