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 №4
дисциплины «Объектно-ориентированное программирование»
Вариант___

Выполнил:
Епифанов Алексей Александрович
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 Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Работа с исключениями в языке Python

Цель: приобретение навыков по работе с исключениями 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 гитхаб: https://github.com/alexeiepif/OOP_4.git

2. Проработал пример лабораторной:

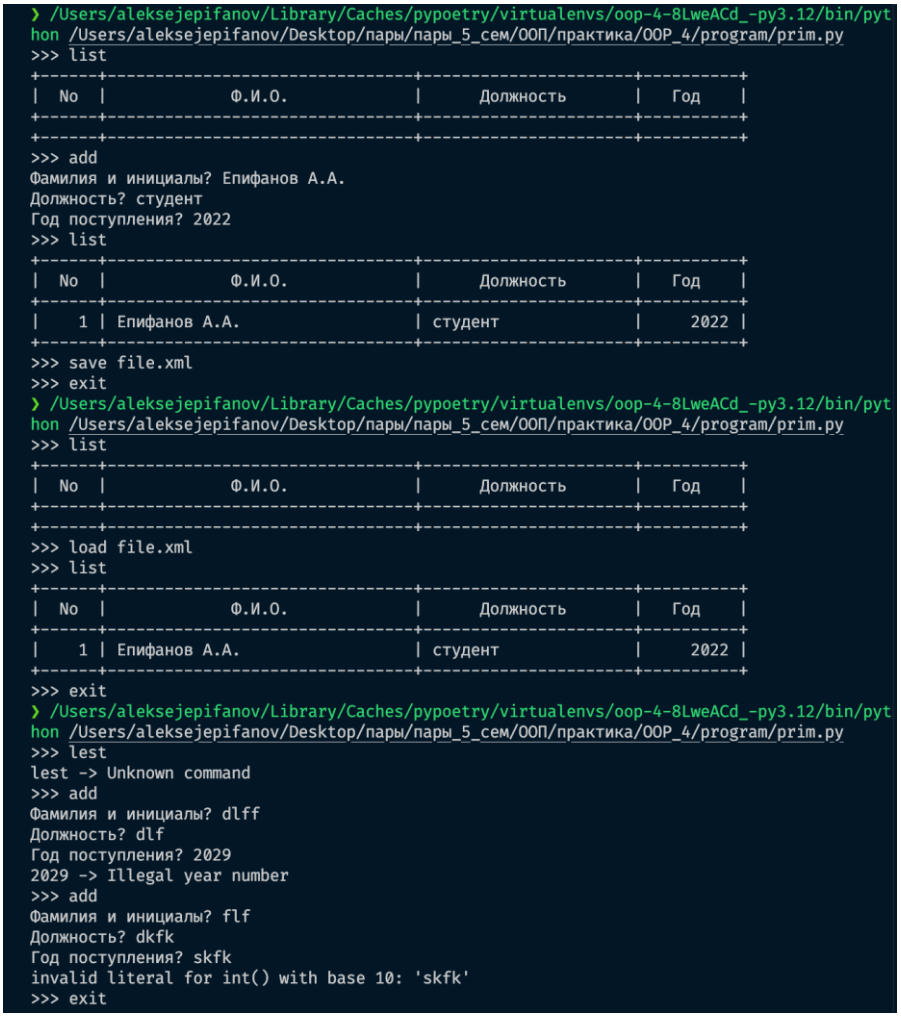

Рисунок 1. Работа примера

```
INFO:root:Отображен список сотрудников.
INFO:root:Добавлен сотрудник: epif a a, grep, поступивший в 2003 году.
INFO:root:Отображен список сотрудников.
ERROR:root:Ошибка: invalid literal for int() with base 10: 'dfl'
INFO:root:Отображен список сотрудников.
INFO:root:Отображен список сотрудников.
ERROR:root:Ошибка: 2025 -> Illegal year number
INFO:root:Отображен список сотрудников.
INFO:root:Добавлен сотрудник: Епифанов A.A., студент, поступивший в 2022 году.
INFO:root:Отображен список сотрудников.
INFO:root:Сохранены данные в файл file.xml.
INFO:root:Отображен список сотрудников.
INFO:root:Загружены данные из файла file.xml.
INFO:root:Отображен список сотрудников.
ERROR:root:Ошибка: lest -> Unknown command
ERROR:root:Ошибка: 2029 -> Illegal year number
ERROR:root:Ошибка: invalid literal for int() with base 10: 'skfk'
```

Рисунок 2. Файл workers.log

3. Выполнил общее задание 1: Разработайте программу по следующему описанию.

