

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 №4.4
дисциплины «Объектно-ориентированное программирование»

Выполнила:

Мурашко Анастасия Юрьевна
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информатика и
вычислительная техника», очная
форма обучения

(подпись)

Проверил:

Богданов С.С., аспирант департамента
цифровых робототехнических систем
и электроники

(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.

Ход работы:

Изучила теоретический материал работы, создала общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами.

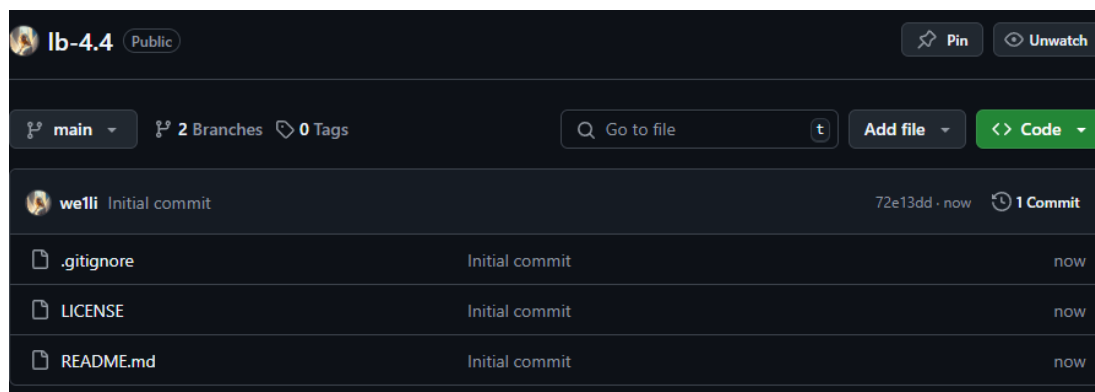


Рисунок 1. Новый репозиторий

Проработала пример лабораторной работы.

Пример 1: Для примера 2 лабораторной работы 9 добавьте возможность работы с исключениями и логгирование.

Исключения: IllegalYearError и UnknownCommandError используются для обработки некорректных данных и команд соответственно.

Журналирование: С помощью модуля logging действия пользователя записываются в файл workers.log.

XML обработка: Класс Staff может сохранять и загружать список работников в/из XML файла.

Напишем программу для решения поставленной задачи:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from dataclasses import dataclass, field
from datetime import date
import logging
import sys
```

```

from typing import List
import xml.etree.ElementTree as ET

# Класс пользовательского исключения в случае, если неверно введен номер года.
class IllegalYearError(Exception):
    def __init__(self, year, message="Illegal year number"):
        self.year = year
        self.message = message
        super(IllegalYearError, self).__init__(message)

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

# Класс пользовательского исключения в случае, если введенная команда является недопустимой.
class UnknownCommandError(Exception):
    def __init__(self, command, message="Unknown command"):
        self.command = command
        self.message = message
        super(UnknownCommandError, self).__init__(message)

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

# Класс для хранения информации о сотруднике.
@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int

# Класс для работы со списком сотрудников.
@dataclass
class Staff:
    workers: List[Worker] = field(default_factory=lambda: [])

    def add(self, name, post, year):
        # Получить текущую дату.
        today = date.today()
        if year < 0 or year > today.year:
            raise IllegalYearError(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

def load(self, filename):
    with open(filename, 'r', encoding='utf8') as fin:
        xml = fin.read()
        parser = ET.XMLParser(encoding="utf8")
        tree = ET.fromstring(xml, parser=parser)
        self.workers = []
        for worker_element in tree:
            name, post, year = None, None, None
            for element in worker_element:
                if element.tag == 'name':
                    name = element.text
                elif element.tag == 'post':
                    post = element.text
                elif element.tag == 'year':
                    year = int(element.text)
            if name is not None and post is not None and year is not None:
                self.workers.append(Worker(name=name, post=post, year=year))

def save(self, filename):

```

```

        root = ET.Element('workers')
        for worker in self.workers:
            worker_element = ET.Element('worker')
            name_element = ET.SubElement(worker_element, 'name')
            name_element.text = worker.name
            post_element = ET.SubElement(worker_element, 'post')
            post_element.text = worker.post
            year_element = ET.SubElement(worker_element, 'year')
            year_element.text = str(worker.year)
            root.append(worker_element)

        tree = ET.ElementTree(root)
        with open(filename, 'wb') as fout:
            tree.write(fout, encoding='utf8', xml_declaration=True)

