

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
№220 дисциплины «Основы программной инженерии»

Выполнил:

Джараян Арег Александрович
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 _____

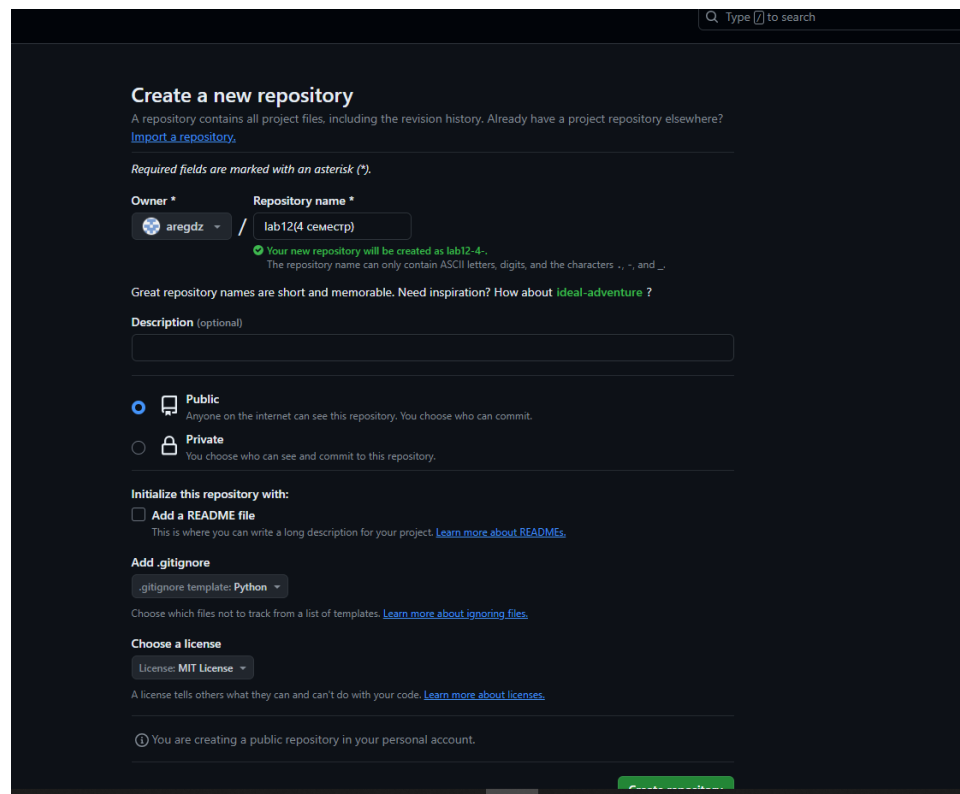
Ставрополь, 2024 г.

Тема: Лабораторная работа 4.4 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 нового репозитория с лицензией MIT.



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 *
aregdz / lab12(4 семестр)
✔ Your new repository will be created as lab12-4-.
The repository name can only contain ASCII letters, digits, and the characters -, ., and _.

Great repository names are short and memorable. Need inspiration? How about [ideal-adventure](#)?

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.](#)

📌 You are creating a public repository in your personal account.

Рисунок 1 – создание репозитория

2. Клонировал репозиторий на рабочий ПК.

```
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/11/11lab (develop)
$ cd "D:\Рабочий стол\4 семестр\опи\12"

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/12
$ git clone "https://github.com/aregdz/lab12-4-.git"
Cloning into 'lab12-4-'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/12
$
```

Рисунок 2 – клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

```
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
```

Рисунок 4 – Файл .gitignore

```
aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/12
$ cd lab12-4-

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/12/lab12-4- (main)
$ git checkout -b develop
Switched to a new branch 'develop'

aregd@DESKTOP-5KV9QA9 MINGW64 /d/Рабочий стол/4 семестр/опи/12/lab12-4- (develop)
$
```