Напишите программу, которая запрашивает ввод двух значений. Если хотя бы одно из них не является числом, то должна выполняться конкатенация, т. е. соединение, строк. В остальных случаях введенные числа суммируются.

```
❯ p task_1
Введите первое число: 9
Введите второе число: ki
Результат: 9ki
❯ p task_1
Введите первое число: 9
Введите второе число: 8
Результат: 17
```

Рисунок 3. Результат работы программы общего задания

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


def main() -> None:
    result: int | str
    try:
        a = input("Введите первое число: ")
        b = input("Введите второе число: ")
        c = int(a)
```

```
        d = int(b)
        result = c + d
    except ValueError:
        result = f"{a}{b}"
    finally:
        print(f"Результат: {result}")


if __name__ == "__main__":
    main()
```

4. Выполнил общее задание 2: напишите программу, которая будет генерировать матрицу из случайных целых чисел. Пользователь может указать число строк и столбцов, а также диапазон целых чисел. Произведите обработку ошибок ввода пользователя.

```
❯ p task_2
Введите количество строк: 8
Введите количество столбцов: 5
Введите начало диапазона: 7
Введите конец диапазона: 89
|       12      58      86      75      52      |
|       40      21      71      23      47      |
|       35      77      71      68      57      |
|       56      87      73      58      78      |
|       51      47      38      21      35      |
|       43      67      18      52      50      |
|       78      72      24      24      37      |
|       77      89      29      53      8       |

❯ p task_2
Введите количество строк: 0
Введите количество столбцов: 1
Введите начало диапазона: 7
Введите конец диапазона: 8
Ошибка:  Значение не является положительным: rows = 0 (ожидалось > 0)
❯ p task_2
Введите количество строк: 6
Введите количество столбцов: 8
Введите начало диапазона: 9
Введите конец диапазона: 0
Ошибка:  Начало диапазона больше конца: 9 > 0
```

Рисунок 4. Результат работы программы общего задания

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


import random
from typing import Generator
```

```
class Matrix:
    names = ["rows", "columns"]

    def __init__(self, rows: int, columns: int, start: int, end: int) -> None:
        self.rows = rows
        self.columns = columns
        self.start = start
        self.end = end

    def generate_matrix(self) -> None:
        for name, value in self.items():
            if value <= 0:
                raise NumberNotPositiveError(name, value)

        if self.start > self.end:
            raise StartGreaterThanEndError(self.start, self.end)

        self.matrix = [
            [random.randint(self.start, self.end) for _ in range(self.columns)]
            for _ in range(self.rows)
        ]

    def items(self) -> Generator[tuple[str, int], None, None]:
        for name in Matrix.names:
            yield name, getattr(self, name)

    def __str__(self) -> str:
        if not self.matrix:
            return "Матрица пока не сгенерирована"
        string = ""
        for row in self.matrix:
            string += "|\t" + "\t".join(map(str, row)) + "\t|\n"
        return string


class StartGreaterThanEndError(Exception):
    def __init__(
        self,
        start: int,
        end: int,
        message: str = "Начало диапазона больше конца",
    ) -> None:
        self.start = start
        self.end = end
        self.message = message
        super(StartGreaterThanEndError, self).__init__(message)

    def __str__(self) -> str:
        return f"{self.message}: {self.start} > {self.end}"


class NumberNotPositiveError(Exception):
    def __init__(
        self,
        name: str,
        number: int,
        message: str = "Значение не является положительным",
    ):
        self.name = name
        self.number = number
        self.message = message
```

```
        super(NumberNotPositiveError, self).__init__(message)

    def __str__(self) -> str:
        return f"{self.message}: {self.name} = {self.number} (ожидалось > 0)"


def main() -> None:
    try:
        matrix = Matrix(
            int(input("Введите количество строк: ")),
            int(input("Введите количество столбцов: ")),
            int(input("Введите начало диапазона: ")),
            int(input("Введите конец диапазона: ")),
        )
        matrix.generate_matrix()
        print(matrix)

    except Exception as e:
        print("Ошибка: ", e)


if __name__ == "__main__":
    main()
```

5. Выполнил индивидуальные задания 1 и 2 вариант 9: Выполнить индивидуальное задание 1 лабораторной работы 2.19, добавив возможность работы с исключениями и логирование. Изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды.

