

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»**  
**Вариант 2**

Выполнил:  
Иващенко Олег Андреевич  
3 курс, группа ИВТ-б-о-22-1,  
09.03.02 «Информационные и  
вычислительные машины»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем»

---

(подпись)

Руководитель практики:  
Воронкин Роман Александрович,  
доцент департамента цифровых,  
робототехнических систем и  
электроники

---

(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:

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

Листинг 1 - Код программы примера

```
#!/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):
        # 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,
            '-' * 32,
            '-' * 8
        )
        table.append(line)
        table.append(
            '| {:^4} | {:^30} | {:^32} | {:^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} | {:<32} | {:>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 - int(worker.year) >= int(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)

```

Добавим несколько записей в нашу условную таблицу при помощи команды `add`, запускающего метод добавления записи (фамилия и инициалы, должность и год поступления работника). Процесс представлен на рисунке 1.1.

```

>>> add
Фамилия и инициалы? Иванов И.И.
Должность? Менеджер проектов
Год поступления? 2020
>>> add
Фамилия и инициалы? Петрова А.А.
Должность? Аналитик
Год поступления? 2019
>>> add
Фамилия и инициалы? Сидоров С.С.
Должность? Программист
Год поступления? 2021
>>> add
Фамилия и инициалы? Кузнецова О.О.
Должность? Дизайнер

```

Рисунок 1.1 – Добавление новых записей

Выведем нашу таблицу для проверки корректности добавленных данных при помощи команды `list` (рисунок 1.2).

```
>>> list
```

№	Ф.И.О.	Должность	Год
1	Иванов И.И.	Инженер-исследователь	2020
2	Петрова А.А.	Менеджер по продажам	2021
3	Сидоров С.С.	Программист	2019
4	Кузнецов Д.Д.	Аналитик данных	2022
5	Васильева Е.Е.	Специалист по кадровым вопросам	2021
6	Федоров М.М.	Дизайнер интерфейсов	2020
7	Григорьев Н.Н.	Системный администратор	2018
8	Смирнова Ю.Ю.	Маркетолог	2022
9	Ковалев П.П.	Экономист	2019
10	Морозов А.А.	Разработчик ПО	2023

Рисунок 1.2 – Вывод существующих записей

Введём ещё 5 новых записей и выведем текущие данные в терминал (рисунок 1.3).

```
>>> list
```

№	Ф.И.О.	Должность	Год
1	Васильева Е.Е.	Специалист по кадровым вопросам	2021
2	Григорьев Н.Н.	Системный администратор	2018
3	Иванов Е.И.	Менеджер проектов	2018
4	Иванов И.И.	Инженер-исследователь	2020
5	Ковалев П.П.	Экономист	2019
6	Кузнецов Д.Д.	Аналитик данных	2022
7	Кузнецова Е.М.	Специалист по маркетингу	2021
8	Морозов А.А.	Разработчик ПО	2023
9	Петров А.В.	Аналитик данных	2020
10	Петрова А.А.	Менеджер по продажам	2021
11	Сидоров А.А.	Финансовый консультант	2017
12	Сидоров С.С.	Программист	2019
13	Смирнов Д.С.	Разработчик	2019
14	Смирнова Ю.Ю.	Маркетолог	2022
15	Федоров М.М.	Дизайнер интерфейсов	2020

Рисунок 1.3 – Вывод общего списка сотрудников

Сохраним имеющиеся данные в папке data в файле example\_data.json (рисунок 1.4).

