

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 №6
дисциплины
«Объектно-ориентированное программирование»
Вариант № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
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 Р.А

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

Тема: 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.

Репозиторий: https://github.com/bickrosss/TM_oop_lab6

Порядок выполнения работы:

1. Базовые примеры обработки исключений.

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

2. Обработка конкретных исключений.

```
print("start")
try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except (ValueError, ZeroDivisionError):
    print("Error!")
print("stop")
```

3. Раздельная обработка исключений.

```
print("start")
try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except ValueError:
    print("ValueError!")
except ZeroDivisionError:
    print("ZeroDivisionError!")
except:
    print("Error!")
print("Stop")
```

4. Обработка с получением объекта исключения.

```
print("start")
```

```

try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except ValueError as ve:
    print("ValueError! {0}".format(ve))
except ZeroDivisionError as zde:
    print("ZeroDivisionError! {0}".format(zde))
except Exception as ex:
    print("Error! {0}".format(ex))
print("Stop")

```

5. Использование finally для гарантированного выполнения кода.

```

try:
    val = int(input("input number: "))
    tmp = 10 / val
    print(tmp)
except:
    print("Exception")
finally:
    print("Finally code")

```

6. Использование else для выполнения кода при успешной операции.

```

try:
    f = open("tmp.txt", "r")
    for line in f:
        print(line)
    f.close()
except Exception as e:
    print(e)
else:
    print("File was readed")

```

7. Вложенные блоки try-except-else.

```

print("Program started")
try:
    print("Opening file...")
    f = open("data.txt", "w")
    try:
        print("Writing to file...")
        f.write("Hello World!")
    except Exception:
        print("Something gone wrong!")
    else:
        print("Success!")
except FileNotFoundError:
    print("File not found!")

```

```
print("Program finished")
```

8. Применение пользовательских исключений.

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

# Пользовательское исключение
class MegValException(Exception):
    pass

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

9. Логирование исключений и событий.

```
import logging

# Настройка логов
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!")

# Логирование исключений в файл
logging.basicConfig(filename="log.txt", level=logging.INFO)
try:
    print(10 / 0)
except Exception as e:
    logging.error(str(e))
```

10. Практическое применение исключений в программе управления персоналом.

```
import logging
import os
import sys
import xml.etree.ElementTree as ET
from dataclasses import dataclass, field
```

```
from datetime import date
```

```
class IllegalYearError(Exception):
```

```
    def __init__(self, year: int, message: str = "Illegal year number") -> None:
        self.year = year
        self.message = message
        super().__init__(self.message)

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

```
class UnknownCommandError(Exception):
```

```
    def __init__(self, command: str, message: str = "Unknown command") -> None:
        self.command = command
        self.message = message
        super().__init__(self.message)

    def __str__(self) -> str:
        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: str, post: str, year: int) -> None:
        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) -> str:
        if not self.workers:
            return "Нет данных о работниках"
```

```
        table = []
```

```

line = f"+{'-' * 6}+{'-' * 34}+{'-' * 24}+{'-' * 12}+"
table.append(line)

table.append(f"| {'№':^4} | {'Ф.И.О':^32} | {'Должность':^22} |
{'Год':^10} |")
table.append(line)

for idx, worker in enumerate(self.workers, 1):

    name = worker.name[:30] + ".." if len(worker.name) > 32 else
worker.name
    post = worker.post[:20] + ".." if len(worker.post) > 22 else
worker.post

    table.append(f"| {idx:>4} | {name:<32} | {post:<22} |
{worker.year:>10} |")

    table.append(line)
return "\n".join(table)

def select(self, period: int) -> list[Worker]:
    today = date.today()

    results = []
    for worker in self.workers:
        if today.year - worker.year >= period:
            results.append(worker)
    return results

def load(self, filename: str) -> None:
    with open(filename, "r", encoding="utf-8") as fin:
        xml = fin.read()
    parser = ET.XMLParser(encoding="utf-8")
    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 or "0")

        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: str) -> None:
    folder = "xml_files"

```

