

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 №5**  
**дисциплины**  
**«Объектно–ориентированное программирование»**  
**Вариант 13**

Выполнил:  
Рябинин Егор Алексеевич  
3 курс, группа ИВТ–б–о–23–2,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Проверил:  
Доцент департамента цифровых,  
робототехнических систем и  
электроники института перспективной  
инженерии  
Воронкин Роман Александрович

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г

**Тема:** Работа с исключениями в языке Python.

**Цель:** приобретение навыков по работе с исключениями при написании программ с помощью языка программирования Python версии 3.13.3.

**Порядок выполнения работы:**

**Ссылка на репозиторий:**

[https://github.com/bohemiaaaaa/Lab6\\_Object-oriented-programming](https://github.com/bohemiaaaaa/Lab6_Object-oriented-programming)

**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.

**Листинг программы:**

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

def main() -> None:
    try:
        first: str = input("Первое значение: ")
        second: str = input("Второе значение: ")

        first_num: float = float(first)
        second_num: float = float(second)

        result: float = first_num + second_num
        print(f"Результат: {result}")
    except ValueError:
        result: str = first + second
        print(f"Результат: {result}")

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

```
PS C:\Users\4isto\00P_lab5> python tasks/task1.py
Первое значение: 1
Второе значение: 2
Результат: 3.0
PS C:\Users\4isto\00P_lab5> python tasks/task1.py
Первое значение: 2
Второе значение: в
Результат: 2в
```

Рисунок 1 – Результат работы программы

**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.

Листинг программы:

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

import random
from typing import List

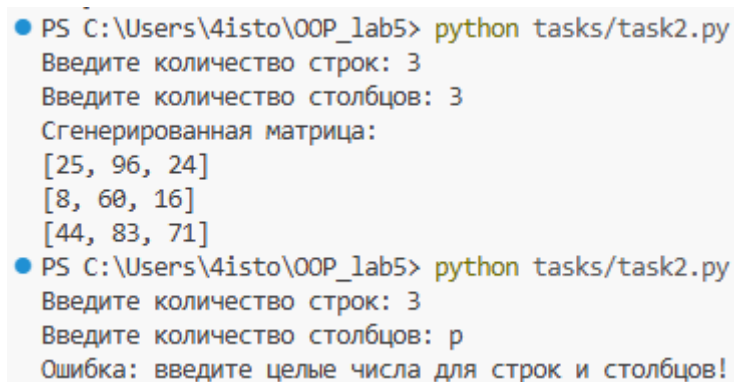
def main() -> None:
    try:
        rows: int = int(input("Введите количество строк: "))
        cols: int = int(input("Введите количество столбцов: "))

        matrix: List[List[int]] = []
        for i in range(rows):
            row: List[int] = []
            for j in range(cols):
                row.append(random.randint(1, 100))
            matrix.append(row)

        print("Сгенерированная матрица:")
        for row in matrix:
            print(row)

    except ValueError:
        print("Ошибка: введите целые числа для строк и столбцов!")

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



```
PS C:\Users\4isto\OOP_lab5> python tasks/task2.py
Введите количество строк: 3
Введите количество столбцов: 3
Сгенерированная матрица:
[25, 96, 24]
[8, 60, 16]
[44, 83, 71]
PS C:\Users\4isto\OOP_lab5> python tasks/task2.py
Введите количество строк: 3
Введите количество столбцов: p
Ошибка: введите целые числа для строк и столбцов!
```

Рисунок 2 – Результат работы программы

**Задание 3.** Проверка уникальности имени пользователя

Пусть имеется список уже занятых имен пользователей. При попытке регистрации проверьте, не содержится ли новое имя в списке. Если да,

выбросите исключение `UsernameAlreadyExistsError`, содержащее имя и сообщение: `UsernameAlreadyExistsError: 'alex' -> имя уже занято`. Если нет, добавьте имя в список и подтвердите регистрацию.