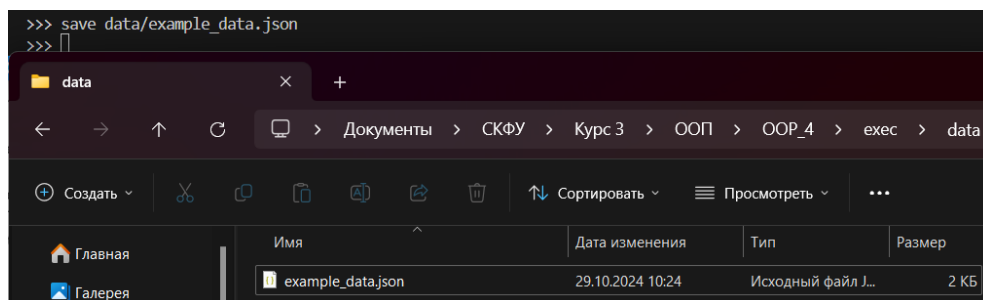


Рисунок 1.4 – Сохранение данных

Теперь можно завершить выполнение программы и посмотреть логи, созданные программой. В них записаны все действия оператора в течение всего выполнения программы (рисунок 1.5).

```
INFO:root:Отображен список сотрудников.
INFO:root:Добавлен сотрудник: Иванов Е.И., Менеджер проектов, поступивший в 2018 году.
ERROR:root:Ошибка: add петров а.в. -> Unknown command
INFO:root:Добавлен сотрудник: Петров А.В., Аналитик данных, поступивший в 2020 году.
INFO:root:Добавлен сотрудник: Смирнов Д.С., Разработчик, поступивший в 2019 году.
INFO:root:Добавлен сотрудник: Кузнецова Е.М., Специалист по маркетингу, поступивший в 2021 году.
INFO:root:Добавлен сотрудник: Сидоров А.А., Финансовый консультант, поступивший в 2017 году.
INFO:root:Отображен список сотрудников.
INFO:root:Сохранены данные в файл data/example_data.json.
```

Рисунок 1.5 – Логи программы



После завершения работы запустим программу вновь. Так как «память» программы обнулилась после её предыдущего завершения, нужно загрузить данные из файла, который был сохранён в предыдущий сеанс. Сделаем это при помощи команды `load`, указав путь до файла. Вновь выведем данные (рисунок 1.6).

```
>>> list
```

№	Ф.И.О.	Должность	Год
---	--------	-----------	-----

```
>>> load data_example.json
[Errno 2] No such file or directory: 'data_example.json'
>>> load data/example_data.json
>>> list
```

№	Ф.И.О.	Должность	Год
1	Васильева Е.Е.	Специалист по кадровым вопросам	2021
2	Григорьев Н.Н.	Системный администратор	2018
3	Иванов Е.И.	Менеджер проектов	2018
4	Иванов И.И.	Инженер-исследователь	2020
5	Ковалев П.П.	Экономист	2019
6	Кузнецов Д.Д.	Аналитик данных	2022
7	Кузнецова Е.М.	Специалист по маркетингу	2021
8	Морозов А.А.	Разработчик ПО	2023
9	Петров А.В.	Аналитик данных	2020
10	Петрова А.А.	Менеджер по продажам	2021
11	Сидоров А.А.	Финансовый консультант	2017
12	Сидоров С.С.	Программист	2019
13	Смирнов Д.С.	Разработчик	2019
14	Смирнова Ю.Ю.	Маркетолог	2022
15	Федоров М.М.	Дизайнер интерфейсов	2020

Рисунок 1.6 – Загрузка данных из файла и вывод

При помощи команды `select` выведем всех сотрудников, стаж которых будет удовлетворять указанному значению (рисунок 1.7).

```
>>> select 10
Работники с заданным стажем не найдены.
>>> select 5
1: Григорьев Н.Н.
2: Иванов Е.И.
3: Ковалев П.П.
4: Сидоров А.А.
5: Сидоров С.С.
6: Смирнов Д.С.
```

Рисунок 1.7 – Вывод сотрудников по стажу

После завершения работы вновь обратимся к логам, чтобы удостовериться, что каждое действие оператора записывается в текстовый файл (рисунок 1.8).

```
INFO:root:Загружены данные из файла data/example_data.json.  
WARNING:root:Работники со стажем более 10 лет не найдены.  
INFO:root:Найдено 6 работников со стажем более 5 лет.
```

Рисунок 1.8 – Новые записи в логах

Задание 1. Написать 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.

Листинг 2 – Код программы задания 1

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
"""  
Написать 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.  
"""  
  
if __name__ == '__main__':  
    # Вводим два значения a и b  
    a = input('Введите первое значение: ')  
    b = input('Введите второе значение: ')  
  
    try:  
        # Случай, если оба значения оказались числами  
        c = int(a) + int(b)  
        print('Оба значения оказались числами. Выполнение блока try...')  
        print(f'Результат суммирования: {a} + {b} = {c}')  
    except Exception:  
        # Случай, если как минимум одно из значений не число  
        c = str(a) + str(b)  
        print('Одно из значений оказалось не числом.',  
              'Выполнение блока except...')  
        print(f'Результат конкатенации: {c}')
```

В программе, код которой представлен в листинге 2, у нас по ходу выполнения может быть 2 случая: либо пользователь ввёл два числа, и они в результате выполнения суммируются, выполнив блок `try` (т.е. нормальное выполнение программы, без возникновения каких-либо исключений), либо пользователь ввёл как минимум в одном из значений не число, и программа соединит два полученных значения как строку (т.е. в процессе выполнения блока `try` возникла ошибка/исключение, и программа выполняет блока `except`). Оба варианта завершения программы представлены на рисунках 2.1 и 2.2.