```

os.makedirs(folder, exist_ok=True)
path = os.path.join(folder, 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(path, "wb") as fout:
    tree.write(fout, encoding="utf-8", xml_declaration=True)

if __name__ == "__main__":
    logging.basicConfig(
        filename="worker.log",
        level=logging.INFO,
    )

    staff = Staff()

    while True:
        try:
            command = input(">>> ").lower()

            if command == "exit":
                break

            elif command == "add":
                name = input("Ф.И.О: ")
                post = input("Должность: ")
                year = int(input("Год приема на работу: "))
                staff.add(name, post, year)
                logging.info(f"Добавлен работник: {name}, {post}, {year}")
            elif command == "list":
                print(staff)

            elif command.startswith("select "):
                parts = command.split(maxsplit=1)
                selected = staff.select(int(parts[1]))
                if selected:

```

```

        for idx, worker in enumerate(selected, 1):
            print(
                "{:>4}: {}".format(
                    idx,
                    worker.name,
                )
            )
        logging.info(
            f"Выборка работников со стажем работы более {parts[1]}"
        )
    elif command.startswith("load "):
        parts = command.split(maxsplit=1)
        staff.load(parts[1])
        logging.info(f"Загружены данные из файла {parts[1]}")
        print(f"Данные из файла {parts[1]} успешно загружены:")
        print(staff)
    elif command.startswith("save "):
        parts = command.split(maxsplit=1)
        staff.save(parts[1])
        logging.info(f"Сохранены данные в файл {parts[1]}")
    elif command == "help":
        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 err:
    logging.error(err)
    print(err, file=sys.stderr)

```

11. Выполнили задание 1.

Создайте класс User, в котором хранится логин и флаг authenticated.

При попытке вызвать метод `access_resource()` без авторизации выбросите исключение `UnauthorizedAccessError`, содержащее имя пользователя.

Пример:

`UnauthorizedAccessError: 'guest' -> доступ запрещён.`

После успешного входа в систему метод должен возвращать: «Доступ разрешён.»

Пользовательские исключения:

```

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

class UnauthorizedAccessError(Exception):
    """Исключение при попытке доступа без авторизации."""
    def __init__(self, username, message="доступ запрещён"):
        self.username = username
        self.message = message
        super().__init__(f'{username}' -> {message})

class InvalidLoginError(Exception):
    """Исключение при некорректном логине."""
    def __init__(self, login, message="некорректный логин"):
        self.login = login
        self.message = message
        super().__init__(f'{login}' -> {message})

class UnknownCommandError(Exception):
    """Исключение при вводе неизвестной команды."""
    def __init__(self, command, message="неизвестная команда"):
        self.command = command
        self.message = message
        super().__init__(f'{command}' -> {message})

class DataFormatError(Exception):
    """Исключение при некорректном формате данных."""
    def __init__(self, filename, message="некорректный формат данных"):
        self.filename = filename
        self.message = message
        super().__init__(f'{filename}' -> {message})

class InvalidPasswordError(Exception):
    """Исключение при некорректном пароле."""
    def __init__(self, message="некорректный пароль"):
        self.message = message
        super().__init__(message)

```

Модели данных:

```

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

from dataclasses import dataclass, field
from typing import List, Optional
import hashlib
import secrets
import string

```

```
from tasks.task1.exceptions import UnauthorizedAccessError, InvalidLoginError, InvalidPasswordError
```

```
def hash_password(password: str, salt: str) -> str:
    """Хэширование пароля с солью."""
    return hashlib.sha256((password + salt).encode()).hexdigest()
```

```
def generate_salt(length: int = 16) -> str:
    """Генерация случайной соли."""
    alphabet = string.ascii_letters + string.digits
    return ''.join(secrets.choice(alphabet) for _ in range(length))
```