Листинг программы:

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

class UsernameAlreadyExistsError(Exception):
    def __init__(self, username: str, message: str = "имя уже занято") ->
None:
    self.username: str = username
    self.message: str = message
    super(UsernameAlreadyExistsError, self).__init__(message)

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

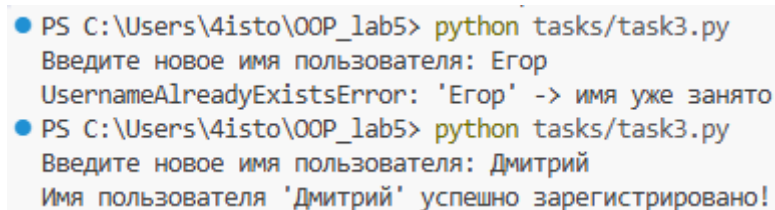
# Список занятых имен
existing_usernames: list[str] = ["Егор", "админ", "администратор", "admin"]

def main() -> None:
    try:
        new_username: str = input("Введите новое имя пользователя: ")

        if new_username in existing_usernames:
            raise UsernameAlreadyExistsError(new_username)
        else:
            existing_usernames.append(new_username)
            print(f"Имя пользователя '{new_username}' успешно
зарегистрировано!")

    except UsernameAlreadyExistsError as e:
        print(e)

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



```
PS C:\Users\4isto\OOP_lab5> python tasks/task3.py
Введите новое имя пользователя: Егор
UsernameAlreadyExistsError: 'Егор' -> имя уже занято
PS C:\Users\4isto\OOP_lab5> python tasks/task3.py
Введите новое имя пользователя: Дмитрий
Имя пользователя 'Дмитрий' успешно зарегистрировано!
```

Рисунок 3 – Результат работы программы

#### Задание 4. Телефонный справочник с поиском по фамилии

Разработайте консольное приложение для ведения телефонного справочника. Каждая запись должна содержать фамилию, имя, номер

телефона и дату рождения. Программа должна поддерживать добавление новых записей, сортировку по первым трем цифрам номера телефона и поиск информации по введенной фамилии. Если такой записи нет, нужно вывести соответствующее сообщение. Создайте пользовательское исключение для проверки корректности телефонного номера. Все события и ошибки должны регистрироваться в журнале логов.

Структура файлов проекта имеет вид:

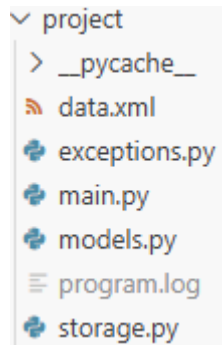


Рисунок 4 – Структура файлов проекта

Листинг программы exceptions.py:

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

class InvalidPhoneError(Exception):
    def __init__(
        self, phone: str, message: str = "Некорректный номер телефона"
    ) -> None:
        self.phone = phone
        self.message = message
        super().__init__(f"{phone} -> {message}")

class UnknownCommandError(Exception):
    def __init__(self, command: str, message: str = "Неизвестная команда") ->
None:
        self.command = command
        self.message = message
        super().__init__(f"{command} -> {message}")

class DataFormatError(Exception):
    def __init__(
        self, filename: str, message: str = "Некорректная структура файла"
    ) -> None:
        self.filename = filename
        self.message = message
        super().__init__(f"{filename} -> {message}")
```

Листинг программы models.py:

```

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

from dataclasses import dataclass, field
from typing import List

@dataclass(frozen=True)
class Contact:
    last_name: str
    first_name: str
    phone: str
    birth_date: str

@dataclass
class PhoneBook:
    contacts: List[Contact] = field(default_factory=list)

    def add(self, last_name: str, first_name: str, phone: str, birth_date:
str) -> None:
        from exceptions import InvalidPhoneError

        if not phone.isdigit() or len(phone) != 11:
            raise InvalidPhoneError(phone)

        self.contacts.append(
            Contact(
                last_name=last_name,
                first_name=first_name,
                phone=phone,
                birth_date=birth_date,
            )
        )

        self.contacts.sort(key=lambda contact: contact.phone[:3])

    def select(self, last_name: str) -> List[Contact]:
        result: List[Contact] = [
            contact
            for contact in self.contacts
            if contact.last_name.lower() == last_name.lower()
        ]
        return result

    def __str__(self) -> str:
        if not self.contacts:
            return "Телефонный справочник пуст"