```
Введите первое значение: 5
Введите второе значение: 10
Оба значения оказались числами. Выполнение блока try...
Результат суммирования: 5 + 10 = 15
```

Рисунок 2.1 – Нормальное выполнение программы при вводе чисел

```
Введите первое значение: 5
Введите второе значение: g
Одно из значений оказалось не числом. Выполнение блока except...
Результат конкатенации: 5g
```

Рисунок 2.2 – Выполнение участка кода, выполняющегося в случае возникновения ошибки/исключения

Задание 2. Написать программу, которая будет генерировать матрицу из случайных целых чисел. Пользователь может указать число строк и столбцов, а также диапазон чисел. Произвести обработку ошибок ввода пользователя.

Листинг 3 – Код программы задания 2

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

"""
Написать программу, которая будет генерировать матрицу из случайных целых
чисел. Пользователь может указать число строк и столбцов, а также диапазон
чисел. Произвести обработку ошибок ввода пользователя.
"""

import random
```

```

class MinValueBiggest(Exception):
    """
    Пустое исключение, вызываемое в случае, если минимальное значение
    чисел для случайной генерации оказалось больше максимального
    """
    pass

if __name__ == '__main__':
    """
    Основное тело программы
    """
    try:
        matrix_width = int(input("Введите ширину матрицы: "))
        matrix_height = int(input("Введите высоту матрицы: "))
        min_value = int(input("Введите минимальное значение диапазона: "))
        max_value = int(input("Введите максимальное значение диапазона: "))

        if min_value > max_value:
            raise MinValueBiggest # Вызов исключения

        matrix = [[0] * matrix_width for _ in range(matrix_height)]

        for i in range(matrix_height): # Заполнение матрицы
            for j in range(matrix_width):
                matrix[i][j] = random.randint(min_value, max_value)

        for i in range(matrix_height): # Вывод матрицы
            temp_str = str()
            j = 0
            for j in range(matrix_width):
                if j < matrix_width - 1: # Проверка на последнее значение
                    temp_str += str(matrix[i][j]) + " "
                else:
                    temp_str += str(matrix[i][j])
            j += 1
            print(f"|{temp_str}|")
        print("Завершение работы")

    except ValueError: # Ошибка при неправильном указании чисел
        print('Ошибка: Ошибка при вводе чисел')
    except MinValueBiggest: # Ошибка при указании min больше max
        print("Ошибка: Минимальное значение больше максимального")

```

Проведём несколько тестов программы, работающей в нормальном режиме, без намеренного вызова ошибок (рисунок 3.1 и 3.2).

```

Введите ширину матрицы: 5
Введите высоту матрицы: 5
Введите минимальное значение диапазона: 3
Введите максимальное значение диапазона: 5
|5 3 5 5 4|
|4 5 3 3 5|
|5 5 4 5 5|
|4 3 5 5 5|
|3 3 4 3 3|
Завершение работы

```

Рисунок 3.1 – Результат работы при правильной работе программы

```

Введите ширину матрицы: 9
Введите высоту матрицы: 5
Введите минимальное значение диапазона: 0
Введите максимальное значение диапазона: 9
|4 1 6 2 0 2 1 4 8|
|6 9 4 2 4 4 8 0 2|
|7 1 5 2 9 7 6 2 3|
|8 0 8 5 1 8 8 9 5|
|1 7 5 9 4 4 0 1 2|
Завершение работы

```

Рисунок 3.2 – Результат работы при правильной работе программы

Укажем одно из числовых значений не числом. В результате программа завершит работу, вызвав исключение `ValueError`, указав это в терминале (рисунок 3.3).

```

Введите ширину матрицы: s
Ошибка: Ошибка при вводе чисел

```

Рисунок 3.3 – Результат работы исключения

```

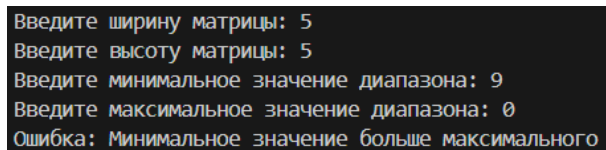
Введите ширину матрицы: 5
Введите высоту матрицы: 5
Введите минимальное значение диапазона: e
Ошибка: Ошибка при вводе чисел

```

Рисунок 3.4 – Результат работы исключения

Ещё одно место, где практически возможно совершить ошибку – указание минимального значения диапазона чисел больше, чем максимальное значение. Это исключение также подходит `ValueError`, но для

пользователя это будет не явный признак конкретно этой совершённой ошибки. Для этого в программа добавляется собственное исключение `MinValueBiggest`, которое далее по коду вызывается после проверки того, какое из значений больше. Если минимальное значение оказывается больше максимального, с помощью оператора `raise` вызывается исключение. Результат срабатывания исключения представлен на рисунке 3.5.