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} году.")
            elif command == 'list':
                # Вывести список.
                print(staff)
                logging.info("Отображен список сотрудников.")
            elif command.startswith('select '):
                # Разбить команду на части для выделения периода.
                parts = command.split(maxsplit=1)

```

```

        selected = staff.select(int(parts[1]))

        # Вывести результаты запроса.
        if selected:
            for idx, worker in enumerate(selected, 1):
                print(f'{idx}: {worker.name}')
                logging.info(f"Найдено {len(selected)} работников со стажем
более {parts[1]} лет.")
            else:
                print("Работники с заданным стажем не найдены.")
                logging.warning(f"Работники со стажем более {parts[1]} лет не
найжены.")
        elif command.startswith('load '):
            # Разбить команду на части для имени файла.
            parts = command.split(maxsplit=1)
            staff.load(parts[1])
            logging.info(f"Загружены данные из файла {parts[1]}.")
        elif command.startswith('save '):
            # Разбить команду на части для имени файла.
            parts = command.split(maxsplit=1)
            staff.save(parts[1])
            logging.info(f"Сохранены данные в файл {parts[1]}.")
        elif command == 'help':
            # Вывести справку о работе с программой.
            print("Список команд:\n")
            print("add - добавить работника;")
            print("list - вывести список работников;")
            print("select <стаж> - запросить работников со стажем;")
            print("load <имя_файла> - загрузить данные из файла;")
            print("save <имя_файла> - сохранить данные в файл;")
            print("help - отобразить справку;")
            print("exit - завершить работу с программой.")
        else:
            raise UnknownCommandError(command)
    except Exception as exc:
        logging.error(f"Ошибка: {exc}")
        print(exc, file=sys.stderr)

```

```

>>> add
Фамилия и инициалы? Романов И.И
Должность? Водитель
Год поступления? 2024
>>> save primer
>>> list

```

№	Ф.И.О.	Должность	Год
1	fggf f.g	fgfgfg	2023
2	Романов И.И	Водитель	2024

Рисунок 2. Результат работы примера

```
INFO:root:Добавлен сотрудник: Романов И.И, Водитель, поступивший в 2024 году.  
ERROR:root:Ошибка: save -> Unknown command  
INFO:root:Сохранены данные в файл primer.  
INFO:root:Отображен список сотрудников.
```

Рисунок 3. Файл workers.log

Задача 1: 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.

Код программы:

```
def is_number(value):  
    """Проверяет, является ли введенное значение числом."""  
    try:  
        float(value) # Пытаемся преобразовать в число  
        return True  
    except ValueError:  
        return False  
  
def main():  
    # Запрос двух значений от пользователя  
    value1 = input("Введите первое значение: ")  
    value2 = input("Введите второе значение: ")  
  
    # Проверяем, являются ли оба значения числами  
    if is_number(value1) and is_number(value2):  
        # Если оба числа, выводим их сумму  
        result = float(value1) + float(value2)  
        print(f"Сумма чисел: {result}")  
    else:  
        # Если хотя бы одно значение не является числом, выполняем конкатенацию  
        result = value1 + value2  
        print(f"Конкатенация строк: {result}")  
  
if __name__ == "__main__":  
    main()
```

```
Введите первое значение: haha  
Введите второе значение: 3  
Конкатенация строк: haha3  
PS C:\Users\nasty> & C:/Users/nasty/AppData/Local/Programs/Python/Python312/python.exe g:/00П/4.4/z1.py  
Введите первое значение: 2  
Введите второе значение: 3  
Сумма чисел: 5.0
```

Рисунок 4. Результат работы программы

Задача 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.

Код программы:

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

import json
import argparse
from pathlib import Path

# Получаем путь к домашнему каталогу пользователя
home_dir = str(Path.home())

# Создаем путь к файлу данных в домашнем каталоге пользователя
file_path = Path(home_dir) / "idz.json"

# Функция для ввода данных о маршрутах
def add_route(routes, start, end, number):
    route = {
        "start": start,
        "end": end,
        "number": number
    }
    routes.append(route)
    return routes

# Функция для вывода информации о маршруте по номеру
def find_route(routes, number):
    found = False
    for route in routes:
        if route["number"] == number:
            print("Начальный пункт маршрута:", route["start"])
            print("Конечный пункт маршрута:", route["end"])
            found = True
            break
    if not found:
        print("Маршрут с таким номером не найден.")

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Управление маршрутами')
    parser.add_argument('--add', action='store_true', help='Добавить новый маршрут')
    parser.add_argument('--number', type=str, help='Номер маршрута для поиска')
```



```

args = parser.parse_args()

try:
    with open(file_path, "r") as file:
        routes = json.load(file)
except FileNotFoundError:
    routes = []

if args.add:
    start = input("Введите начальный пункт маршрута: ")
    end = input("Введите конечный пункт маршрута: ")
    number = input("Введите номер маршрута: ")
    routes = add_route(routes, start, end, number)

if args.number:
    find_route(routes, args.number)

# Сохраняем данные в файл JSON после ввода информации
with open(file_path, "w") as file:
    json.dump(routes, file)