Рисунок 4 – организация ветки

(venv) PS D:\Рабочий стол\4 сем	
Package	Version

black	24.4.0
cfgv	3.4.0
click	8.1.7
colorama	0.4.6
distlib	0.3.8
pyflakes	3.2.0
PyYAML	6.0.1
setuptools	69.5.1
virtualenv	20.25.2

Рисунок 5 – создание виртуального окружения

4.Пример 1.

```
#!/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}"

@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):
        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):
        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)
        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):
        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}, "
                    f"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 номера года.
        parts = command.split(maxsplit=1)

        # Запросить работников.
        selected = staff.select(parts[1])

        # 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 "
                f"стажем более {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.
        parts = command.split(maxsplit=1)

        # Загрузить данные из файла.
        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)

        # Сохранить данные в файл.
        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)

```

Рисунок 6 – пример 1

```
"D:\Рабочий стол\4 семестр\опи\11\11lab\venv\S
3/4
Введите обыкновенную дробь: 4/8
1/2
5/4
1/4
3/8
2/3

Process finished with exit code 0
|
```

Рисунок 7 – выполнение примера 1

5.Задание 1. . Решите следующую задачу: 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.

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

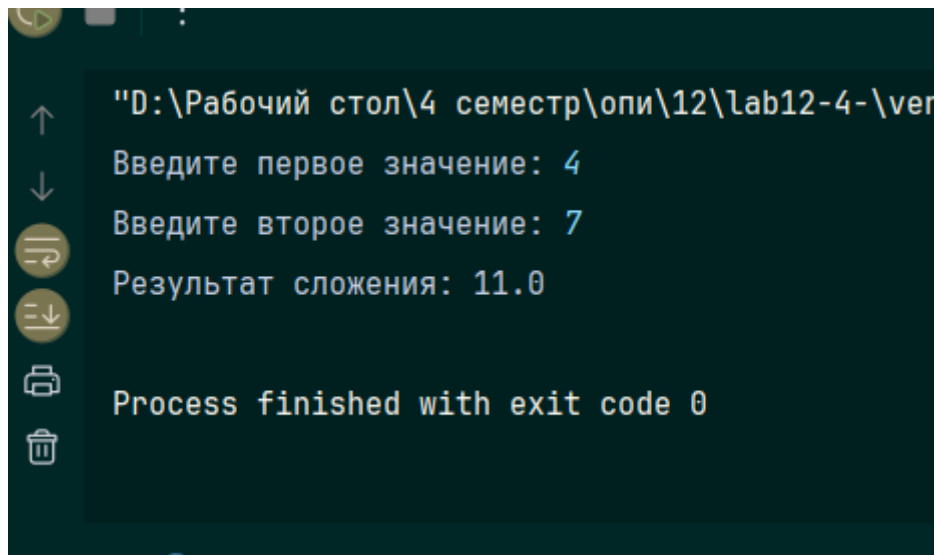
def main():
    try:
        value1 = input("Введите первое значение: ")
        value2 = input("Введите второе значение: ")

        num1 = float(value1)
        num2 = float(value2)

        result = num1 + num2
        print("Результат сложения:", result)
    except ValueError:
        result = value1 + value2
        print("Результат конкатенации:", result)

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

Рисунок 8 – Задание 1



```
"D:\Рабочий стол\4 семестр\опи\12\lab12-4-\ven
Введите первое значение: 4
Введите второе значение: 7
Результат сложения: 11.0
Process finished with exit code 0
```

Рисунок 9 – пример выполнения примера 2

6. Задание 2. Решите следующую задачу: 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.

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

import random

def generate_matrix(rows, cols, min_value, max_value):
    matrix = []
    for _ in range(rows):
        row = [random.randint(min_value, max_value) for _ in range(cols)]
        matrix.append(row)
    return matrix

def main():
    try:
        rows = int(input("Введите количество строк: "))
        cols = int(input("Введите количество столбцов: "))

        min_value = int(input("Введите минимальное значение: "))
        max_value = int(input("Введите максимальное значение: "))

        matrix = generate_matrix(rows, cols, min_value, max_value)

        print("Сгенерированная матрица:")
        for row in matrix:
            print(row)
    except ValueError:
        print("Ошибка: Введите целое число.")