        table: List[str] = []
        line: str = "+{ }-+-{ }-+-{ }-+-{ }-+-{ }-+-{ }-+-".format(
            "-" * 4, "-" * 20, "-" * 15, "-" * 15, "-" * 12
        )
        table.append(line)
        table.append(
            "| {:^4} | {:^20} | {:^15} | {:^15} | {:^12} |".format(
                "№", "Фамилия", "Имя", "Телефон", "Дата рождения"
            )
        )
        table.append(line)

        for idx, contact in enumerate(self.contacts, 1):
            table.append(
                "| {:>4} | {:<20} | {:<15} | {:<15} | {:<12} |".format(

```

```

        idx,
        contact.last_name,
        contact.first_name,
        contact.phone,
        contact.birth_date,
    )
    table.append(line)

return "\n".join(table)

```

### Листинг программы storage.py:

```

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

import xml.etree.ElementTree as ET

from exceptions import DataFormatError
from models import Contact, PhoneBook

def save_phonebook(phonebook: PhoneBook, filename: str) -> None:
    root = ET.Element("phonebook")

    for contact in phonebook.contacts:
        contact_element = ET.Element("contact")

        last_name_element = ET.SubElement(contact_element, "last_name")
        last_name_element.text = contact.last_name

        first_name_element = ET.SubElement(contact_element, "first_name")
        first_name_element.text = contact.first_name

        phone_element = ET.SubElement(contact_element, "phone")
        phone_element.text = contact.phone

        birth_date_element = ET.SubElement(contact_element, "birth_date")
        birth_date_element.text = contact.birth_date

        root.append(contact_element)

    tree = ET.ElementTree(root)
    with open(filename, "wb") as fout:
        tree.write(fout, encoding="utf-8", xml_declaration=True)

def load_phonebook(filename: str) -> PhoneBook:
    try:
        with open(filename, "r", encoding="utf8") as fin:
            xml = fin.read()

        parser = ET.XMLParser(encoding="utf8")
        tree = ET.fromstring(xml, parser=parser)

        phonebook = PhoneBook()

        for contact_element in tree:
            last_name: str = ""
            first_name: str = ""
            phone: str = ""
            birth_date: str = ""

```

```

        for element in contact_element:
            if element.tag == "last_name":
                last_name = element.text or ""
            elif element.tag == "first_name":
                first_name = element.text or ""
            elif element.tag == "phone":
                phone = element.text or ""
            elif element.tag == "birth_date":
                birth_date = element.text or ""

        if all([last_name, first_name, phone, birth_date]):
            phonebook.contacts.append(
                Contact(
                    last_name=last_name,
                    first_name=first_name,
                    phone=phone,
                    birth_date=birth_date,
                )
            )

        phonebook.contacts.sort(key=lambda contact: contact.phone[:3])
        return phonebook

    except Exception as e:
        raise DataFormatError(filename) from e

```

### Листинг программы main.py:

```

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

import logging
import os

from exceptions import DataFormatError, InvalidPhoneError,
UnknownCommandError
from models import PhoneBook
from storage import load_phonebook, save_phonebook

os.makedirs("tasks/project", exist_ok=True)

logging.basicConfig(
    filename="tasks/project/program.log",
    level=logging.INFO,
    format="%(asctime)s %(levelname)s: %(message)s",
)

def main() -> None:
    phonebook: PhoneBook = PhoneBook()

    try:
        phonebook = load_phonebook("tasks/project/data.xml")
        print("Данные загружены автоматически")
        logging.info("Данные загружены автоматически при запуске")
    except (DataFormatError, FileNotFoundError):
        print("Файл данных не найден или поврежден, начинаем с пустой базы")
        logging.info("Начало работы с пустой базой данных")

    print("Телефонный справочник")
    print("Введите 'help' для просмотра команд")

    while True:

```



```

try:
    command: str = input(">>> ").lower().strip()

    if command == "exit":
        print("Завершение работы...")
        break

    elif command == "add":
        last_name: str = input("Фамилия: ")
        first_name: str = input("Имя: ")
        phone: str = input("Номер телефона (11 цифр): ")
        birth_date: str = input("Дата рождения (дд.мм.гггг): ")