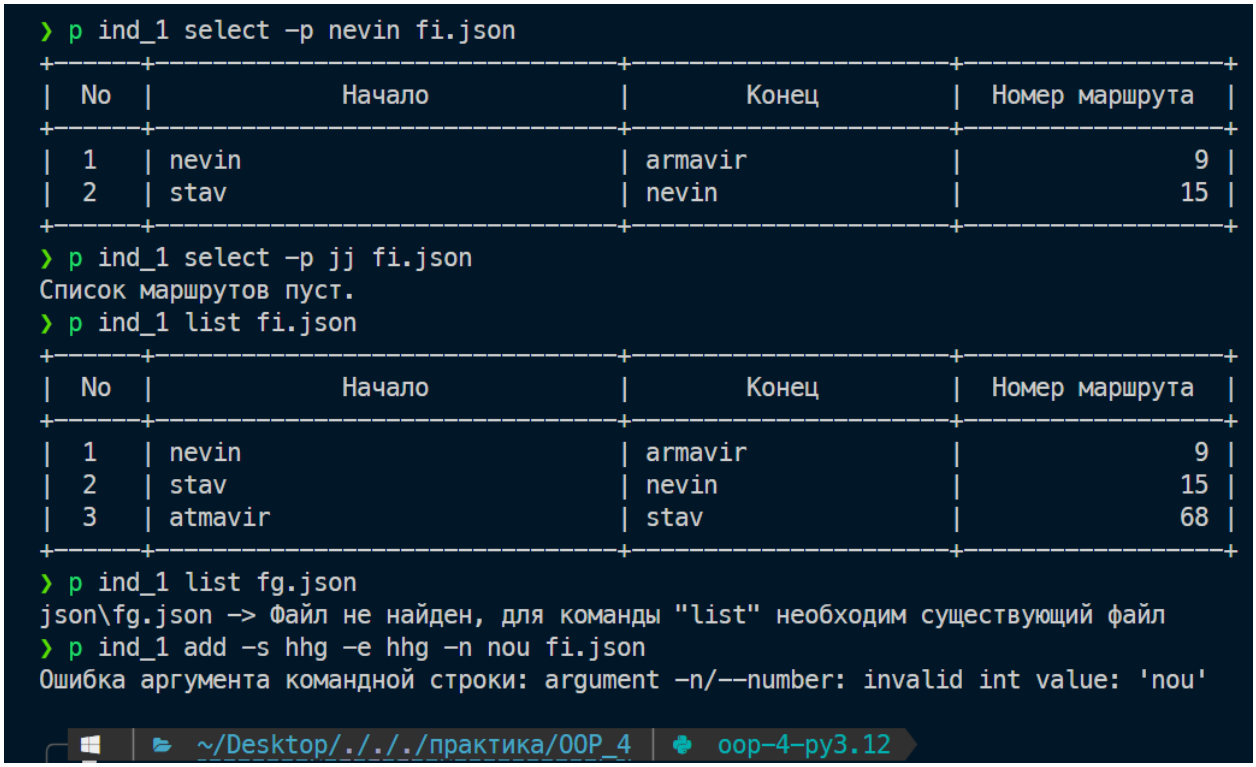

Рисунок 5. Результат работы программы индивидуального задания

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-


import argparse
import bisect
import json
import logging
import os
import sys
import traceback
from dataclasses import asdict, dataclass, field
from pathlib import Path
from typing import List, NoReturn

from jsonschema import ValidationError, validate


class CustomArgumentParser(argparse.ArgumentParser):
    def error(self, message: str) -> NoReturn:
        raise argparse.ArgumentError(None, message)


# Класс пользовательского исключения в случае, если введенный маршрут
# уже существует.
class RouteExistsError(Exception):
    def __init__(
        self, route: "Route", message: str = "Route already exists"
    ) -> None:
        self.route = route
        self.message = message
        super(RouteExistsError, self).__init__(message)

    def __str__(self) -> str:
        return f"{self.route} -> {self.message}"