```

```

Введите количество строк: 4
Введите количество столбцов: 10
Введите минимальное значение диапазона: 1
Введите максимальное значение диапазона: 1000
Сгенерированная матрица:
395 453 910 493 685 154 481 827 902 120
350 40 124 498 607 783 990 601 814 145
919 565 355 414 69 306 395 477 525 14
271 854 599 936 277 659 180 481 116 96

```

Рисунок 5. Результат работы программы

```

Введите количество строк: 3
Введите количество столбцов: rr
Ошибка: Введите целое число.
Введите количество столбцов: █

```

Рисунок 6. Обработка ошибок

Индивидуальное задание 1.

Вариант – 8.

Выполнить индивидуальное задание 1 лабораторной работы 2.19, добавив возможность работы с исключениями и логгирование.

Разбор программы:

Импортирование библиотек: В начале кода импортируем необходимые библиотеки для работы с JSON, аргументами командной строки, логгированием и путями.

Настройка логирования: Устанавливаем конфигурацию для логирования, чтобы записывать важные события и ошибки в файл.

Функции:

- `add_route`: Добавляет маршрут в список и проверяет на дубликаты.
- `find_route`: Находит и выводит информацию о маршруте по номеру.
- `display_menu`: Показывает доступные команды пользователю.

Основной блок программы:

Попытка загрузить существующий файл с маршрутами. Если файл не найден, создается новый.

Бесконечный цикл, который показывает меню и запрашивает команды.

В зависимости от команды выполняются соответствующие действия (добавление маршрута, поиск маршрута и т.д.).

После каждой операции данные сохраняются в файл, чтобы изменения не потерялись.

Код программы:

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

import json
import argparse
import logging
from pathlib import Path

# Настройка логгирования
logging.basicConfig(
    filename='routes.log', # Имя файла для записи логов
    level=logging.INFO,   # Уровень логирования (INFO - для общего учета
                          # событий)
    format='%(asctime)s - %(levelname)s - %(message)s' # Формат логов
)

# Получаем путь к домашнему каталогу пользователя
home_dir = str(Path.home())

# Создаем путь к файлу данных в домашнем каталоге пользователя
file_path = Path(home_dir) / "idz.json"

# Функция для ввода данных о маршрутах
def add_route(routes, start, end, number):
```

```

"""Добавляет новый маршрут в список маршрутов."""

# Проверка на дубликаты маршрута по номеру
for route in routes:
    if route["number"] == number:
        logging.warning(f"Попытка добавить дублирующий маршрут: {number}.")
        print(f"Ошибка: Маршрут с номером {number} уже существует.")
        return routes # Возвращаем неизменённый список маршрутов

# Создаем словарь для нового маршрута
route = {
    "start": start,
    "end": end,
    "number": number
}
routes.append(route) # Добавляем маршрут в список
logging.info(f"Добавлен маршрут: {route}") # Логируем добавление маршрута
return routes

# Функция для вывода информации о маршруте по номеру
def find_route(routes, number):
    """Ищет маршрут по его номеру и выводит его информацию."""

    found = False # Флаг, указывающий на то, найден ли маршрут
    for route in routes:
        if route["number"] == number: # Сравниваем номер маршрута
            print("Начальный пункт маршрута:", route["start"])
            print("Конечный пункт маршрута:", route["end"])
            found = True # Устанавливаем флаг в True, если маршрут найден
            break # Прекращаем поиск, так как маршрут найден
    if not found:
        logging.warning(f"Маршрут с номером {number} не найден.") # Логируем
предупреждение
        print("Маршрут с таким номером не найден.")