        phonebook.add(last_name, first_name, phone, birth_date)
        print("Контакт успешно добавлен")
        log_msg = f"Добавлен: {last_name} {first_name}"
        logging.info(log_msg)

    elif command == "list":
        print(phonebook)
        logging.info("Отображен список контактов")

    elif command.startswith("select"):
        parts: list[str] = command.split(" ", maxsplit=1)
        if len(parts) > 1:
            last_name: str = parts[1]
            selected = phonebook.select(last_name)

            if selected:
                msg = f"Найдено с фамилией '{last_name}':
{len(selected)}"

                print(msg)
                for idx, contact in enumerate(selected, 1):
                    contact_info = (
                        f"{idx}: {contact.first_name} "
                        f"{contact.last_name}, тел: {contact.phone},

                        f"рожд: {contact.birth_date}"
                    )
                    print(contact_info)
                log_msg = f"Найдено {len(selected)} контактов"
                logging.info(log_msg)
            else:
                msg = f"Контакты с фамилией '{last_name}' не найдены"
                print(msg)
                logging.warning(msg)
        else:
            error_msg = "Ошибка: используйте 'select <фамилия>'"
            print(error_msg)
            logging.error(error_msg)

    elif command.startswith("save "):
        parts: list[str] = command.split(" ", maxsplit=1)
        if len(parts) > 1:
            filename: str = "tasks/project/" + parts[1]
            save_phonebook(phonebook, filename)
            print(f"Данные сохранены в файл: {filename}")
            logging.info(f"Данные сохранены в файл: {filename}")
        else:
            error_msg = "Ошибка: используйте 'save <имя_файла>'"
            print(error_msg)
            logging.error(error_msg)

    elif command.startswith("load "):

```

```

        parts: list[str] = command.split(" ", maxsplit=1)
        if len(parts) > 1:
            filename: str = "tasks/project/" + parts[1]
            phonebook = load_phonebook(filename)
            print(f"Данные загружены из файла: {filename}")
            logging.info(f"Данные загружены из файла: {filename}")
        else:
            error_msg = "Ошибка: используйте 'load <имя_файла>'"
            print(error_msg)
            logging.error(error_msg)

    elif command == "help":
        print("Список команд:")
        print("add - добавить контакт")
        print("list - показать все контакты")
        print("select <фамилия> - найти по фамилии")
        print("save <файл> - сохранить в XML")
        print("load <файл> - загрузить из XML")
        print("help - показать справку")
        print("exit - выйти")

    else:
        raise UnknownCommandError(command)

except InvalidPhoneError as e:
    error_msg = f"InvalidPhoneError: {e}"
    print(f"Ошибка: {e}")
    logging.error(error_msg)
except UnknownCommandError as e:
    error_msg = f"UnknownCommandError: {e}"
    print(f"Ошибка: {e}")
    logging.error(error_msg)
except DataFormatError as e:
    error_msg = f"DataFormatError: {e}"
    print(f"Ошибка формата данных: {e}")
    logging.error(error_msg)
except Exception as e:
    error_msg = f"Unexpected error: {e}"
    print(f"Неизвестная ошибка: {e}")
    logging.error(error_msg)

if __name__ == "__main__":
    main()

```

Результат работы программы:

```

>>> add
Фамилия: Петров
Имя: Петр
Номер телефона (11 цифр): 88005553535
Дата рождения (дд.мм.гггг): 01.12.2020
Контакт успешно добавлен

```

Рисунок 5 – Добавление записи

```

>>> save data.xml
Данные сохранены в файл: tasks/project/data.xml

```

Рисунок 6 – Сохранение записи в xml-файл

```
>>> list
```

№	Фамилия	Имя	Телефон	Дата рождения
1	Петров	Петр	88005553535	01.12.2020
2	Рябинин	Егор	89097580527	05.12.2005
3	тест	тест	89097580527	02.09.2001
4	Петров	Петр	89993482939	03.04.99

Рисунок 7 – Вывод всех записей