# Класс пользовательского исключения в случае, если для команд
# вывода маршрутов использовался несуществующий файл
class FileNotExistsError(Exception):
    def __init__(
        self, file_path: Path, message: str = "File not exists"
    ) -> None:
        self.file_path = file_path
        self.message = message
        super(FileNotExistsError, self).__init__(message)

    def __str__(self) -> str:
        return f"{self.file_path} -> {self.message}"


@dataclass(frozen=True)
class Route:
    start: str
    end: str
    number: int


@dataclass
class Routes:
    routes: List[Route] = field(default_factory=list)

    def add(self, start: str, end: str, number: int) -> None:
        """
```

```
    Добавить данные о маршруте.
    """
    route = Route(
        start.lower(),
        end.lower(),
        number,
    )
    if route not in self.routes:
        bisect.insort(
            self.routes,
            route,
            key=lambda item: item.number,
        )
    else:
        raise RouteExistsError(route)

def __str__(self) -> str:
    """
    Отобразить список маршрутов.
    """
    if self.routes:
        table = []
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 20, "-" * 16
        )
        table.append(line)
        table.append(
            "| {:^4} | {:^30} | {:^20} | {:^16} |".format(
                "No", "Начало", "Конец", "Номер маршрута"
            )
        )
        table.append(line)
        for idx, route in enumerate(self.routes, 1):
            table.append(
                "| {:^4} | {:<30} | {:<20} | {:>16} |".format(
                    idx, route.start, route.end, route.number
                )
            )
        table.append(line)
        return "\n".join(table)
    else:
        return "Список маршрутов пуст."

def __len__(self) -> int:
    return len(self.routes)

def select(self, name_point: str) -> "Routes":
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    selected: List[Route] = []
    for route in self.routes:
        if route.start == name_point or route.end == name_point:
            selected.append(route)

    return Routes(selected)

def save(self, file_path: Path) -> None:
    """
    Сохранить все маршруты в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with file_path.open("w", encoding="utf-8") as file_out:
```

```
            data_with_type = [
                {"__type__": route.__class__.__name__, **asdict(route)}
                for route in self.routes
            ]
            # Записать данные из словаря в формат JSON и сохранить их
            # в открытый файл.
            json.dump(data_with_type, file_out, ensure_ascii=False, indent=4)

    def load(self, file_path: Path) -> None:
        """
        Загрузить все маршруты из файла JSON.
        """
        schema = {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "__type__": {"type": "string", "enum": ["Route"]},
                    "start": {"type": "string"},
                    "end": {"type": "string"},
                    "number": {"type": "integer"},
                },
                "required": [
                    "__type__",
                    "start",
                    "end",
                    "number",
                ],
            },
        }
        # Открыть файл с заданным именем и прочитать его содержимое.
        with file_path.open("r", encoding="utf-8") as file_in:
            data = json.load(file_in)  # Прочитать данные из файла

        validate(instance=data, schema=schema)
        for route in data:
            route.pop("__type__", None)
            self.routes.append(Route(**route))


def main(command_line: str | None = None) -> None:
    """
    Главная функция программы.
    """
    logging.basicConfig(
        filename="app.log",
        filemode="a",
        encoding="utf-8",
        format="%(asctime)s.%(msecs)03d - %(levelname)s - %(message)s",
        level=logging.INFO,
    )
    file_parser = CustomArgumentParser(add_help=False)
    file_parser.add_argument(
        "--home",
        action="store_true",
        help="Save the file in the user's home directory",
    )
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )
    parser = CustomArgumentParser("routes")
    parser.add_argument(
        "--version", action="version", version="%(prog)s 0.2.0"
```

```
)
subparsers = parser.add_subparsers(dest="command")
add = subparsers.add_parser(
    "add", parents=[file_parser], help="Add a new route"
)
add.add_argument(
    "-s", "--start", action="store", required=True, help="The route start"
)
add.add_argument(
    "-e", "--end", action="store", required=True, help="The route endpoint"
)
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="The number of route",
)