```
@dataclass(frozen=True)
```

```
class User:
```

```
    """Класс пользователя."""
```

```
    login: str
```

```
    password_hash: str
```

```
    salt: str
```

```
    authenticated: bool = False
```

```
def verify_password(self, password: str) -> bool:
```

```
    """Проверить пароль."""
```

```
    return hash_password(password, self.salt) == self.password_hash
```

```
def access_resource(self) -> str:
```

```
    """Метод для доступа к ресурсу."""
```

```
    if not self.authenticated:
```

```
        raise UnauthorizedAccessError(self.login)
```

```
    return f"Пользователь '{self.login}': Доступ разрешён."
```

```
def authenticate(self, password: str) -> None:
```

```
    """Метод для аутентификации пользователя."""
```

```
    if not self.verify_password(password):
```

```
        raise InvalidLoginError(self.login, "неверный пароль")
```

```
    self._set_authenticated(True)
```

```
def logout(self) -> None:
```

```
    """Метод для выхода из системы."""
```

```
    self._set_authenticated(False)
```

```
def _set_authenticated(self, value: bool) -> None:
```

```
    """Внутренний метод для изменения состояния аутентификации."""
```

```
    object.__setattr__(self, 'authenticated', value)
```

```
def change_password(self, old_password: str, new_password: str) -> None:
```

```
    """Сменить пароль пользователя."""
```

```
    if not self.verify_password(old_password):
```

```
        raise InvalidLoginError(self.login, "старый пароль неверен")
```

```

        if len(new_password) < 6:
            raise InvalidPasswordError("пароль должен быть не менее 6 символов")

        # Генерируем новую соль для повышения безопасности
        new_salt = generate_salt()
        new_hash = hash_password(new_password, new_salt)

        object.__setattr__(self, 'password_hash', new_hash)
        object.__setattr__(self, 'salt', new_salt)

@dataclass
class UserManager:
    """Класс для управления коллекцией пользователей."""
    users: List[User] = field(default_factory=list)

    def add_user(self, login: str, password: str) -> User:
        """Добавить нового пользователя."""
        # Проверка логина
        if not login or len(login) < 3:
            raise InvalidLoginError(login, "логин должен быть не менее 3
символов")

        if not login.isalnum():
            raise InvalidLoginError(login, "логин должен содержать только буквы и
цифры")

        # Проверка пароля
        if len(password) < 6:
            raise InvalidPasswordError("пароль должен быть не менее 6 символов")

        # Проверка уникальности логина
        for user in self.users:
            if user.login == login:
                raise InvalidLoginError(login, "логин уже существует")

        # Создание пользователя с хэшированным паролем
        salt = generate_salt()
        password_hash = hash_password(password, salt)

        user = User(
            login=login,
            password_hash=password_hash,
            salt=salt,
            authenticated=False
        )
        self.users.append(user)
        return user

    def find_user(self, login: str) -> Optional[User]:

```

```

        """Найти пользователя по логину."""
        for user in self.users:
            if user.login == login:
                return user
        return None

def get_user_or_raise(self, login: str) -> User:
    """Найти пользователя или вызвать исключение."""
    user = self.find_user(login)
    if user is None:
        raise InvalidLoginError(login, "пользователь не найден")
    return user

def authenticate_user(self, login: str, password: str) -> User:
    """Аутентифицировать пользователя."""
    user = self.get_user_or_raise(login)
    user.authenticate(password)
    return user

def logout_user(self, login: str) -> User:
    """Выйти из системы."""
    user = self.get_user_or_raise(login)
    user.logout()
    return user

def change_user_password(self, login: str, old_password: str, new_password:
str) -> User:
    """Сменить пароль пользователя."""
    user = self.get_user_or_raise(login)
    user.change_password(old_password, new_password)
    return user

def get_authenticated_users(self) -> List[User]:
    """Получить список аутентифицированных пользователей."""
    return [user for user in self.users if user.authenticated]

def get_all_users(self) -> List[User]:
    """Получить список всех пользователей."""
    return self.users

def sort_users(self) -> None:
    """Отсортировать пользователей по логину."""
    self.users.sort(key=lambda user: user.login)

def __str__(self) -> str:
    """Представление всех пользователей в виде таблицы."""
    if not self.users:
        return "Нет пользователей."