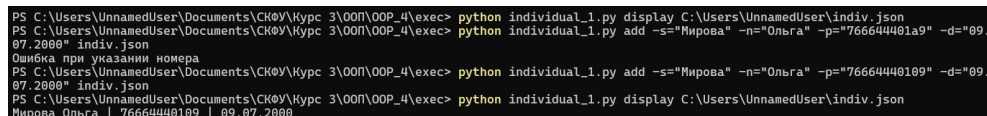


```
Введите ширину матрицы: 5
Введите высоту матрицы: 5
Введите минимальное значение диапазона: 9
Введите максимальное значение диапазона: 0
Ошибка: Минимальное значение больше максимального
```

Рисунок 3.5 – Вызов пользовательского исключения

Индивидуальное задание 1. Для индивидуального задания 1 лабораторной работы 2.19 добавить возможность работы с исключениями и логгирование.

Одно из самых банальным мест, где пользователь может допустить ошибку – добавление пользователя. Так как данная процедура требует ввода множества данных, шанс ошибки тут наибольший. Для имени и фамилии пользователя ограничения не нужно делать, но, например, для номера телефона стоит сделать проверку на правильность введённого значения. Проверка будет наиболее банальной – если указанный номер – число, то программа пропустит его. Но если в нём указаны какие-то дополнительные символы, то система вызовет исключение и запись не будет добавлена (рисунок 4.1).



```
PS C:\Users\UnnamedUser\Documents\CK0\Курс 3\00П\00Р_4\exec> python individual_1.py display C:\Users\UnnamedUser\indiv.json
PS C:\Users\UnnamedUser\Documents\CK0\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга" -p="766644401a9" -d="09.07.2000" indiv.json
Ошибка при указании номера
PS C:\Users\UnnamedUser\Documents\CK0\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга" -p="76664440109" -d="09.07.2000" indiv.json
PS C:\Users\UnnamedUser\Documents\CK0\Курс 3\00П\00Р_4\exec> python individual_1.py display C:\Users\UnnamedUser\indiv.json
Мирова Ольга | 76664440109 | 09.07.2000
```

Рисунок 4.1 – Попытка добавления работника с неправильным номером

Аналогично нужно реализовать исключение для даты. В ней не должно быть ничего, кроме как чисел указания дня, месяца и года, и разделительных символов – в нашем случае это точки. Результат представлен на рисунке 4.2.

```

PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга"
-r="78889996622" -d="09.07.1800" indiv.json
Ошибка при указании даты
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга"
-r="78889996622" -d="09.13.2000" indiv.json
Ошибка при указании даты
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга"
-r="78889996622" -d="35.07.2000" indiv.json
Ошибка при указании даты
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec> python individual_1.py add -s="Мирова" -n="Ольга"
-r="78889996622" -d="09.07.2000" indiv.json
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec> python individual_1.py display C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\00П\00Р_4\exec>
r\indiv.json
Мирова Ольга | 78889996622 | 09.07.2000

```

Рисунок 4.2 – Вызов ошибки при указании неправильной даты (или формата даты)

Для сохранения действий, выполненных пользователем при работе программы, необходимо добавить запись этих действий в текстовый файл – лог. Он позволит проверить выполненные действия, а также более ясно определить, что вызвало ошибку при выполнении программы. Пример лога представлен на рисунке 4.3.

```

INFO:root:Добавлен сотрудник: Мирова Ольга, поступивший 09.07.2000. Номер телефона: 78886663355
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
INFO:root:Добавлен сотрудник: Иванов Иван, поступивший 01.01.2010. Номер телефона: 76665557896
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
INFO:root:Выведен список сотрудников. Всего: 2
INFO:root:Добавлен сотрудник: Иванов Алексей, поступивший 15.03.2015. Номер телефона: 77775552211
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
INFO:root:Добавлен сотрудник: Смирнова Ольга, поступивший 22.07.2018. Номер телефона: 72221114488
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
INFO:root:Добавлен сотрудник: Петров Дмитрий, поступивший 10.01.2020. Номер телефона: 7444226172
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
INFO:root:Выведен список сотрудников. Всего: 5
INFO:root:Проведена выборка (период - 5). Результат: 4 записей.
INFO:root:Выведен список сотрудников. Всего: 4
INFO:root:Проведена выборка (период - 2). Результат: 5 записей.
INFO:root:Выведен список сотрудников. Всего: 5
INFO:root:Проведена выборка (период - 10). Результат: 2 записей.
INFO:root:Выведен список сотрудников. Всего: 2
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json
ERROR:root:Не удалось добавить запись: ошибка при указании даты
INFO:root:Сохранение файла по пути: C:\Users\UnnamedUser\indiv.json