_ = subparsers.add_parser(
    "list", parents=[file_parser], help="Display all routes"
)

select = subparsers.add_parser(
    "select", parents=[file_parser], help="Select the routes"
)
select.add_argument(
    "-p",
    "--point",
    action="store",
    required=True,
    help="Routes starting or ending at this point",
)

args = parser.parse_args(command_line)

# Загрузить всех работников из файла, если файл существует.
is_dirty = False
routes = Routes()
if args.home:
    filepath: Path = Path.home() / args.filename
else:
    filepath = Path("json") / args.filename

if os.path.exists(filepath):
    routes.load(filepath)
    logging.info(f"Загружены маршруты из файла {filepath}")
elif args.command.lower() in ("list", "select"):
    raise FileNotExistsError(
        filepath,
        f'Файл не найден, для команды "{args.command.lower()}" '
        "необходим существующий файл",
    )
else:
    logging.info(
        f"Файл {filepath} не найден, будет создан при сохранении."
    )

match args.command.lower():
    case "add":
        routes.add(args.start, args.end, args.number)
        is_dirty = True
```

```
            logging.info(
                f"Добавлен маршрут: {args.start} -> {args.end} ({args.number})"
            )

        case "list":
            print(routes)
            logging.info("Выведены все маршруты")

        case "select":
            name_point = args.point.lower()
            selected = routes.select(name_point)
            print(selected)
            if selected:
                logging.info(
                    f"Найдено {len(selected)} маршрутов, "
                    f"начинающихся или заканчивающихся в точке {name_point}"
                )
            else:
                logging.warning(
                    f"Найдено 0 маршрутов, "
                    f"начинающихся или заканчивающихся в точке {name_point}"
                )

    if is_dirty:
        routes.save(filepath)
        logging.info(f"Сохранены маршруты в файл {filepath}")


if __name__ == "__main__":
    try:
        main()
    except ValidationError as exc:
        logging.error(f"Ошибка валидации: {exc}")
        print(f"Ошибка валидации: {exc.message}", file=sys.stderr)
    except argparse.ArgumentError as exc:
        logging.error(f"Ошибка аргумента командной строки: {exc}")
        print(f"Ошибка аргумента командной строки: {exc}", file=sys.stderr)
    except Exception as exc:
        logging.error(f"Ошибка:\n{traceback.format_exc()}")
        print(exc, file=sys.stderr)
```

Ответы на контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

В Python выделяют два различных вида ошибок: синтаксические ошибки и исключения.

2. Как осуществляется обработка исключений в языке программирования Python?

Обработка исключений нужна для того, чтобы приложение не завершалось аварийно каждый раз, когда возникает исключение. Для этого блок кода, в котором возможно появление исключительной ситуации необходимо поместить во внутрь синтаксической конструкции try except.

3. Для чего нужны блоки finally и else при обработке исключений?

Чтобы обеспечить выполнение определенного программного кода при завершении блока try/except, следует использовать оператор finally. Этот оператор гарантирует, что указанный код будет выполнен независимо от того, возникло ли исключение или нет, что позволяет обеспечить надежное завершение процесса.

Конструкция else выполняется в случае, если в коде не произошло исключений.

4. Как осуществляется генерация исключений в языке Python?

Для принудительной генерации исключения используется инструкция raise.

5. Как создаются классы пользовательский исключений в языке Python?

В Python можно создавать собственные исключения. Такая практика позволяет увеличить гибкость процесса обработки ошибок в рамках той предметной области, для которой написана ваша программа.

Для реализации собственного типа исключения необходимо создать класс, являющийся наследником от одного из классов исключений.

6. Каково назначение модуля logging?

Для вывода специальных сообщений, не влияющих на функционирование программы, в Python применяется библиотека логов. Чтобы воспользоваться ею, необходимо выполнить импорт в верхней части файла.

7. Какие уровни логгирования поддерживаются модулем logging? Приведите примеры, в которых могут быть использованы сообщения с этим уровнем журналирования.

DEBUG — для отладки, подробная информация.

INFO — для общих сообщений о ходе работы программы.

WARNING — для предупредительных сообщений о возможных проблемах.

ERROR — для ошибок, которые могут повлиять на работу, но не останавливают программу.

CRITICAL — для самых серьёзных ошибок, ведущих к сбою программы.

Вывод: в ходе выполения работы были приобретены навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x.