Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических
систем и электроники института перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Объектно-ориентированное программирование»

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

_____
(подпись)

Доцент кафедры инфокоммуникаций:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Работа с исключениями в языке Python.

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

Ход работы

1. Создание репозитория.

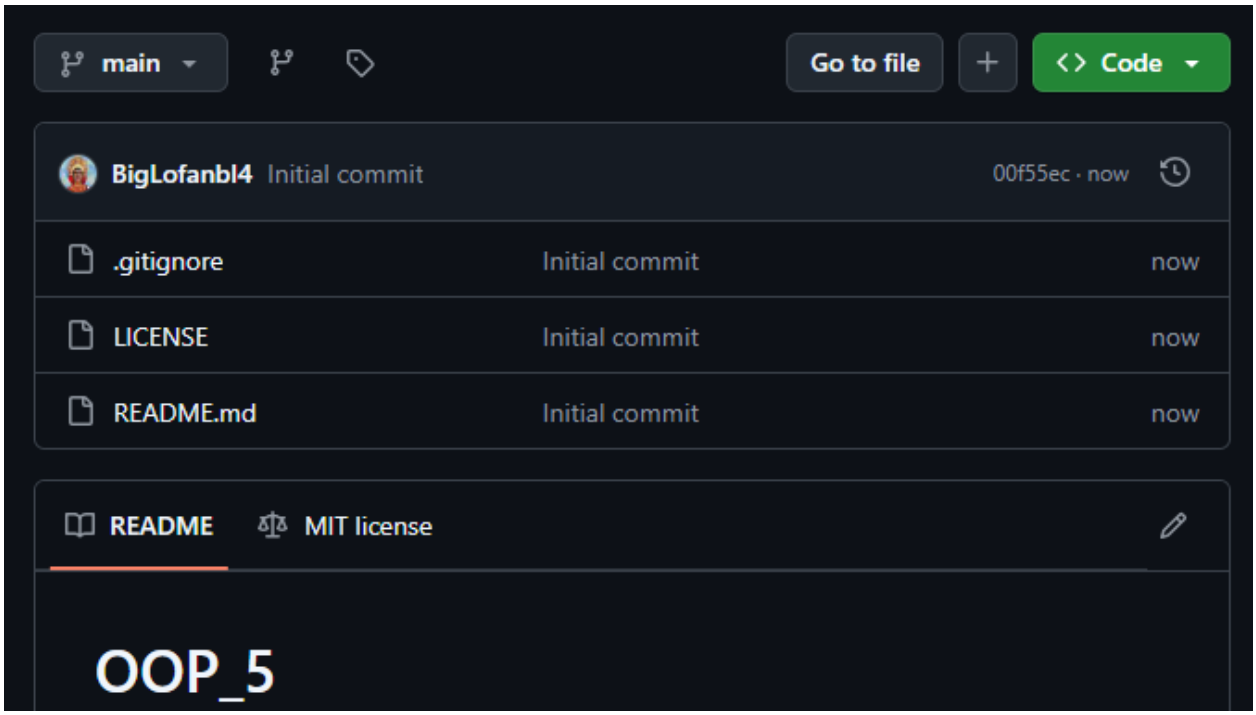

Рисунок 1.1 – Созданный репозиторий

2. Выполнение примеров из лабораторной работы.

```
if __name__ == "__main__":
    # Выполнить настройку логгера.
    logging.basicConfig(filename="workers.log", level=logging.INFO)
    # Список работников.
    staff = Staff()
    # Организовать бесконечный цикл запроса команд.
    while True:
        try:
            # Запросить команду из терминала.
            command = input(">>> ").lower()
            # Выполнить действие в соответствие с командой.
            if command == "exit":
                break
            elif command == "add":
                # Запросить данные о работнике.
                name = input("Фамилия и инициалы? ")
                post = input("Должность? ")
                year = int(input("Год поступления? "))
                # Добавить работника.
                staff.add(name, post, year)
                logging.info(
                    f"Добавлен сотрудник: {name}, {post}, поступивший в {year}"
                    f" году."
                )
            elif command == "list":
                # Вывести список.
                print(staff)
                logging.info("Отображен список сотрудников.")
            elif command.startswith("select "):
                # Разбить команду на части для выделения номера года.
                parts = command.split(maxsplit=1)
                period = int(parts[1])
                # Запросить работников.
                selected = staff.select(period)
                # Вывести результаты запроса.
                if selected:
                    for idx, worker in enumerate(selected, 1):
                        print("{:>4}: {}".format(idx, worker.name))
                    logging.info(
                        f"Найдено {len(selected)} работников со стажем более"
                        f" {parts[1]} лет."
                    )
                else:
                    print("Работники с заданным стажем не найдены.")
                    logging.warning(
                        f"Работники со стажем более {parts[1]} лет не найдены."
                    )
            elif command.startswith("load "):
                # Разбить команду на части для имени файла.
```