```

Рисунок 4.3 – Лог программы

#### Листинг 4 – Код программы индивидуального задания 1

```

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

```

```
'''
```

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

```
'''
```

```

import json
from datetime import datetime
import argparse
import os.path
import pathlib

```

```
import logging

class DateError(Exception):
    pass

def print_help():
    """
    Функция вывода доступных пользователю команд.

    Аргументы:
        None

    Возвращает:
        None
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")

def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи в список.

    Аргументы:
        workers (list) - Список, в который будет добавляться запись
        surname (str) - Фамилия добавляемого сотрудника
        name (str) - Имя добавляемого сотрудника
        phone (int) - Номер телефона добавляемого сотрудника
        date (datetime) - Дата найма добавляемого сотрудника

    Возвращает:
        workers (list) - Обновлённый список сотрудников
    """

    try:
        phone = int(phone)
        if not isinstance(phone, int): # Проверка номера на правильность
            raise ValueError

        temp_data = str(date).split('.')
        for i in range(len(temp_data)):
            temp_data[i] = int(temp_data[i])
            if not isinstance(temp_data[i], int):
                raise DateError

        if (temp_data[0] < 0 or temp_data[0] > 31):
            raise DateError
```



```

if (temp_data[1] < 0 or temp_data[1] > 12):
    raise DateError
if (temp_data[2] < 1900 or temp_data[2] > 2024):
    raise DateError

workers.append(
    {
        "surname": surname,
        'name': name,
        'phone': phone,
        'date': date
    }
)
logging.info(
    f"Добавлен сотрудник: {surname} {name}, "
    f"поступивший {date}. Номер телефона: {phone}"
)

except ValueError:
    print("Ошибка при указании номера")
    logging.error("Не удалось добавить запись:"
                  "ошибка при указании номера телефона")
except DateError:
    print("Ошибка при указании даты")
    logging.error("Не удалось добавить запись: ошибка при указании даты")

return workers

def print_list(list):
    """
    Функция выводит на экран список всех существующих записей.

    Аргументы:
        list (list) - Список всех сотрудников (записей) для вывода

    Возвращает:
        None
    """
    count = 0

    for member in list:
        print(f"{member['surname']} {member['name']} | "
              f"{member['phone']} | {member['date']}")
        count += 1

    logging.info(f"Выведен список сотрудников. Всего: {count}")

def find_member(workers, period):
    """
    Функция для вывода на экран всех записей, чей стаж работы больше

```

или равен указанному.

Аргументы:

workers (list) - Список сотрудников для поиска  
period (int) - Количество лет стажа

Возвращает:

members (list) - Сипсок сотрудников, чей стаж больше  
или равен указанному

"""

count = 0

members = []

for member in workers:

year = datetime.strptime(member['date'], "%d.%m.%Y").year

if datetime.now().year - period >= year:

members.append(member)

count += 1

logging.info(f"Проведена выборка (период - {period}). "  
f"Результат: {count} записей.")

if count == 0:

print("Записи не найдены")

else:

return members

def get\_home\_path(filename):

"""

Получение полного пути к файлу, расположенному в домашнем каталоге  
пользователя (пример: C:/Users/<имя\_пользователя>/<имя\_файла>).

Аргументы:

filename (str) - Имя файла

Возвращает:

Полный путь к файлу (str)

"""

home\_dir = pathlib.Path.home()

return home\_dir / filename

def save\_file(filename, workers):

"""

Сохранение списка сотрудников в файл.

Аргументы:

filename (str) - Имя сохраняемого файла

workers (list) - Список сохраняемых сотрудников (записей)

Возвращает:

None

"""

try:

```
with open(get_home_path(filename), "w", encoding="utf-8") as file:
    logging.info(f"Сохранение файла по пути: {get_home_path(filename)}")
    json.dump(workers, file, ensure_ascii=False, indent=4)
```

except Exception as e:

```
logging.error(f"Ошибка при сохранении файла: {e}")
```

def load\_file(filename):

"""

Загрузка данных о сотрудниках из указанного JSON-файла.