    # Заголовок таблицы
    table = []

```

```

line = '+{}+{}+{}+'.format(
    '-' * 4,
    '-' * 30,
    '-' * 15
)
table.append(line)
table.append(
    '| {:^4} | {:^30} | {:^15} |'.format(
        "№",
        "Логин",
        "Аутентифицирован"
    )
)
table.append(line)

# Данные пользователей
for idx, user in enumerate(self.users, 1):
    status = "Да" if user.authenticated else "Нет"
    table.append(
        '| {:^4} | {:^30} | {:^15} |'.format(
            idx,
            user.login,
            status
        )
    )
table.append(line)
return '\n'.join(table)

```

Работа с XML-хранилищем:

```

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

import xml.etree.ElementTree as ET
from typing import List
from tasks.task1.models import User
from tasks.task1.exceptions import DataFormatError

class UserStorage:
    """Класс для сохранения и загрузки пользователей в XML."""

    @staticmethod
    def save(users: List[User], filename: str) -> None:
        """Сохранить пользователей в XML файл."""
        root = ET.Element('users')

        for user in users:
            user_element = ET.Element('user')

            login_element = ET.SubElement(user_element, 'login')

```

```

login_element.text = user.login

password_hash_element = ET.SubElement(user_element, 'password_hash')
password_hash_element.text = user.password_hash

salt_element = ET.SubElement(user_element, 'salt')
salt_element.text = user.salt

authenticated_element = ET.SubElement(user_element, 'authenticated')
authenticated_element.text = str(user.authenticated).lower()

root.append(user_element)

tree = ET.ElementTree(root)
with open(filename, 'wb') as fout:
    fout.write(b'<?xml version="1.0" encoding="utf-8"?>\n')
    tree.write(fout, encoding='utf-8')

@staticmethod
def load(filename: str) -> List[User]:
    """Загрузить пользователей из XML файла."""
    try:
        tree = ET.parse(filename)
        root = tree.getroot()
    except (ET.ParseError, FileNotFoundError) as e:
        raise DataFormatError(filename, f"ошибка чтения файла: {e}")

    users = []

    for user_element in root:
        login = None
        password_hash = None
        salt = None
        authenticated = False

        for element in user_element:
            if element.tag == 'login':
                login = element.text
            elif element.tag == 'password_hash':
                password_hash = element.text
            elif element.tag == 'salt':
                salt = element.text
            elif element.tag == 'authenticated':
                try:
                    authenticated = element.text.lower() == 'true'
                except AttributeError:
                    authenticated = False

        if all([login, password_hash, salt]):
            # Создаем нового пользователя с нужным состоянием
            user = User(

```

```

        login=login,
        password_hash=password_hash,
        salt=salt,
        authenticated=authenticated
    )
    users.append(user)
else:
    raise DataFormatError(filename, "неполные данные пользователя")

return users

```

ЛИСТИНГ ВЫЗОВА:

```

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

import logging
import sys
from pathlib import Path

if __name__ == "__main__":
    project_root = Path(__file__).resolve().parents[2]
    if str(project_root) not in sys.path:
        sys.path.insert(0, str(project_root))

import getpass
from models import UserManager
from storage import UserStorage
from tasks.task1.exceptions import (
    UnauthorizedAccessError,
    InvalidLoginError,
    UnknownCommandError,
    DataFormatError,
    InvalidPasswordError
)

def setup_logging() -> None:
    """Настройка системы логирования."""
    logging.basicConfig(
        filename='auth_system.log',
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s',
        encoding='utf-8'
    )

def print_help() -> None:
    """Вывод справки по командам."""
    help_text = """
Доступные команды:

```

add <логин>	- Добавить нового пользователя (запросит пароль)
auth <логин>	- Аутентифицировать пользователя (запросит пароль)
logout <логин>	- Выйти из системы
access <логин>	- Проверить доступ к ресурсу
changepass <логин>	- Сменить пароль (запросит старый и новый пароли)
list	- Показать всех пользователей
list_auth	- Показать аутентифицированных пользователей
save <файл.xml>	- Сохранить пользователей в XML
load <файл.xml>	- Загрузить пользователей из XML
help	- Показать эту справку
exit	- Выйти из программы

Требования к паролю:

- Не менее 6 символов
- Рекомендуется использовать буквы, цифры и специальные символы

"""