def display_menu():
    """Выводит меню доступных команд."""
    print("Доступные команды:")
    print("1. add - Добавить новый маршрут") # Команда для добавления маршрута
    print("2. find - Найти маршрут по номеру") # Команда для поиска маршрута
    print("3. exit - Выйти из программы") # Команда для выхода
    print("4. help - Вывести справку") # Команда для вывода справки

if __name__ == '__main__':
    # Попробуем загрузить маршруты из файла
    try:
        with open(file_path, "r", encoding='utf-8') as file:
            routes = json.load(file) # Загружаем маршруты из файла
    except FileNotFoundError:
        logging.info("Файл не найден, создается новый.") # Логируем, что файл не
найден

```

```

        routes = [] # Инициализируем пустой список маршрутов
    except json.JSONDecodeError:
        logging.error("Ошибка чтения файла. Проверьте формат JSON.") # Логируем
ошибку
        print("Ошибка чтения файла. Проверьте формат JSON.")
        routes = [] # Инициализируем пустой список маршрутов

    while True:
        display_menu() # Отображаем меню
        command = input("Введите команду: ").strip().lower() # Получаем команду
от пользователя

        if command == "exit": # Если команда "exit", выходим из программы
            print("Выход из программы.")
            break
        elif command == "add": # Если команда "add", добавляем новый маршрут
            start = input("Введите начальный пункт маршрута: ") # Получаем
начальный пункт
            end = input("Введите конечный пункт маршрута: ") # Получаем конечный
пункт
            number = input("Введите номер маршрута: ") # Получаем номер маршрута
            routes = add_route(routes, start, end, number) # Добавляем маршрут
        elif command == "find": # Если команда "find", ищем маршрут по номеру
            number = input("Введите номер маршрута для поиска: ") # Получаем
номер маршрута
            find_route(routes, number) # Ищем маршрут
        elif command == "help": # Если команда "help", показываем меню
            display_menu() # Показать меню снова
        else:
            print("Неизвестная команда. Попробуйте снова.") # Если команда не
распознана

    # Сохраняем данные в файл JSON после ввода информации
    try:
        with open(file_path, "w", encoding='utf-8') as file:
            json.dump(routes, file, ensure_ascii=False, indent=4) #
Сохраняем маршруты в файл
            logging.info("Данные успешно сохранены в файл.") # Логируем
успешное сохранение
    except Exception as e:
        logging.error(f"Ошибка при сохранении данных в файл: {e}") #
Логируем ошибку
        print(f"Ошибка при сохранении данных в файл: {e}")

```

```
idz1 > {} idz.json > ...  
1  [  
2    {  
3      "start": "1",  
4      "end": "2",  
5      "number": "44"  
6    }  
7  ]  
  
PROBLEMS  OUTPUT  TERMINAL  ...  
  
find - Найти маршрут по номеру  
exit - Выйти из программы  
Введите команду: add  
Введите начальный пункт маршрута: 1  
Введите конечный пункт маршрута: 2  
Введите конечный пункт маршрута: 2  
Ошибка: Маршрут с номером 44 уже существует.
```

Рисунок 7. Результат работы программы

Индивидуальное задание 2.

Изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды.

Разбор программы:

Теперь в логах будет отображаться время выполнения команд с точностью до миллисекунд. Это полезно для отслеживания времени выполнения операций.

```
format='% (asctime)s.%(msecs)03d - %(levelname)s - %(message)s'
```

После выполнения каждой команды добавлено логирование, чтобы фиксировать действия пользователя. Например:

```
logging.info(f"Выполнена команда: добавлен маршрут {number}.")
```

```
logging.info(f"Выполнена команда: поиск маршрута {number}.")
```

Ошибка при чтении файла:

Добавлена обработка исключений при загрузке данных из файла. Если файл не существует или он имеет неправильный формат, это будет логироваться:

except json.JSONDecodeError:

 logging.error("Ошибка чтения файла. Проверьте формат JSON.")

Ошибка при сохранении данных:

except Exception as e:

 logging.error(f"Ошибка при сохранении данных в файл: {e}")

Код программы:

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

import json
import argparse
import logging
from pathlib import Path

# Настройка логгирования
logging.basicConfig(
    filename='routes.log', # Имя файла для записи логов
    level=logging.INFO,    # Уровень логгирования (INFO - для общего учета
                           # событий)
    format='%(asctime)s.%(msecs)03d - %(levelname)s - %(message)s', # Формат
                           # логов с миллисекундами
    datefmt='%Y-%m-%d %H:%M:%S' # Формат даты и времени
)

# Создаем путь к файлу данных в домашнем каталоге пользователя
file_path = Path("G:/ООП/4.4") / "idz.json"

# Функция для ввода данных о маршрутах
def add_route(routes, start, end, number):
    """Добавляет новый маршрут в список маршрутов."""

    # Проверка на дубликаты маршрута по номеру
    for route in routes:
        if route["number"] == number:
            logging.warning(f"Попытка добавить дублирующий маршрут: {number}.")
            print(f"Ошибка: Маршрут с номером {number} уже существует.")
            return routes

    route = {
        "start": start,
        "end": end,
        "number": number
    }
    routes.append(route) # Добавляем маршрут в список
    logging.info(f"Добавлен маршрут: {route}") # Логируем добавление маршрута
    return routes
```

```

# Функция для вывода информации о маршруте по номеру
def find_route(routes, number):
    """Ищет маршрут по его номеру и выводит его информацию."""

    found = False # Флаг, указывающий на то, найден ли маршрут
    for route in routes:
        if route["number"] == number: # Сравниваем номер маршрута
            print("Начальный пункт маршрута:", route["start"])
            print("Конечный пункт маршрута:", route["end"])
            found = True # Устанавливаем флаг в True, если маршрут найден
            break # Прекращаем поиск, так как маршрут найден
    if not found:
        logging.warning(f"Маршрут с номером {number} не найден.") # Логируем
        предупреждение
        print("Маршрут с таким номером не найден.")

def display_menu():
    """Выводит меню доступных команд."""