```
>>> select Рябинин
Найдено с фамилией 'рябинин': 1
1: Егор Рябинин, тел: 89097580527, рожд: 05.12.2005
```

Рисунок 8 – Извлечение записи

```
>>> help
Список команд:
add - добавить контакт
list - показать все контакты
select <фамилия> - найти по фамилии
save <файл> - сохранить в XML
load <файл> - загрузить из XML
help - показать справку
exit - выйти
```

Рисунок 9 – Вызов справки

## Тесты для написанных программ:

### Листинг программы для задания 1:

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

from task1 import main as addition_main

def test_addition_numbers(monkeypatch, capsys):
    """Тест сложения чисел"""
    inputs = iter(["5", "3"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    addition_main()
    captured = capsys.readouterr()
    assert "Результат: 8.0" in captured.out

def test_addition_strings(monkeypatch, capsys):
    """Тест конкатенации строк"""
    inputs = iter(["hello", "world"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    addition_main()
    captured = capsys.readouterr()
    assert "Результат: helloworld" in captured.out

def test_addition_mixed(monkeypatch, capsys):
```

```

"""Тест смешанных типов"""
inputs = iter(["5", "world"])
monkeypatch.setattr("builtins.input", lambda _: next(inputs))

addition_main()
captured = capsys.readouterr()
assert "Результат: 5world" in captured.out

```

## Листинг программы для задания 2:

```

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

from task2 import main as matrix_main

def test_matrix_creation(monkeypatch, capsys):
    """Тест создания матрицы"""
    inputs = iter(["2", "3"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    matrix_main()
    captured = capsys.readouterr()

    assert "Сгенерированная матрица:" in captured.out
    # Проверяем что выведено 2 строки матрицы
    lines = [line for line in captured.out.split("\n") if
line.strip().startswith("[")]
    assert len(lines) == 2

def test_matrix_invalid_input(monkeypatch, capsys):
    """Тест неверного ввода"""
    inputs = iter(["abc", "3"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    matrix_main()
    captured = capsys.readouterr()

    assert "Ошибка: введите целые числа для строк и столбцов!" in
captured.out

```

## Листинг программы для задания 3:

```

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

import pytest
from task3 import UsernameAlreadyExistsError, existing_usernames
from task3 import main as username_main

def test_username_already_exists():
    """Тест исключения при существующем имени"""
    with pytest.raises(UsernameAlreadyExistsError) as exc_info:
        raise UsernameAlreadyExistsError("Ероп")

    assert "Ероп" in str(exc_info.value)
    assert "UsernameAlreadyExistsError" in str(exc_info.value)

```

```

def test_username_registration_success(monkeypatch, capsys):
    """Тест успешной регистрации"""
    original_length = len(existing_usernames)
    inputs = iter(["NewUser"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    username_main()
    captured = capsys.readouterr()

    assert "успешно зарегистрировано" in captured.out
    assert "NewUser" in existing_usernames
    assert len(existing_usernames) == original_length + 1

def test_username_registration_failure(monkeypatch, capsys):
    """Тест неудачной регистрации"""
    inputs = iter(["Ероп"])
    monkeypatch.setattr("builtins.input", lambda _: next(inputs))

    username_main()
    captured = capsys.readouterr()

    assert "UsernameAlreadyExistsError" in captured.out
    assert "Ероп" in captured.out

```

#### Листинг программы для задания 4:

```

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

import os
import tempfile

import pytest
from exceptions import DataFormatError, InvalidPhoneError
from models import Contact, PhoneBook
from storage import load_phonebook, save_phonebook

def test_contact_creation():
    """Тест создания контакта"""
    contact = Contact("Иванов", "Иван", "79101234567", "01.01.1990")
    assert contact.last_name == "Иванов"
    assert contact.first_name == "Иван"
    assert contact.phone == "79101234567"
    assert contact.birth_date == "01.01.1990"

def test_phonebook_add_valid():
    """Тест добавления корректного контакта"""
    phonebook = PhoneBook()
    phonebook.add("Петров", "Петр", "79051234567", "15.05.1985")

    assert len(phonebook.contacts) == 1
    assert phonebook.contacts[0].last_name == "Петров"
    assert phonebook.contacts[0].phone == "79051234567"

def test_phonebook_add_invalid_phone():
    """Тест добавления контакта с некорректным телефоном"""
    phonebook = PhoneBook()

