

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 №27
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 _____

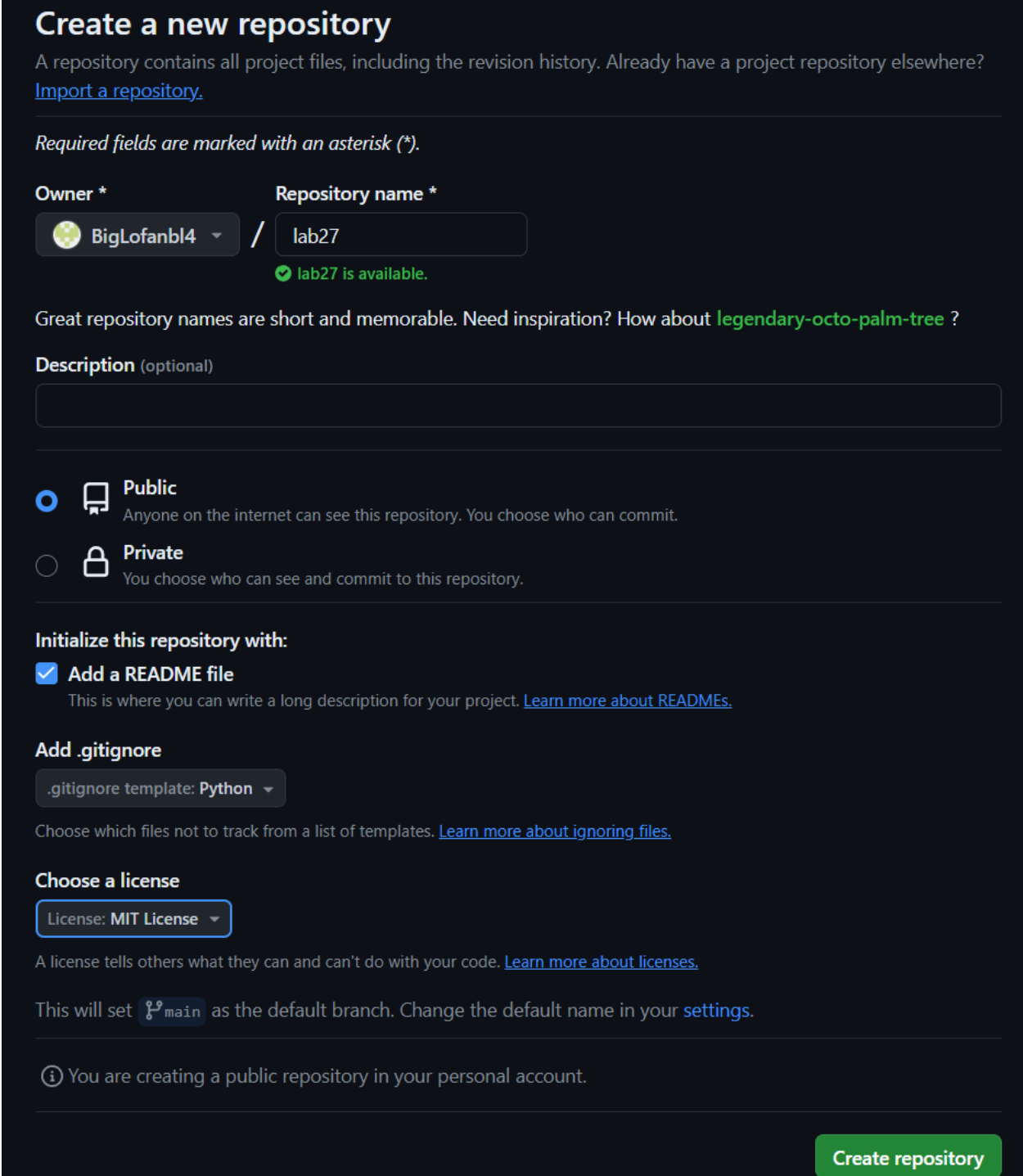
Ставрополь, 2024 г.