Аргументы:

filename (str) - Имя загружаемого файла

Возвращает:

file (list) - список записей из файла

Пустой список - если произошла ошибка с при чтении файла

"""

try:

```
with open(get_home_path(filename), "r", encoding="utf-8") as file:
    return json.load(file)
```

except json.JSONDecodeError as e:

```
logging.error(f"Ошибка загрузки JSON: {e}")
```

```
print("Файл поврежден или имеет некорректный формат.")
```

```
return []
```

except FileNotFoundError:

```
logging.warning("Файл не найден. Создан пустой список.")
```

```
return []
```

def parse\_datetime(value):

"""

Преобразование указанного значения в формат даты.

Аргументы:

value (str) - указанная дата в текстовом формате

Возвращает:

Дата (datetime) - Преобразованное значение value в формат datetime

Текст 'Error' - В случае ошибки преобразования

"""

try:

```
return datetime.strptime(value, "%d.%m.%Y")
```

except ValueError:

```
print("Error")
```

def main(command\_line=None):

```
file_parser = argparse.ArgumentParser(add_help=False)
file_parser.add_argument(
    "filename",
    action="store",
    help="The data file name"
)

parser = argparse.ArgumentParser("workers")
parser.add_argument(
    "--version",
    action="version",
    version="% (prog)s 0.1.0"
)

subparsers = parser.add_subparsers(dest="command")

add = subparsers.add_parser(
    "add",
    parents=[file_parser],
    help="Add a new worker"
)
add.add_argument(
    "-s",
    "--surname",
    action="store",
    required=True,
    help="The worker's surname"
)
add.add_argument(
    "-n",
    "--name",
    action="store",
    required=True,
    help="The worker's name"
)
add.add_argument(
    "-p",
    "--phone",
    action="store",
    help="The worker's phone"
)
add.add_argument(
    "-d",
    "--date",
    action="store",
    required=True,
    help="The date of hiring"
)

_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
```

```

        help="Display all workers"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    args = parser.parse_args(command_line)

    is_dirty = False
    try:
        workers = load_file(args.filename)
    except FileExistsError:
        workers = []

    if args.command == "add":
        workers = add_worker(
            workers,
            args.surname,
            args.name,
            args.phone,
            args.date
        )
        is_dirty = True

    elif args.command == "display":
        print_list(workers)

    elif args.command == "select":
        selected = find_member(workers, args.period)
        print_list(selected)

    if is_dirty:
        save_file(args.filename, workers)

if __name__ == "__main__":
    """
    Основная программа
    """

    logging.basicConfig(

```

```
filename='individual_1.log',  
level=logging.INFO  
)  
  
main()
```

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.

Добавление даты и времени в модуле logging происходит при настройке базовой конфигурации этого модуля. В нашем случае в блоке основной программы мы уже имеем строки конфигурации модуля, в которых указываются имя файла, куда записываются логи, а также минимальный уровень логирования, который будет записываться в файл (в нашем случае минимальный уровень логирования – INFO, что значит, что все уровни, кроме DEBUG (так как по уровню он ниже), будут записываться в файл). Для того, чтобы добавить дату и время в логи, нам необходимо добавить ещё одну конфигурацию модуля – format, в котором мы изменим вывод информации, добавив к сообщению дату и время. Происходит это при помощи строки:

```
format='[%(asctime)s] %(message)s'
```

Этой строкой мы изменяем формат выводимых сообщений. `%(asctime)s` значит, что в сообщении будет выводиться дата и время, при этом время будет выводиться с точностью до миллисекунды. `%(message)s` – есть само выводимое сообщение. Как итог, при выводе списка сотрудников или при отборе по стажу мы получаем записи в логе, продемонстрированные на рисунке 5.1.

```
[2024-12-10 10:04:47,265] Выведен список сотрудников. Всего: 5  
[2024-12-10 10:15:04,187] Проведена выборка (период - 20). Результат: 1 записей.  
[2024-12-10 10:15:04,187] Выведен список сотрудников. Всего: 1
```

Рисунок 5.1 – Записи в файле лога

Эти записи были выведены в соответствии с введенными командами пользователем, указанные на рисунке 5.2.

```
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\ООП\ООР_4\exec> python individual_2.py display indiv.json
Мирова Ольга | 78886663355 | 09.07.2000
Иванов Иван | 76665557896 | 01.01.2010
Иванов Алексей | 77775552211 | 15.03.2015
Смирнова Ольга | 72221114488 | 22.07.2018
Петров Дмитрий | 74442226172 | 10.01.2020
PS C:\Users\UnnamedUser\Documents\СКФУ\Курс 3\ООП\ООР_4\exec> python individual_2.py select indiv.json -p=20
Мирова Ольга | 78886663355 | 09.07.2000
```

Рисунок 5.2 – Введенные команды

## Листинг 5 – Код программы индивидуального задания 2

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

'''
К предыдущему заданию (файл individual_1.py) добавить вывод в лог даты
и времени записей с точностью до миллисекунду.
'''

import json
from datetime import datetime
import argparse
import os.path
import pathlib
import logging

class DateError(Exception):
    pass

def print_help():
    """
    Функция вывода доступных пользователю команд.