```

```

with pytest.raises(InvalidPhoneError):
    phonebook.add("Сидоров", "Алексей", "abc", "20.10.1990")

def test_phonebook_add_short_phone():
    """Тест добавления контакта с коротким телефоном"""
    phonebook = PhoneBook()

    with pytest.raises(InvalidPhoneError):
        phonebook.add("Сидоров", "Алексей", "123", "20.10.1990")

def test_phonebook_select():
    """Тест поиска по фамилии"""
    phonebook = PhoneBook()
    phonebook.add("Иванов", "Иван", "79101234567", "01.01.1990")
    phonebook.add("Петров", "Петр", "79051234567", "15.05.1985")
    phonebook.add("Иванов", "Мария", "79211234567", "10.10.1995")

    result = phonebook.select("Иванов")
    assert len(result) == 2
    assert all(contact.last_name == "Иванов" for contact in result)

def test_phonebook_sorting():
    """Тест сортировки по первым трем цифрам телефона"""
    phonebook = PhoneBook()
    phonebook.add("Иванов", "Иван", "79211234567", "01.01.1990") # 792
    phonebook.add("Петров", "Петр", "79051234567", "15.05.1985") # 790
    phonebook.add("Сидоров", "Алексей", "79101234567", "20.10.1990") # 791

    # Должны быть отсортированы: 790, 791, 792
    assert phonebook.contacts[0].phone == "79051234567"
    assert phonebook.contacts[1].phone == "79101234567"
    assert phonebook.contacts[2].phone == "79211234567"

def test_save_and_load_phonebook():
    """Тест сохранения и загрузки телефонной книги"""
    with tempfile.NamedTemporaryFile(mode="w", suffix=".xml", delete=False)
as f:
    temp_filename = f.name

    try:
        phonebook = PhoneBook()
        phonebook.add("Иванов", "Иван", "79101234567", "01.01.1990")
        phonebook.add("Петров", "Петр", "79051234567", "15.05.1985")

        save_phonebook(phonebook, temp_filename)

        loaded_phonebook = load_phonebook(temp_filename)

        assert len(loaded_phonebook.contacts) == 2

        assert loaded_phonebook.contacts[0].last_name == "Петров"
        assert loaded_phonebook.contacts[0].phone == "79051234567"
        assert loaded_phonebook.contacts[1].last_name == "Иванов"
        assert loaded_phonebook.contacts[1].phone == "79101234567"

    finally:
        if os.path.exists(temp_filename):
            os.unlink(temp_filename)

```

```

def test_load_invalid_xml():
    """Тест загрузки некорректного XML"""
    with tempfile.NamedTemporaryFile(mode="w", suffix=".xml", delete=False)
as f:
    f.write("Это не XML файл")
    temp_filename = f.name

    try:
        with pytest.raises(DataFormatError):
            load_phonebook(temp_filename)
    finally:
        if os.path.exists(temp_filename):
            os.unlink(temp_filename)

def test_phonebook_str_empty():
    """Тест строкового представления пустой телефонной книги"""
    phonebook = PhoneBook()
    result = str(phonebook)
    assert "Телефонный справочник пуст" in result

def test_phonebook_str_with_contacts():
    """Тест строкового представления с контактами"""
    phonebook = PhoneBook()
    phonebook.add("Иванов", "Иван", "79101234567", "01.01.1990")
    result = str(phonebook)

    assert "Иванов" in result
    assert "Иван" in result
    assert "79101234567" in result
    assert "01.01.1990" in result