Тема: 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.




Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*


Owner * **Repository name ***


 BigLofanbl4 / lab27

✔ lab27 is available.

Great repository names are short and memorable. Need inspiration? How about **legendary-octo-palm-tree** ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **Python**


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **MIT License**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1.1 – Создание репозитория

2. Выполнение примеров из лабораторной работы.

```
2
3 @dataclass
4 class Staff:
5     workers: List[Worker] = field(default_factory=lambda: [])
6
7     def add(self, name, post, year):
8         # Получить текущую дату.
9         today = date.today()
10        if year < 0 or year > today.year:
11            raise IllegalYearError(year)
12        self.workers.append(Worker(name=name, post=post, year=year))
13        self.workers.sort(key=lambda worker: worker.name)
14
15    def __str__(self):
16        # Заголовок таблицы.
17        table = []
18        line = "+-{}-+-{}-+-{}-+-{}-+".format(
19            "-" * 4, "-" * 30, "-" * 20, "-" * 8
20        )
21        table.append(line)
22        table.append(
23            "| {:^4} | {:^30} | {:^20} | {:^8} |".format(
24                "No", "Ф.И.О.", "Должность", "Год"
25            )
26        )
27        table.append(line)
28        # Вывести данные о всех сотрудниках.
29        for idx, worker in enumerate(self.workers, 1):
30            table.append(
31                "| {:>4} | {:<30} | {:<20} | {:>8} |".format(
32                    idx, worker.name, worker.post, worker.year
33                )
34            )
35            table.append(line)
36        return "\n".join(table)
37
38    def select(self, period):
39        # Получить текущую дату.
40        today = date.today()
41        result = []
42        for worker in self.workers:
43            if today.year - worker.year >= period:
44                result.append(worker)
45        return result
46
```

Рисунок 2.1 – Код примера

3. Выполнение первого задания.

```
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 3.1 – Код задания

```
num_1: a
num_2: 3
a3
```

Рисунок 3.2 – Вывод программы

4. Выполнение второго задания.

```
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 & Cols 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 4.1 – Код задания

```
rows: 2
columns: 2
min_val: 1
max_val: 5
Generated matrix:
[4, 4]
[4, 4]
```

Рисунок 4.2 – Вывод программы

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

```
is_dirty = False
if home_path.exists():
    try:
        people.load(home_path)
        logger.info(f"Загружены данные в файл {args.filename}")
    except ValidationError:
        logger.error("Ошибка валидации при загрузке данных из файла.")

match args.command:
    case "add":
        (variable) people: People
        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")

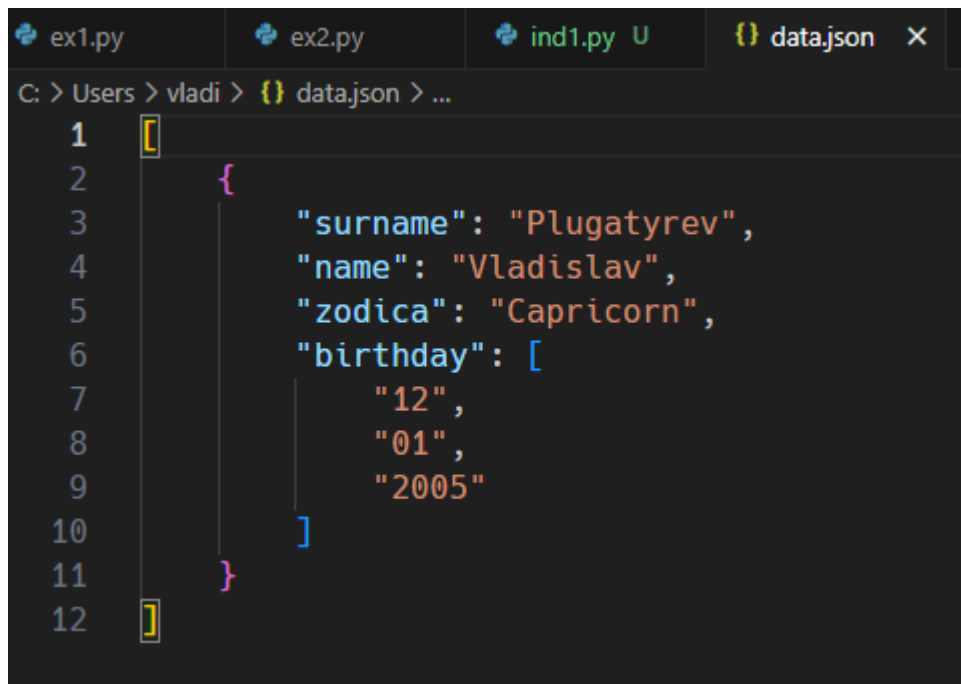
    case _:
        logger.error(f"Введена неверная команда: {args.command}")

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

Рисунок 4.1 – Код программы

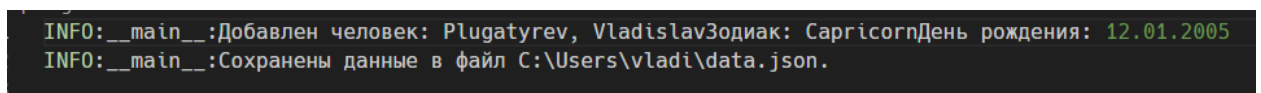
```
python individuals/ind1.py add data.json -s="Plugatyrev" -n="Vladislav" -z="Capricorn" -b="12.01.2005"
```

Рисунок 4.2 – Запуск программы