    Аргументы:
        None

    Возвращает:
        None
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")

def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи в список.
```

Аргументы:

workers (list) - Список, в который будет добавляться запись  
surname (str) - Фамилия добавляемого сотрудника  
name (str) - Имя добавляемого сотрудника  
phone (int) - Номер телефона добавляемого сотрудника  
date (datetime) - Дата найма добавляемого сотрудника

Возвращает:

workers (list) - Обновлённый список сотрудников  
""

try:

```
phone = int(phone)
if not isinstance(phone, int): # Проверка номера на правильность
    raise ValueError
```

```
temp_data = str(date).split('.')
for i in range(len(temp_data)):
    temp_data[i] = int(temp_data[i])
    if not isinstance(temp_data[i], int):
        raise DateError
```

```
if (temp_data[0] < 0 or temp_data[0] > 31):
    raise DateError
if (temp_data[1] < 0 or temp_data[1] > 12):
    raise DateError
if (temp_data[2] < 1900 or temp_data[2] > 2024):
    raise DateError
```

```
workers.append(
    {
        "surname": surname,
        'name': name,
        'phone': phone,
        'date': date
    }
)
logging.info(
    f"Добавлен сотрудник: {surname} {name}, "
    f"поступивший {date}. Номер телефона: {phone}"
)
```

except ValueError:

```
print("Ошибка при указании номера")
logging.error("Не удалось добавить запись:"
              "ошибка при указании номера телефона")
```

except DateError:

```
print("Ошибка при указании даты")
logging.error("Не удалось добавить запись: ошибка при указании даты")
```

return workers



```

def print_list(list):
    """
    Функция выводит на экран список всех существующих записей.

    Аргументы:
        list (list) - Список всех сотрудников (записей) для вывода

    Возвращает:
        None
    """
    count = 0

    for member in list:
        print(f"{member['surname']} {member['name']} | "
              f"{member['phone']} | {member['date']}")
        count += 1

    logging.info(f"Выведен список сотрудников. Всего: {count}")

def find_member(workers, period):
    """
    Функция для вывода на экран всех записей, чей стаж работы больше
    или равен указанному.

    Аргументы:
        workers (list) - Список сотрудников для поиска
        period (int) - Количество лет стажа

    Возвращает:
        members (list) - Список сотрудников, чей стаж больше
        или равен указанному
    """
    count = 0
    members = []

    for member in workers:
        year = datetime.strptime(member['date'], "%d.%m.%Y").year
        if datetime.now().year - period >= year:
            members.append(member)
            count += 1

    logging.info(f"Проведена выборка (период - {period}). "
                  f"Результат: {count} записей.")

    if count == 0:
        print("Записи не найдены")
    else:
        return members

```

```
def get_home_path(filename):  
    """
```

Получение полного пути к файлу, расположенному в домашнем каталоге пользователя (пример: C:/Users/<имя\_пользователя>/<имя\_файла>).

Аргументы:

filename (str) - Имя файла

Возвращает:

Полный путь к файлу (str)

```
    """
```

```
    home_dir = pathlib.Path.home()  
    return home_dir / filename
```

```
def save_file(filename, workers):  
    """
```

Сохранение списка сотрудников в файл.

Аргументы:

filename (str) - Имя сохраняемого файла

workers (list) - Список сохраняемых сотрудников (записей)

Возвращает:

None

```
    """
```

```
    try:
```

```
        with open(get_home_path(filename), "w", encoding="utf-8") as file:
```

```
            logging.info(f"Сохранение файла по пути: {get_home_path(filename)}")
```

```
            json.dump(workers, file, ensure_ascii=False, indent=4)
```

```
    except Exception as e:
```

```
        logging.error(f"Ошибка при сохранении файла: {e}")
```

```
def load_file(filename):  
    """
```

Загрузка данных о сотрудниках из указанного JSON-файла.

Аргументы:

filename (str) - Имя загружаемого файла

Возвращает:

file (list) - список записей из файла

Пустой список - если произошла ошибка с при чтении файла

```
    """
```

```
    try:
```

```
        with open(get_home_path(filename), "r", encoding="utf-8") as file:
```

```
            return json.load(file)
```

```
    except json.JSONDecodeError as e:
```

```
        logging.error(f"Ошибка загрузки JSON: {e}")
```

```

    print("Файл поврежден или имеет некорректный формат.")
    return []
except FileNotFoundError:
    logging.warning("Файл не найден. Создан пустой список.")
    return []

def parse_datetime(value):
    """
    Преобразование указанного значения в формат даты.