```
print(help_text)
```

```
def get_password(prompt: str = "Пароль: ") -> str:
    """Безопасный ввод пароля."""
    return getpass.getpass(prompt)
```

```
def main() -> None:
    """Основная функция программы."""
    setup_logging()
    user_manager = UserManager()

    print("Система управления пользователями с безопасной аутентификацией")
    print("=" * 60)
    print_help()

    while True:
        try:
            # Ввод команды
            command_input = input("\n>>> ").strip()
            if not command_input:
                continue

            parts = command_input.split()
            command = parts[0].lower()

            # Обработка команд
            if command == 'exit':
                logging.info("Завершение работы программы.")
                print("До свидания!")
                break

            elif command == 'help':
                print_help()
```

```

logging.info("Вывод справки.")

elif command == 'add' and len(parts) >= 2:
    login = parts[1]
    print(f"Добавление пользователя: {login}")
    password = get_password("Введите пароль (мин. 6 символов): ")
    confirm = get_password("Повторите пароль: ")

    if password != confirm:
        print("Ошибка: пароли не совпадают!", file=sys.stderr)
        continue

    user = user_manager.add_user(login, password)
    print(f"✓ Пользователь '{login}' успешно добавлен.")
    logging.info(f"Добавлен пользователь: {login}")

elif command == 'auth' and len(parts) >= 2:
    login = parts[1]
    password = get_password(f"Пароль для {login}: ")
    user = user_manager.authenticate_user(login, password)
    print(f"✓ Пользователь '{login}' успешно аутентифицирован.")
    logging.info(f"Аутентифицирован пользователь: {login}")

elif command == 'logout' and len(parts) >= 2:
    login = parts[1]
    user = user_manager.logout_user(login)
    print(f"✓ Пользователь '{login}' вышел из системы.")
    logging.info(f"Выход пользователя: {login}")

elif command == 'access' and len(parts) >= 2:
    login = parts[1]
    user = user_manager.get_user_or_raise(login)
    result = user.access_resource()
    print(f"✓ {result}")
    logging.info(f"Проверка доступа для пользователя: {login}")

elif command == 'changepass' and len(parts) >= 2:
    login = parts[1]
    print(f"Смена пароля для пользователя: {login}")
    old_password = get_password("Старый пароль: ")
    new_password = get_password("Новый пароль (мин. 6 символов): ")
    confirm = get_password("Повторите новый пароль: ")

    if new_password != confirm:
        print("Ошибка: новые пароли не совпадают!", file=sys.stderr)
        continue

    user = user_manager.change_user_password(login, old_password,
new_password)
    print(f"✓ Пароль для пользователя '{login}' успешно изменен.")
    logging.info(f"Смена пароля для пользователя: {login}")

```

```

elif command == 'list':
    user_manager.sort_users()
    print("\n" + str(user_manager))
    logging.info("Вывод списка всех пользователей.")

elif command == 'list_auth':
    auth_users = user_manager.get_authenticated_users()
    if auth_users:
        print("\nАутентифицированные пользователи:")
        for idx, user in enumerate(auth_users, 1):
            print(f"{idx:>4}: {user.login}")
        logging.info(f"Вывод {len(auth_users)} аутентифицированных
пользователей.")
    else:
        print("\nНет аутентифицированных пользователей.")
        logging.info("Нет аутентифицированных пользователей.")