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

Рисунок 9 – пример 3

```
"D:\Рабочий стол\4 семестр\опи\12\lab12-4-\venv\Scripts\python.exe"
Введите количество строк: 4
Введите количество столбцов: 5
Введите минимальное значение: 3
Введите максимальное значение: 5
Сгенерированная матрица:
[3, 4, 5, 5, 5]
[3, 4, 5, 5, 4]
[5, 4, 5, 5, 3]
[4, 3, 3, 5, 5]

Process finished with exit code 0
```

Рисунок 10 – пример выполнения задания 2

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

```
#!/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

class FlightManager:
    def __init__(self):
        self.aircrafts = []

    def add(self):
        """
        Метод для добавления информации о новых рейсах
        """
        # Запросить данные о работнике.
        name = input("Название пункта назначения рейса? ")
        number = int(input("Номер рейса? "))
        tip = input("Тип самолета? ")
        # Создать словарь.
        i = {
```

```

        'name': name,
        'number': number,
        'tip': tip,
    }

    self.aircrafts.append(i)
    # Отсортировать список в случае необходимости.
    if len(self.aircrafts) > 1:
        self.aircrafts.sort(key=lambda item: item.get('name', ''))

def list(self):
    """
    Метод для вывода списка добавленных рейсов
    """
    # Заголовок таблицы.
    line = '+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 30,
        '-' * 20,
        '-' * 8
    )
    print(line)

    # Вывести данные о всех сотрудниках.
    for idx, i in enumerate(self.aircrafts, 1):
        print(
            '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                idx,
                i.get('name', ''),
                i.get('number', ''),
                i.get('tip', '')
            )
        )
    print(line)

def select(self):
    """
    Метод для получения номера рейса и пункта назначения по заданному типу
    самолёта.
    """
    # Разбить команду на части для выделения номера года.
    parts = input("Введите значение: ")
    # Проверить сведения работников из списка.
    count = 0

    for i in self.aircrafts:
        for k, v in i.items():

            if v == parts:
                print("Пункт назначения - ", i["name"])
                print("Номер рейса - ", i["number"])
                count += 1

    # Если счетчик равен 0, то работники не найдены.
    if count == 0:
        print("Рейс с заданным типом самолёта не найден.")

def help(self):
    """
    Метод для вывода списка команд
    """
    print("Список команд:\n")
    print("add - добавить рейс;")
    print("list - вывести список рейсов;")
    print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для данного типа самолёта")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

if __name__ == '__main__':

```

```

logging.basicConfig(
    filename='fly1.log',
    level=logging.INFO,
)

flight_manager = FlightManager()

while True:
    try:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствие с командой.

        match command:
            case 'exit':
                break

            case 'add':
                flight_manager.add()

            case 'list':
                flight_manager.list()

            case 'select':
                flight_manager.select()

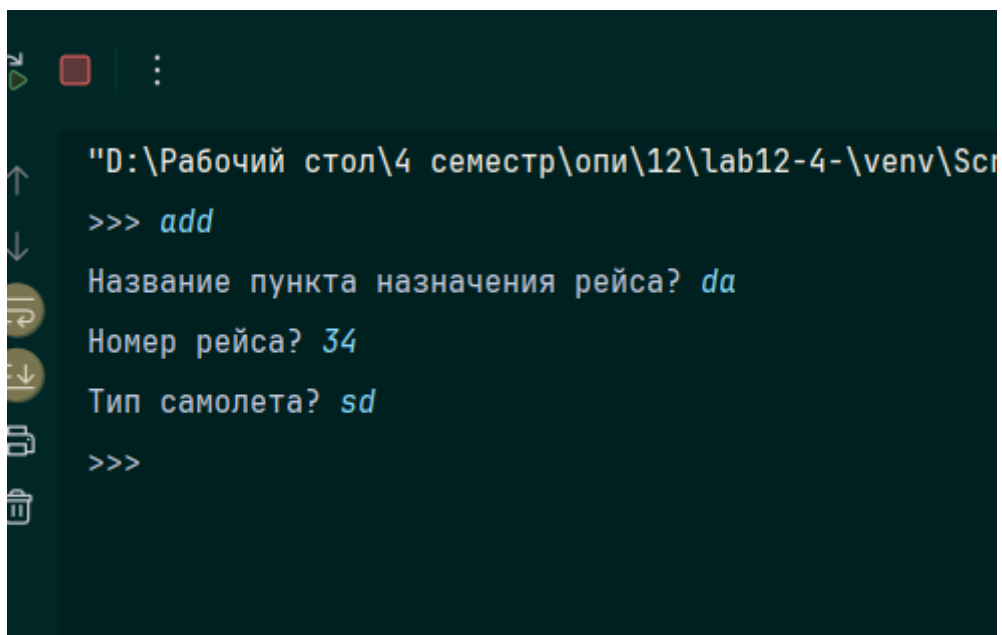
            case 'help':
                flight_manager.help()

            case _:
                raise UnknownCommandError(command)

    except Exception as exc:
        logging.error(f"Ошибка: {exc}")
        print(exc, file=sys.stderr)

