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»**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Кожуховский Виктор Андреевич  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. Изучил теоретический материал работы.

2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.

3. Выполнил клонирование созданного репозитория.

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE.

5. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.

6. Создал проект в папке репозитория.

7. Проработал примеры лабораторной работы.

9. Разработайте программы по следующим описаниям.

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.

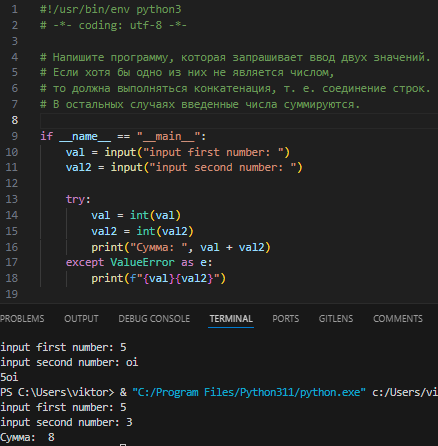


Рисунок 1. Выполнение общего задания 1

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.

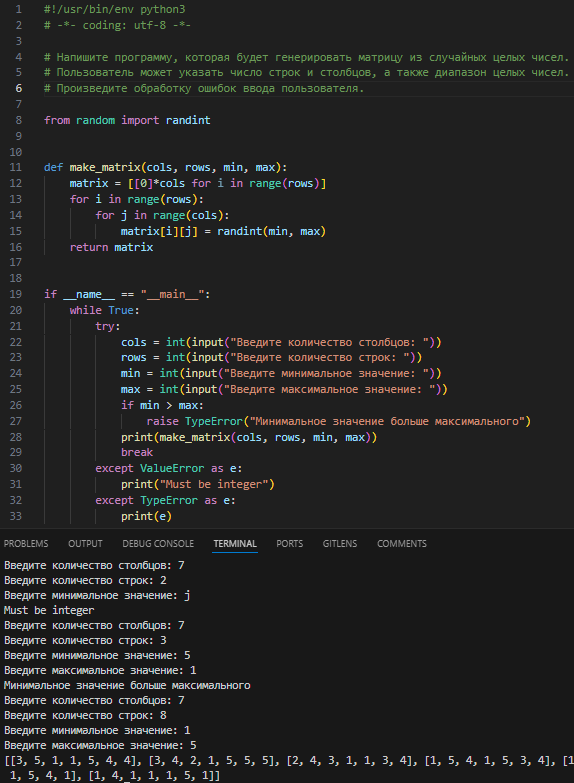


Рисунок 2. Выполнение общего задания 2

8. Выполнил индивидуальное задание для варианта 14.

Задание 1

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

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

from pathlib import Path

import argparse

import json

import os

from datetime import datetime

import jsonschema

import logging

logging.basicConfig(level=logging.DEBUG)

person\_schema = {

"type": "object",

"properties": {

"name": {"type": "string"},

"surname": {"type": "string"},

"date\_of\_birth": {"type": "string", "format": "date"},

"zodiac\_sign": {"type": "string"}

},

"required": ["name", "surname", "date\_of\_birth", "zodiac\_sign"]

}

def validate\_person(person\_data, schema):

try:

jsonschema.validate(person\_data, schema)

return True

except jsonschema.exceptions.ValidationError as e:

logging.error(f"Данные человека не соответствуют схеме: {e}")

return False

def add\_person(people, name, surname, date\_of\_birth, zodiac\_sign):

"""

Добавление нового человека в список.

Список сортируется по знаку зодиака после добавления нового элемента.

"""

date\_of\_birth = datetime.strptime(date\_of\_birth, '%d.%m.%Y')

person = {

'name': name,

'surname': surname,

'date\_of\_birth': date\_of\_birth,

'zodiac\_sign': zodiac\_sign

}

people.append(person)

people.sort(key=lambda item: item.get('zodiac\_sign', ''))

return people

def list\_people(people):

"""

Вывод таблицы людей.

"""

line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(

'-' \* 4,

'-' \* 20,

'-' \* 20,

'-' \* 15,

'-' \* 13

)

print(line)

print(

'| {:^4} | {:^20} | {:^20} | {:^15} | {:^12} |'.format(

"№",

"Имя",

"Фамилия",

"Знак Зодиака",

"Дата рождения"

)

)

print(line)

for idx, person in enumerate(people, 1):

birth\_date\_str = person.get('date\_of\_birth').strftime('%d.%m.%Y')

print(

'| {:^4} | {:<20} | {:<20} | {:<15} | {:<13} |'.format(

idx,

person.get('name', ''),

person.get('surname', ''),

person.get('zodiac\_sign', ''),

birth\_date\_str

)

)

print(line)

def select\_people(people, month):

"""

Вывести список людей, родившихся в заданном месяце.

"""

count = 0

for person in people:

if person.get('date\_of\_birth').month == month:

count += 1

print('{:>4}: {} {}'.format(count, person.get(

'name', ''), person.get('surname', '')))

if count == 0:

print("Люди, родившиеся в указанном месяце, не найдены.")

def save\_people(file\_name, staff):

"""

Сохранить всех работников в файл JSON.

"""

staff\_formatted = [{\*\*person, 'date\_of\_birth': person.get(

'date\_of\_birth').strftime('%d.%m.%Y')} for person in staff]

# Открыть файл с заданным именем для записи.

with open(file\_name, "w", encoding="utf-8") as fout:

# Выполнить сериализацию данных в формат JSON.

json.dump(staff\_formatted, fout, ensure\_ascii=False, indent=4)

def load\_people(file\_name):

"""

Загрузить всех людей из файла JSON.