Рисунок 1.2 – Код примера

3. Выполнение первого задания.

```python
if __name__ == "__main__":
    num_1, num_2 = input("num_1: "), input("num_2: ")
    try:
        num_1 = int(num_1)
        num_2 = int(num_2)
        print(f"{num_1 + num_2}")
    except ValueError:
        print(f"{num_1 + num_2}")
```

Рисунок 1.3 – Код программы

```
num_1: f
num_2: 3
f3
```

Рисунок 1.4 – Вывод программы

4. Выполнение второго задания.

```python
from random import randint


if __name__ == "__main__":
    rows = input("rows: ")
    columns = input("columns: ")
    min_val = input("min_val: ")
    max_val = input("max_val: ")

    try:
        rows = int(rows)
        columns = int(columns)
        min_val = int(min_val)
        max_val = int(max_val)

        if rows <= 0 or columns <= 0:
            raise ValueError("Rows & Colums must be > 0!")

        if min_val >= max_val:
            raise ValueError("min_val must be < max_val!")
    except ValueError as e:
        print(e)
        exit(1)

    # matrix = []
    # for i in range(rows):
    #     row = []
    #     for j in range(columns):
    #         num = randint(min_val, max_val)
    #         row.append(num)
    #     matrix.append(row)

    matrix = [
        [randint(min_val, max_val) for _ in range(columns)] for _ in range(rows)
    ]
    print("Generated matrix:")
    for row in matrix:
        print(row)
```

Рисунок 1.5 – Код программы

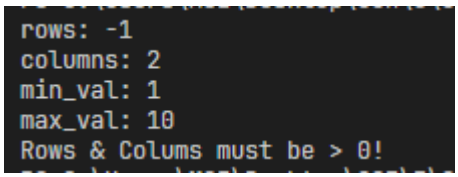

Рисунок 1.6 – Вывод

5. Выполнение первого индивидуального задания: выполнить индивидуальное задание 1 лабораторной работы 2.19, добавив возможность работы с исключениями и логгирование.

```
    match args.command:
        case "add":
            try:
                people.add(args.surname, args.name, args.zodiac, args.birthday)
                is_dirty = True
                logger.info(
                    f"Добавлен человек: {args.surname}, {args.name}"
                    f"Зодиак: {args.zodiac}"
                    f"День рождения: {args.birthday}"
                )
            except Exception as e:
                logger.error(f"Ошибка: {e}")

        case "select":
            selected = people.select(args.surname, people)
            if selected.people:
                print(selected)
                logger(
                    f"Найдено {len(selected.people)} людей с"
                    f"фамилией {args.surname}"
                )
            else:
                print("Люди с заданной фамилией не найдены")
                logger.warning(f"Люди с фамилией {args.surname} не найдены")

        case "display":
            print(people)
            logger.info("Отображен список сотрудников")

        case _:
            logger.error(f"Введена неверная команда: {args.command}")

    if is_dirty:
        people.save(home_path)
        logger.info(f"Сохранены данные в файл {home_path}.")
```