elif command == 'save' and len(parts) >= 2:
    filename = parts[1]
    UserStorage.save(user_manager.get_all_users(), filename)
    print(f"✓ Данные сохранены в файл: {filename}")
    logging.info(f"Сохранение данных в файл: {filename}")

elif command == 'load' and len(parts) >= 2:
    filename = parts[1]
    users = UserStorage.load(filename)
    user_manager.users = users
    user_manager.sort_users()
    print(f"✓ Данные загружены из файла: {filename}")
    print(f"  Загружено пользователей: {len(users)}")
    logging.info(f"Загрузка данных из файла: {filename}")

else:
    raise UnknownCommandError(command_input)

except KeyboardInterrupt:
    print("\n\nПрограмма прервана пользователем.")
    logging.warning("Программа прервана пользователем (Ctrl+C).")
    break

except UnknownCommandError as e:
    print(f"\nX Ошибка: {e}", file=sys.stderr)
    logging.error(f"Неизвестная команда: {e}")

except InvalidLoginError as e:
    print(f"\nX Ошибка авторизации: {e}", file=sys.stderr)
    logging.error(f"Ошибка авторизации: {e}")

except UnauthorizedAccessError as e:
    print(f"\nX Ошибка доступа: {e}", file=sys.stderr)

```

```

logging.error(f"Ошибка доступа: {e}")

except InvalidPasswordError as e:
    print(f"\nX Ошибка пароля: {e}", file=sys.stderr)
    logging.error(f"Ошибка пароля: {e}")

except DataFormatError as e:
    print(f"\nX Ошибка данных: {e}", file=sys.stderr)
    logging.error(f"Ошибка данных: {e}")

except Exception as e:
    print(f"\nX Непредвиденная ошибка: {e}", file=sys.stderr)
    logging.error(f"Непредвиденная ошибка: {e}", exc_info=True)

if __name__ == '__main__':
    main()

```

@bickrosss → /workspaces/ТМ_oop_lab6 (main) \$ python tasks/task1/main.py

Система управления пользователями с безопасной аутентификацией

=====

Доступные команды:

add <логин>	- Добавить нового пользователя (запросит пароль)
auth <логин>	- Аутентифицировать пользователя (запросит пароль)
logout <логин>	- Выйти из системы
access <логин>	- Проверить доступ к ресурсу
changepass <логин>	- Сменить пароль (запросит старый и новый пароли)
list	- Показать всех пользователей
list_auth	- Показать аутентифицированных пользователей
save <файл.xml>	- Сохранить пользователей в XML
load <файл.xml>	- Загрузить пользователей из XML
help	- Показать эту справку
exit	- Выйти из программы

Требования к паролю:

- Не менее 6 символов
- Рекомендуется использовать буквы, цифры и специальные символы

>>> exit

До свидания!

Рисунок 1. Результат выполнения программы

12. Решили задание 2.

Разработать консольное приложение для ведения базы данных товаров, продающихся в разных магазинах.

Каждая запись должна включать название товара, магазин и цену в рублях. Программа должна обеспечивать сортировку по названию магазина, добавление записей и выборку всех товаров, доступных в указанном пользователем магазине. Если такого магазина нет - вывести сообщение об его отсутствии. При вводе некорректной цены должно возбуждаться пользовательское исключение.

Все действия программы и ошибки должны логироваться.

Пользовательские исключения:

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

class InvalidPriceError(Exception):
    """Исключение при некорректной цене."""
    def __init__(self, price, message="некорректная цена"):
        self.price = price
        self.message = message
        super().__init__(f"{'price'}' -> {message}")

class StoreNotFoundError(Exception):
    """Исключение при поиске несуществующего магазина."""
    def __init__(self, store, message="магазин не найден"):
        self.store = store
        self.message = message
        super().__init__(f"{'store'}' -> {message}")

class UnknownCommandError(Exception):
    """Исключение при вводе неизвестной команды."""
    def __init__(self, command, message="неизвестная команда"):
        self.command = command
        self.message = message
        super().__init__(f"{'command'}' -> {message}")

class DataFormatError(Exception):
    """Исключение при некорректном формате данных."""
    def __init__(self, filename, message="некорректный формат данных"):
        self.filename = filename
        self.message = message
        super().__init__(f"{'filename'}' -> {message}")
```