"""

# Открыть файл с заданным именем для чтения.

with open(file\_name, "r", encoding="utf-8") as fin:

staff\_loaded = json.load(fin)

result\_people = []

cnt = 0

for person in staff\_loaded:

cnt += 1

if validate\_person(person, person\_schema):

try:

person['date\_of\_birth'] = datetime.strptime(

person['date\_of\_birth'], '%d.%m.%Y')

result\_people.append(person)

except:

logging.error(

f"Ошибка при разборе даты в записи, пропуск записи"

"{cnt}.")

else:

logging.error("Неверные данные человека, пропуск записи.")

return result\_people

def main():

file\_parser = argparse.ArgumentParser(add\_help=False)

file\_parser.add\_argument(

"filename",

action="store",

help="The data file name"

)

# Создание основного парсера.

parser = argparse.ArgumentParser(description="Управление списком людей")

subparsers = parser.add\_subparsers(dest="command")

# Создание парсера для добавления человека.

parser\_add = subparsers.add\_parser(

'add', parents=[file\_parser], help="Добавить человека")

parser\_add.add\_argument("-n", "--name", help="Имя человека")

parser\_add.add\_argument("-s", "--surname", help="Фамилия человека")

parser\_add.add\_argument(

"-d", "--date\_of\_birth", help="Дата рождения (формат ДД.ММ.ГГГГ)")

parser\_add.add\_argument("-z", "--zodiac\_sign", help="Знак зодиака")

# Создание парсера для вывода списка людей.

\_ = subparsers.add\_parser(

'list', parents=[file\_parser], help="Вывести список людей")

# Создание парсера для выбора человека по месяцу рождения.

parser\_select = subparsers.add\_parser(

'select', parents=[file\_parser], help="Выбрать людей по месяцу рождения")

parser\_select.add\_argument(

"-m", "--month", type=int, help="Месяц рождения")

# Разбираем аргументы командной строки.

args = parser.parse\_args()

is\_dirty = False

home\_directory = Path.home()

data\_directory = home\_directory / 'data'

# Создаем каталог, если он не существует.

data\_directory.mkdir(exist\_ok=True)

try:

filename = data\_directory / args.filename

except Exception as e:

logging.error(

"Отсутствуют один или нестолько обязательных аргументов.")

exit(1)

if os.path.exists(filename):

try:

people = load\_people(filename)

except Exception as e:

logging.error(

"Ошибка при загрузке данных из файла.")

exit(1)

else:

people = []

# Определяем, какую команду нужно выполнить.

if args.command == 'add':

try:

people = add\_person(people, args.name, args.surname,

args.date\_of\_birth, args.zodiac\_sign)

is\_dirty = True

except Exception as e:

logging.error(

"Отсутствуют один или нестолько обязательных аргументов.")

elif args.command == 'list':

list\_people(people)

elif args.command == 'select':

select\_people(people, args.month)

if is\_dirty:

save\_people(filename, people)

if \_\_name\_\_ == '\_\_main\_\_':

main()

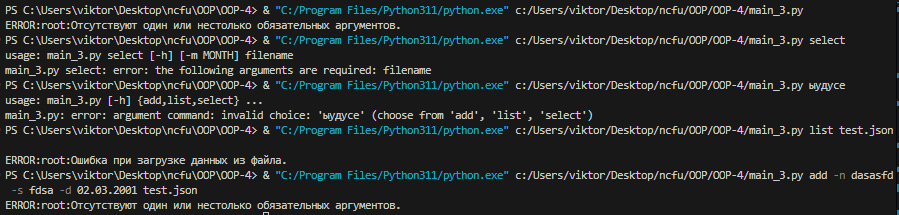


Рисунок 3. Код решения индивидуального задания 1 и его выполнение

Задание 2

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

Изменена только строка logging.basicConfig(level=logging.DEBUG) на

logging.basicConfig(level=logging.DEBUG, filename='app.log', encoding='utf-8', format='%(asctime)s - %(levelname)s - %(message)s')

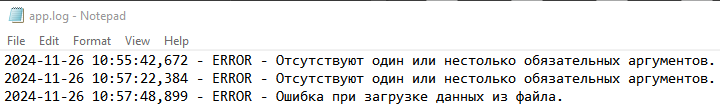


Рисунок 4. Код решения индивидуального задания 2 и его выполнение

9. Зафиксировал сделанные изменения в репозитории.

10. Выполнил слияние ветки для разработки с веткой master/main.

11. Отправил сделанные изменения на сервер GitHub.

Ссылка: https://github.com/Viktorkozh/OOP-4

Контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

Синтаксические ошибки и исключения.

2. Как осуществляется обработка исключений в языке программирования Python?

С помощью конструкции try… except и finally.

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

Не зависимо от того, возникнет или нет во время выполнения кода в блоке try исключение, код в блоке finally все равно будет выполнен.

4. Как осуществляется генерация исключений в языке Python?

Для принудительной генерации исключения используется инструкция raise.

5. Как создаются классы пользовательский исключений в языке Python?

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

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

logging.basicConfig(level = logging.DEBUG)

logging.debug("Debug message!")

logging.info("Info message!")

logging.warning("Warning message!")

logging.error("Error message!")

logging.critical("Critical message!")

Вывод: приобрел навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x