```

Рисунок 11 – Выполнение задания 1



The screenshot shows a terminal window with a dark background. The prompt is a Windows file path: "D:\Рабочий стол\4 семестр\опи\12\lab12-4-\venv\Scr". The user enters the command ">>> add". The program then prompts for flight details: "Название пункта назначения рейса? da", "Номер рейса? 34", and "Тип самолета? sd". After the user enters these details, the prompt returns to ">>>".

Рисунок 12– пример выполнение задания 1

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

```
#!/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

class FlightManager:
    def __init__(self):
        self.aircrafts = []

    def add(self):
        """
        Метод для добавления информации о новых рейсах
        """
        # Запросить данные о работнике.
        name = input("Название пункта назначения рейса? ")
        number = int(input("Номер рейса? "))
        tip = input("Тип самолета? ")
        # Создать словарь.
        i = {
            'name': name,
            'number': number,
            'tip': tip,
        }

        self.aircrafts.append(i)
        # Отсортировать список в случае необходимости.
        if len(self.aircrafts) > 1:
            self.aircrafts.sort(key=lambda item: item.get('name', ''))

    def list(self):
        """
        Метод для вывода списка добавленных рейсов
        """
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8,
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, i in enumerate(self.aircrafts, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    i.get('name', ''),
                    i.get('number', ''),
                    i.get('tip', '')
                )
            )
        print(line)
```

```

def select(self):
    """
    Метод для получения номера рейса и пункта назначения по заданному типу
    самолёта.
    """
    # Разбить команду на части для выделения номера года.
    parts = input("Введите значение: ")
    # Проверить сведения работников из списка.
    count = 0

    for i in self.aircrafts:
        for k, v in i.items():

            if v == parts:
                print("Пункт назначения - ", i["name"])
                print("Номер рейса - ", i["number"])
                count += 1

    # Если счетчик равен 0, то работники не найдены.
    if count == 0:
        print("Рейс с заданным типом самолёта не найден.")

def help(self):
    """
    Метод для вывода списка команд
    """
    print("Список команд:\n")
    print("add - добавить рейс;")
    print("list - вывести список рейсов;")
    print("select <тип> - вывод на экран пунктов назначения и номеров рейсов для
данного типа самолёта")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

if __name__ == '__main__':

    logging.basicConfig(
        filename='fly2.log',
        level=logging.INFO,
        format='%(asctime)s.%(msecs)03d %(levelname)s %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S'
    )

    flight_manager = FlightManager()

    while True:
        try:
            # Запросить команду из терминала.
            command = input(">>> ").lower()
            # Выполнить действие в соответствии с командой.

            match command:
                case 'exit':
                    break

                case 'add':
                    flight_manager.add()

                case 'list':
                    flight_manager.list()

                case 'select':
                    flight_manager.select()

                case 'help':
                    flight_manager.help()

                case _:
                    raise UnknownCommandError(command)

        except Exception as exc:

```

```
logging.error(f"Ошибка: {exc}")
print(exc, file=sys.stderr)
```

Рисунок 13 – выполнение задания 2

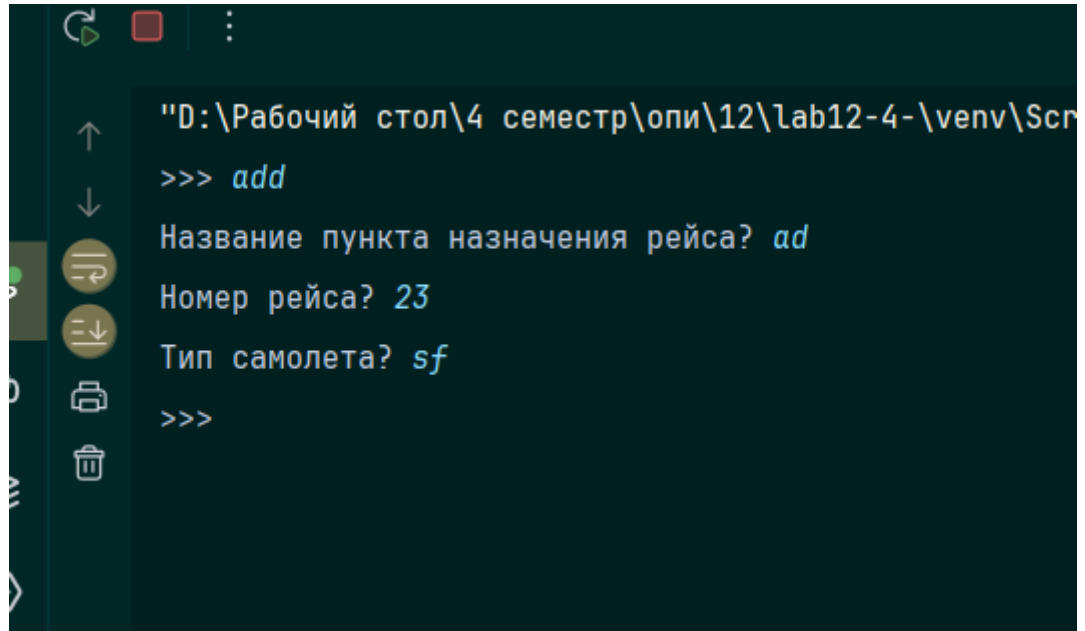


Рисунок 14 – пример выполнения задания 2

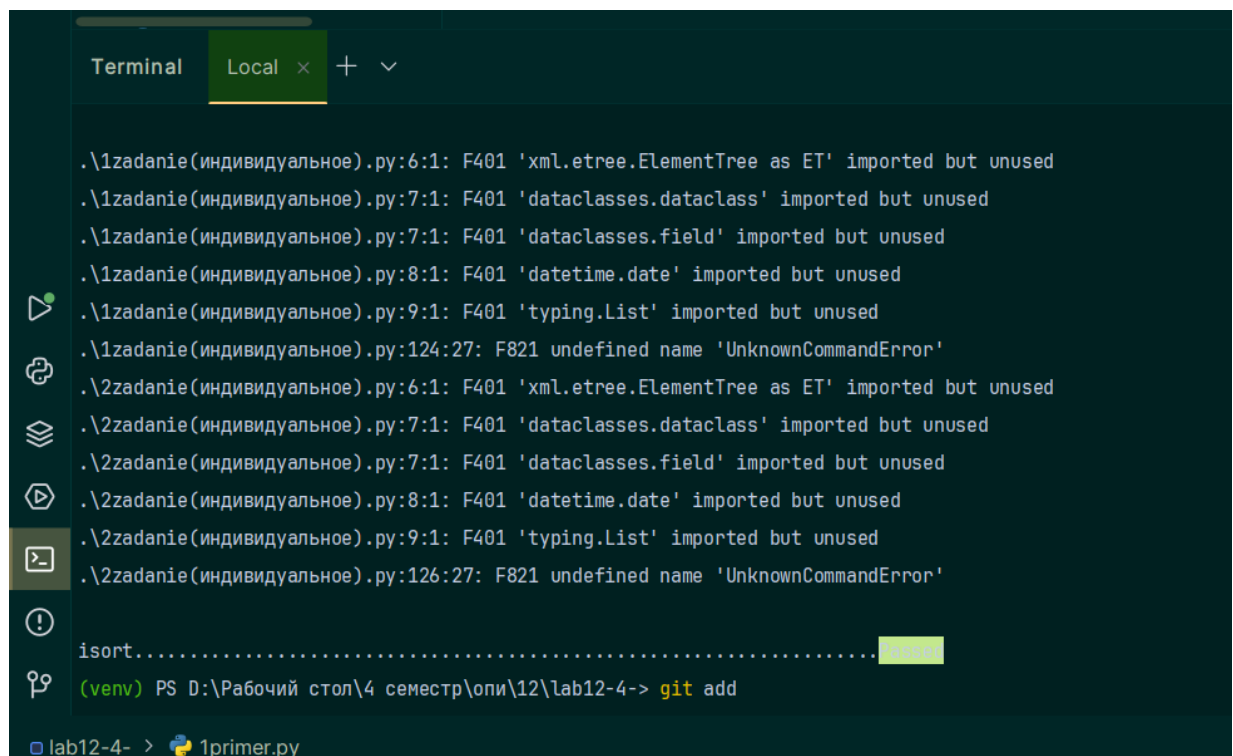


Рисунок 15 - фиксация изменений

Контрольные вопросы:

1. В языке программирования Python существуют различные виды ошибок, включая:

-Синтаксические ошибки: возникают при нарушении правил языка Python при написании кода. Примером может быть ошибка в использовании ключевого слова или неправильное использование операторов.

-Исключения времени выполнения: возникают во время выполнения программы из-за некорректных условий или операций. Примерами могут быть деление на ноль, обращение к несуществующему индексу списка и другие.

2.Обработка исключений в языке программирования Python осуществляется с помощью конструкции try-except. Код, который может вызвать исключение, помещается в блок try, а код для обработки исключения - в блок except. Если исключение произошло в блоке try, интерпретатор Python переходит к соответствующему блоку except, который содержит обработчик исключения.

3.Блок finally используется для выполнения кода независимо от того, возникло исключение или нет. Он выполняется после блока try-except и гарантирует, что определенные операции будут выполнены, даже если произошло исключение.Блок else в try-except используется для выполнения кода, который должен быть выполнен в случае, если в блоке try не было исключений. Если исключение не произошло, интерпретатор Python выполняет код в блоке else.

4. Исключения могут быть сгенерированы с помощью оператора raise.

5. Для создания пользовательских исключений в Python нужно определить новый класс, который наследуется от одного из встроенных классов исключений, например, Exception.

6. Модуль logging предоставляет гибкие средства для журналирования сообщений в Python. Он позволяет контролировать уровень подробности и формат вывода сообщений, а также сохранять сообщения в файлы логов.

7. Модуль logging поддерживает следующие уровни логгирования:

-DEBUG: Используется для вывода отладочной информации. Пример использования: вывод значений переменных для отладки программы.

-INFO: Используется для информационных сообщений о ходе выполнения программы. Пример использования: вывод сообщений о старте и завершении определенных операций.

-WARNING: Используется для предупреждений о потенциальных проблемах. Пример использования: предупреждение о неожиданных значениях переменных.

-ERROR: Используется для сообщений об ошибках, которые не приводят к прекращению работы программы. Пример использования: сообщение о неудачном завершении операции.

-CRITICAL: Используется для сообщений о критических ошибках, которые приводят к немедленному завершению программы. Пример использования: сообщение о невозможности доступа к важным ресурсам.