    print("Доступные команды:")
    print("add - Добавить новый маршрут") # Команда для добавления маршрута
    print("find - Найти маршрут по номеру") # Команда для поиска маршрута
    print("exit - Выйти из программы") # Команда для выхода

if __name__ == '__main__':
    # Попробуем загрузить маршруты из файла
    try:
        with open(file_path, "r", encoding='utf-8') as file:
            routes = json.load(file) # Загружаем маршруты из файла
    except FileNotFoundError:
        logging.info("Файл не найден, создается новый.") # Логируем, что файл не
        найден
        routes = [] # Инициализируем пустой список маршрутов
    except json.JSONDecodeError:
        logging.error("Ошибка чтения файла. Проверьте формат JSON.") # Логируем
        ошибку
        print("Ошибка чтения файла. Проверьте формат JSON.")
        routes = [] # Инициализируем пустой список маршрутов

    while True:
        display_menu() # Отображаем меню
        command = input("Введите команду: ").strip().lower() # Получаем команду
        от пользователя

        if command == "exit": # Если команда "exit", выходим из программы
            print("Выход из программы.")
            break
        elif command == "add": # Если команда "add", добавляем новый маршрут
            start = input("Введите начальный пункт маршрута: ") # Получаем
            начальный пункт

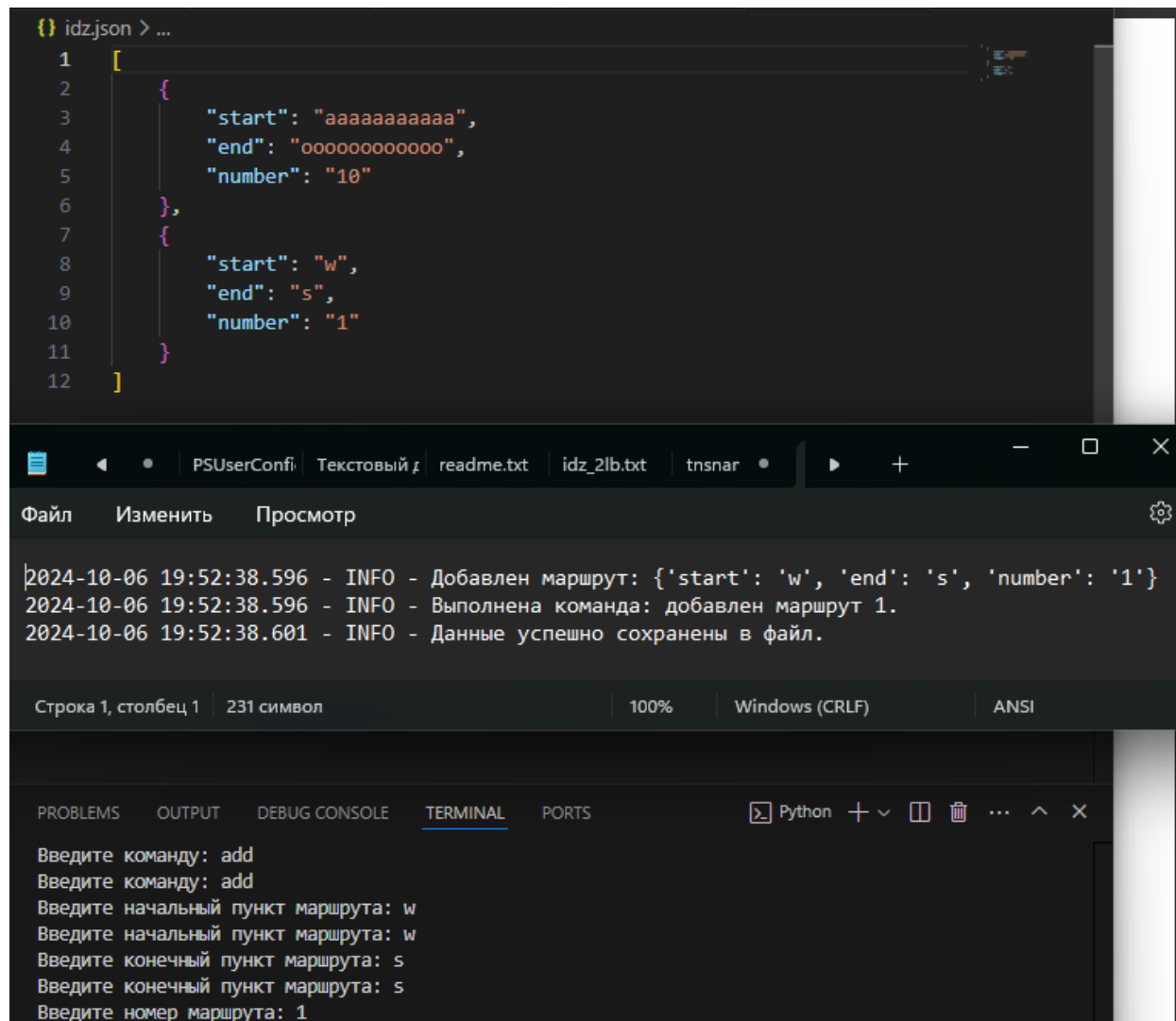
```

```

        end = input("Введите конечный пункт маршрута: ") # Получаем конечный
пункт
        number = input("Введите номер маршрута: ") # Получаем номер маршрута
        routes = add_route(routes, start, end, number) # Добавляем маршрут
        logging.info(f"Выполнена команда: добавлен маршрут {number}.") #
Логируем выполнение команды
        elif command == "find": # Если команда "find", ищем маршрут по номеру
            number = input("Введите номер маршрута для поиска: ") # Получаем
номер маршрута
            find_route(routes, number) # Ищем маршрут
            logging.info(f"Выполнена команда: поиск маршрута {number}.") #
Логируем выполнение команды
        else:
            print("Неизвестная команда. Попробуйте снова.") # Сообщение о
неверной команде

# Сохраняем данные в файл JSON после ввода информации
try:
    with open(file_path, "w", encoding='utf-8') as file:
        json.dump(routes, file, ensure_ascii=False, indent=4) #
Сохраняем маршруты в файл
        logging.info("Данные успешно сохранены в файл.") # Логируем
успешное сохранение
except Exception as e:
    logging.error(f"Ошибка при сохранении данных в файл: {e}") #
Логируем ошибку
    print(f"Ошибка при сохранении данных в файл: {e}")