```
1  [
2      {
3          "surname": "Plugatyrev",
4          "name": "Vladislav",
5          "zodica": "Capricorn",
6          "birthday": [
7              "12",
8              "01",
9              "2005"
10         ]
11     }
12 ]
```

Рисунок 4.3 – Данные в файле



```
INFO:__main__:Добавлен человек: Plugatyrev, VladislavЗодиак: CapricornДень рождения: 12.01.2005
INFO:__main__:Сохранены данные в файл C:\Users\vladi\data.json.
```

Рисунок 4.4 – Лог

5. 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.

```
is_dirty = False
if home_path.exists():
    try:
        people.load(home_path)
        logger.info(f"{datetime.now()} Загружены данные в файл {args.filename}")
    except ValidationError:
        logger.error(f"{datetime.now()} Ошибка валидации при загрузке данных из файла JSON.")

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.name}"
                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 найдены"
            )

    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 5.1 – Код программы

```
>python individuals/ind2.py add data.json -s="Plugatyrev1" -n="Vladislav1" -z="Capricorn1" -b="13.01.2005"
```

Рисунок 5.2 – Запуск программы

```
1  [
2      {
3          "surname": "Plugatyrev",
4          "name": "Vladislav",
5          "zodica": "Capricorn",
6          "birthday": [
7              "12",
8              "01",
9              "2005"
10         ]
11     },
12     {
13         "surname": "Plugatyrev1",
14         "name": "Vladislav1",
15         "zodica": "Capricorn1",
16         "birthday": [
17             "13",
18             "01",
19             "2005"
20         ]
21     }
22 ]
```

Рисунок 5.3 – Файл с данными

```
INFO: __main__:2024-04-29 16:59:12.470116 Запуск программы.  
INFO: __main__:2024-04-29 16:59:12.475152 Загружены данные в файл data.json  
INFO: __main__:2024-04-29 16:59:12.475152 Добавлен человек: Plugatyrev1, Vladislav1 Зодиак: Capricorn1 День рождения: 13.01.2005  
INFO: __main__:2024-04-29 16:59:12.478244 Сохранены данные в файл C:\Users\vladi\data.json.
```

Рисунок 5.4 – 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, optionally 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 программе: помогает отслеживать события, отлаживать код и диагностировать проблемы.

7. Уровни логгирования:

- «DEBUG»: Низкий 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.