Рисунок 1.7 – Код программы

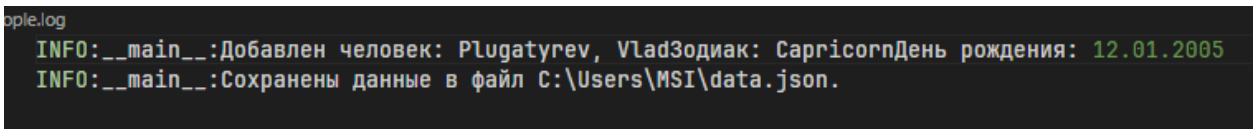

Рисунок 1.8 – Полученный лог

6. Выполнение второго индивидуального задания: изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды

```
match args.command:
    case "add":
        try:
            people.add(args.surname, args.name, args.zodiac, args.birthday)
            is_dirty = True
            logger.info(
                f"{datetime.now()} Добавлен человек: {args.surname}, {args.
                f" Зодиак: {args.zodiac}"
                f" День рождения: {args.birthday}"
            )
        except Exception as e:
            logger.error(f"{datetime.now()} Ошибка: {e}")

    case "select":
        selected = people.select(args.surname, people)
        if selected.people:
            print(selected)
            logger.info(
                f"{datetime.now()} Найдено {len(selected.people)} людей с"
                f"фамилией {args.surname}"
            )
        else:
            print("Люди с заданной фамилией не найдены")
            logger.warning(
                f"{datetime.now()} Люди с фамилией {args.surname} не найден
            )

    case "display":
        print(people)
        logger.info(f"{datetime.now()} Отображен список сотрудников")

    case _:
        logger.error(
            f"{datetime.now()} Введена неверная команда: {args.command}"
        )

if is_dirty:
    people.save(home_path)
    logger.info(f"{datetime.now()} Сохранены данные в файл {home_path}.")
```

Рисунок 1.9 – Код программы

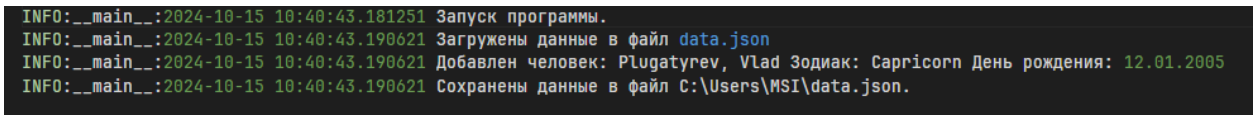

Рисунок 2.1 – Полученный лог

Ответы на контрольные вопросы

1. Виды ошибок в Python: - Синтаксические ошибки («SyntaxError»): возникают при нарушении правил синтаксиса языка. - Ошибки времени выполнения («RuntimeError»): возникают во время выполнения программы, например: - «IndexError»: доступ за пределы индексов списка. - «KeyError»: отсутствующий ключ в словаре. - «TypeError»: операция применена к объекту неподходящего типа. - «ValueError»: операция применена к объекту с подходящим типом, но неправильным значением. - «NameError»: обращение к неопределенной переменной. - Исключения из-за ошибок ввода/вывода («IOError», «FileNotFoundError» и прочие).

2. Обработка исключений: В Python обработка исключений осуществляется с помощью блоков «try» и «except». Блок «try» содержит код, который может вызывать исключение, а блок(и) «except» обрабатывают определенные исключения, если они возникают:

3. Блоки finally и else: - «finally»: блок кода, который выполняется в любом случае, независимо от того, возникло исключение или нет. Обычно используется для закрытия файлов или освобождения ресурсов. - «else»: блок кода, который выполняется если в блоке «try» не возникло исключений. Часто используется для кода, который должен выполняться только тогда, когда «try» блок успешен.

4. Генерация исключений: Исключения можно генерировать с помощью инструкции «raise», указав тип исключения и, optionalno сообщение.

5. Создание пользовательских исключений: Пользовательские исключения создаются путем определения нового класса, который наследуется от встроенного класса исключений, например, от «Exception».

6. Модуль logging: Модуль «logging» используется для журналирования сообщений. Он обеспечивает удобный способ записи информации о том, что происходит в программа: помогает отслеживать события, отлаживать код и диагностировать проблемы.

7. Уровни логгирования: - «DEBUG»: Низкое уровень важности, используется для диагностики проблем. - «INFO»: Общая информация о нормальном функционировании программы. - «WARNING»: Предупреждение о небольших проблемах, которые не остановили выполнение программы. - «ERROR»: Ошибка, из-за которой программа не смогла выполнить какую-то функцию. - «CRITICAL»: Серьезная ошибка, указывающая на возможный крах программы Примеры использования: - «DEBUG» может использоваться для вывода переменных на определенном этапе выполнения или для данного состояния. - «INFO» мог бы использоваться для подтверждения, что определенная задача или этап выполнены. - «WARNING» мог бы предупредить о конфигурации, которая не идеальна, но работает. - «ERROR» мог бы указывать на невозможность чтения файла. - «CRITICAL» мог бы сообщать о потере соединения с базой данных или о недоступности критически важного ресурса.