```

```
{
  "start": "aaaaaaaaaa",
  "end": "oooooooooooo",
  "number": "10"
},
{
  "start": "w",
  "end": "s",
  "number": "1"
}
]
```

2024-10-06 19:52:38.596 - INFO - Добавлен маршрут: {'start': 'w', 'end': 's', 'number': '1'}

2024-10-06 19:52:38.596 - INFO - Выполнена команда: добавлен маршрут 1.

2024-10-06 19:52:38.601 - INFO - Данные успешно сохранены в файл.

Введите команду: add

Введите команду: add

Введите начальный пункт маршрута: w

Введите начальный пункт маршрута: w

Введите конечный пункт маршрута: s

Введите конечный пункт маршрута: s

Введите номер маршрута: 1

Рисунок 8. Результат работы программы

Контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

В языке Python существует два основных вида ошибок:

- Синтаксические ошибки: эти ошибки возникают, когда код написан с нарушением правил синтаксиса Python. Такие ошибки фиксируются ещё до выполнения программы. Пример: отсутствие двоеточия после определения функции.
- Исключения (Exceptions): это ошибки, возникающие во время выполнения программы, когда происходит непредвиденная ситуация, например, деление на ноль, обращение к несуществующему элементу списка

или файл, который не существует.

Примеры исключений:

- `ZeroDivisionError`: деление на ноль.
- `IndexError`: обращение к несуществующему индексу списка.
- `KeyError`: обращение к отсутствующему ключу в словаре.
- `FileNotFoundError`: файл не найден.

2. Как осуществляется обработка исключений в языке программирования Python?

В Python обработка исключений осуществляется с помощью конструкции `try-except`. Код, который может вызвать исключение, помещается в блок `try`, а в блоке `except` указывается, как обработать это исключение.

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

Блок `finally`: этот блок выполняется всегда, независимо от того, произошло исключение или нет. Он используется для выполнения завершающих действий, таких как закрытие файлов или освобождение ресурсов.

`try:`

```
file = open('data.txt', 'r')
```

```
data = file.read()
```

`except FileNotFoundError:`

```
print("Файл не найден.")
```

`finally:`

```
file.close() # Закрытие файла выполнится всегда
```

Блок `else`: этот блок выполняется, если в блоке `try` не возникло исключений. Он используется для выполнения кода, который должен быть выполнен только в случае успеха.

`try:`

```
result = 10 / 2
```

`except ZeroDivisionError:`

```
print("Ошибка: деление на ноль")
```

else:

```
print(f'Результат: {result}') # Выполнится, если не было исключений
```

4. Как осуществляется генерация исключений в языке Python?

В Python можно вручную сгенерировать исключение с помощью оператора `raise`. Это используется, когда нужно сигнализировать об ошибочной ситуации в программе.

5. Как создаются классы пользовательский исключений в языке Python?

В Python можно создавать свои собственные исключения, создавая классы, наследуемые от базового класса `Exception`. Это позволяет создавать специфичные для программы исключения и обрабатывать их.

6. Каково назначение модуля `logging`?

Модуль `logging` используется для ведения журнала (логирования) событий в программе. Он позволяет записывать сообщения в разные места: в консоль, файлы или другие выходные потоки. Это помогает отслеживать работу программы, записывать важные события, ошибки, предупреждения и другую информацию.

Основные преимущества модуля `logging`:

- Можно вести логи разного уровня важности (информация, отладка, ошибки).
- Можно записывать логи в файл, что полезно для анализа после завершения программы.
- Можно настроить разные форматы для логов (например, добавлять дату и время).
- Можно записывать логи в разные места одновременно.

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

Модуль `logging` поддерживает следующие уровни логгирования, которые обозначают важность события:

1. **DEBUG:** Отладочные сообщения, которые полезны для разработки и анализа работы программы.
2. **INFO:** Информационные сообщения, показывающие нормальный ход работы программы.
3. **WARNING:** Предупреждения, которые сигнализируют о потенциальных проблемах, но программа продолжает работать.
4. **ERROR:** Ошибки, которые приводят к частичной неработоспособности программы.
5. **CRITICAL:** Критические ошибки, из-за которых программа может остановиться.

Пример настройки уровней логирования:

```
import logging
logging.basicConfig(level=logging.DEBUG)
logging.debug("Это отладочное сообщение")
logging.info("Это информационное сообщение")
logging.warning("Это предупреждение")
logging.error("Это ошибка")
logging.critical("Это критическая ошибка")
```

Вывод: в ходе лабораторной работы приобретены навыки в работе по исключениям при написании программ помощью языка программирования Python.