```

```

tests/test_addition.py::test_addition_numbers PASSED
tests/test_addition.py::test_addition_strings PASSED
tests/test_addition.py::test_addition_mixed PASSED
tests/test_matrix.py::test_matrix_creation PASSED
tests/test_matrix.py::test_matrix_invalid_input PASSED
tests/test_phonebook.py::test_contact_creation PASSED
tests/test_phonebook.py::test_phonebook_add_valid PASSED
tests/test_phonebook.py::test_phonebook_add_invalid_phone PASSED
tests/test_phonebook.py::test_phonebook_add_short_phone PASSED
tests/test_phonebook.py::test_phonebook_select PASSED
tests/test_phonebook.py::test_phonebook_sorting PASSED
tests/test_phonebook.py::test_save_and_load_phonebook PASSED
tests/test_phonebook.py::test_load_invalid_xml PASSED
tests/test_phonebook.py::test_phonebook_str_empty PASSED
tests/test_phonebook.py::test_phonebook_str_with_contacts PASSED
tests/test_username.py::test_username_already_exists PASSED
tests/test_username.py::test_username_registration_success PASSED
tests/test_username.py::test_username_registration_failure PASSED

===== 18 passed in 0.10s

```

Рисунок 10 – Результат работы тестов

### Контрольные вопросы:

#### 1. Какие существуют виды ошибок в языке программирования Python?

Основные виды ошибок в Python делятся на синтаксические ошибки (SyntaxError), которые возникают при нарушении правил синтаксиса Python,

и исключения (Exceptions), которые возникают во время выполнения программы даже если синтаксис корректен. Примеры исключений: `ZeroDivisionError` (деление на ноль), `TypeError` (несовместимый тип данных), `ValueError` (некорректное значение), `NameError` (использование необъявленной переменной), `IndexError` (обращение по несуществующему индексу), `KeyError` (обращение по несуществующему ключу в словаре) и `FileNotFoundError` (файл не найден).

## **2. Как осуществляется обработка исключений в языке программирования Python?**

Обработка исключений в Python осуществляется с помощью конструкций `try` и `except`. Код, который может вызвать исключение, помещается в блок `try`. Если исключение происходит, управление передается в блок `except`, где указывается, как обработать эту ошибку. Можно обрабатывать конкретные типы исключений, перечисляя их после `except`.

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

Блок `else` выполняется только в том случае, если исключение в блоке `try` не было вызвано. Он используется для кода, который должен выполниться при успешном выполнении блока `try`. Блок `finally` выполняется всегда, независимо от того, было исключение или нет. Он используется для обязательных действий по очистке ресурсов, например, закрытия файлов или сетевых подключений, чтобы гарантировать, что эти операции выполнятся в любом случае.

## **4. Как осуществляется генерация исключений в языке Python?**

Генерация исключений в Python осуществляется с помощью оператора `raise`. После оператора `raise` указывается объект исключения, который нужно вызвать. Это может быть встроенное исключение или пользовательский класс исключения. Например, `raise ValueError("Некорректное значение")` или `raise MyCustomException()`.

## **5. Как создаются классы пользовательский исключений в языке Python?**



Пользовательские исключения создаются путем определения нового класса, который наследуется от встроенного класса `Exception` или одного из его подклассов. Обычно такой класс остается пустым или содержит строку документации, чтобы описать ошибку. Например: `class MyError(Exception): pass`. Это позволяет создавать специализированные типы ошибок для конкретных ситуаций в программе.

## **6. Каково назначение модуля `logging`?**

Назначение модуля `logging` – это предоставление гибкой системы ведения журналов (логов) для приложений на Python. Он позволяет записывать сообщения отладки, информации, предупреждения и ошибки в различные выходные потоки, такие как консоль, файлы, системный журнал. Это помогает отслеживать события, происходящие во время выполнения программы, и диагностировать проблемы.

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

Модуль `logging` поддерживает следующие стандартные уровни логирования (в порядке возрастания серьезности): `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`. `DEBUG` используется для диагностики и отладки, например, вывод значения переменной в цикле. `INFO` для подтверждения того, что программа работает как надо, например, сообщение "Приложение запущено успешно". `WARNING` для указания на потенциальную проблему, которая пока не является ошибкой, например, предупреждение о низком объеме дискового пространства. `ERROR` для сообщений об ошибках, которые привели к сбою части программы, например, ошибка подключения к базе данных. `CRITICAL` для сообщений о критических ошибках, которые могут привести к полной остановке программы, например, исчерпание памяти.

**Вывод:** в ходе лабораторной работы были приобретены навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.13.3.