    Аргументы:
        value (str) - указанная дата в текстовом формате

    Возвращает:
        Дата (datetime) - Преобразованное значение value в формат datetime
        Текст 'Error' - В случае ошибки преобразования
    """
    try:
        return datetime.strptime(value, "%d.%m.%Y")
    except ValueError:
        print("Error")

def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )

    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The worker's surname"
    )

```

```

    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="The worker's phone"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="The date of hiring"
    )

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

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    args = parser.parse_args(command_line)

    is_dirty = False
    try:
        workers = load_file(args.filename)
    except FileExistsError:
        workers = []

    if args.command == "add":
        workers = add_worker(

```

```

        workers,
        args.surname,
        args.name,
        args.phone,
        args.date
    )
    is_dirty = True

elif args.command == "display":
    print_list(workers)

elif args.command == "select":
    selected = find_member(workers, args.period)
    print_list(selected)

if is_dirty:
    save_file(args.filename, workers)

if __name__ == "__main__":
    """
    Основная программа
    """

    logging.basicConfig(
        filename='individual_2.log',
        level=logging.INFO,
        format='[%(asctime)s] %(message)s'
    )

    main()

```

#### Ответы на контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

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

Синтаксические ошибки возникают в случае, если программа написана с нарушением требований Python к синтаксису. Определяются они в процессе парсинга программы.

Исключения возникают в случае, если синтаксически программа корректна, но в процессе выполнения возникает ошибка (деление на ноль и т.д.).

2. Как осуществляется обработка исключений в языке программирования Python?

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

Пример:

```
print("start")
try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except Exception as e:
    print("Error! " + str(e))
print("stop")
```

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

Для выполнения определённого программного кода при выходе из блока `try/except` используется оператор `finally`.

Независимо от того, возникнет исключение или нет, код в блоке `finally` всё равно будет выполнен.

Если необходимо выполнить какой-то программный код, в случае если в процессе выполнения блока `try` не возникло исключений, то можно использовать оператор `else`.

4. Как осуществляется генерация исключений в языке Python?

Для принудительной генерации исключения используется инструкция `raise`. Пример:

```
try:  
    raise Exception("Some exception")  
except Exception as e:  
    print("Exception exception " + str(e))
```

Таким образом можно «вручную» вызывать исключения при необходимости.

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 программа.

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

```
class NegValException(Exception)  
    pass  
  
try:  
    val = int(input("input positive number: "))  
    if val < 0:  
        raise NegValException("Neg val: " + str(val))  
    print(val + 10)  
except NegValException as e:  
    print(e)
```

6. Каково назначение модуля `logging`?

С помощью `logging` на Python можно записывать в лог исключения. Обычно лог пишется в файл. Уровень `INFO` указывает, что сообщения

уровней ниже не будут отображаться в логе. Python позволяет в try-excerpt получить текст ошибки.

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

В модуле logging присутствуют следующие уровни логгирования:

Уровень	Числовое значение	Применение
CRITICAL	50	Серьёзная ошибка, обозначающая, что программа не может продолжить своё выполнение
ERROR	40	Из-за серьёзных проблем программа не может выполнять некоторые функции
WARNING	30	Индикация какого-либо неожиданного события или проблемы в ближайшем будущем (к примеру «на диске мало места»). Программа продолжает свою работу
INFO	20	Индикация штатного выполнения программы (например, действие пользователя или инициализация каких-либо модулей)
DEBUG	10	Детальная информация, нужная для диагностики программы
NOTSET	0	Не указанный явно уровень логгирования, который будет наследовать уровень логгирования от своего родительского логгера

**Выводы:** В процессе выполнения лабораторной работы были приобретены навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x, приобретены навыки по работе с логгированием, а также проработаны примеры, выполнены общие и индивидуальные задания.

Ссылка на репозиторий GitHub:

[IUnnamedUserI/OOP\\_4: Объектно-ориентированное программирование.](#)  
[Лабораторная работа №4](#)