Модели данных:

```
#!/usr/bin/env python3
```

```

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

from dataclasses import dataclass, field
from typing import List
from tasks.task2.exceptions import InvalidPriceError, StoreNotFoundError

@dataclass(frozen=True)
class Product:
    """Класс товара."""
    name: str
    store: str
    price: float

    def __post_init__(self):
        """Валидация цены после инициализации."""
        if self.price <= 0:
            raise InvalidPriceError(self.price, "цена должна быть больше 0")

@dataclass
class ProductCatalog:
    """Класс для управления каталогом товаров."""
    products: List[Product] = field(default_factory=list)

    def add(self, name: str, store: str, price: float) -> Product:
        """Добавить новый товар."""
        # Валидация ввода
        if not name or not store:
            raise ValueError("Название товара и магазин не могут быть пустыми")

        if price <= 0:
            raise InvalidPriceError(price, "цена должна быть больше 0")

        product = Product(name=name, store=store, price=price)
        self.products.append(product)
        self.products.sort(key=lambda p: p.store) # Сортировка по магазину
        return product

    def select(self, store: str) -> List[Product]:
        """Выбрать все товары в указанном магазине."""
        if not store:
            raise ValueError("Название магазина не может быть пустым")

        result = [p for p in self.products if p.store.lower() == store.lower()]

        if not result:
            raise StoreNotFoundError(store)

        return result

```

```

def get_stores(self) -> List[str]:
    """Получить список всех уникальных магазинов."""
    return sorted(set(p.store for p in self.products))

def __str__(self) -> str:
    """Представление всех товаров в виде таблицы."""
    if not self.products:
        return "Каталог товаров пуст."

    table = []
    line = '+{}+{}+{}+{}+'.format('-' * 4, '-' * 25, '-' * 20, '-' * 12)
    table.append(line)
    table.append(' | {:^4} | {:^25} | {:^20} | {:^12} | '.format(
        "№", "Название товара", "Магазин", "Цена, руб.))
    table.append(line)

    for idx, product in enumerate(self.products, 1):
        table.append(' | {:^4} | {:^25} | {:^20} | {:^12.2f} | '.format(
            idx, product.name[:25], product.store[:20], product.price))

    table.append(line)
    return '\n'.join(table)

```

Работа с XML-хранилищем:

```

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

```

```

import xml.etree.ElementTree as ET
from typing import List
from tasks.task2.models import Product
from tasks.task2.exceptions import DataFormatError, InvalidPriceError

```

```

class ProductStorage:
    """Класс для сохранения и загрузки товаров в XML."""

    @staticmethod
    def save(products: List[Product], filename: str) -> None:
        """Сохранить товары в XML файл."""
        root = ET.Element('products')

        for product in products:
            product_element = ET.Element('product')

            ET.SubElement(product_element, 'name').text = product.name
            ET.SubElement(product_element, 'store').text = product.store
            ET.SubElement(product_element, 'price').text = str(product.price)

            root.append(product_element)

```

```

tree = ET.ElementTree(root)
with open(filename, 'wb') as fout:
    fout.write(b'<?xml version="1.0" encoding="utf-8"?>\n')
    tree.write(fout, encoding='utf-8')

@staticmethod
def load(filename: str) -> List[Product]:
    """Загрузить товары из XML файла."""
    try:
        tree = ET.parse(filename)
        root = tree.getroot()
    except (ET.ParseError, FileNotFoundError) as e:
        raise DataFormatError(filename, f"ошибка чтения файла: {e}")

    products = []

    for product_element in root:
        try:
            # Простой подход, который вызывает AttributeError если элемент не
            найден

            name = product_element.find('name').text
            store = product_element.find('store').text
            price = float(product_element.find('price').text)

            product = Product(name=name, store=store, price=price)
            products.append(product)

        except AttributeError as e:
            # AttributeError возникает если .find() вернет None или .text
            будет None
            raise DataFormatError(filename, f"ошибка данных: {e}")
        except ValueError as e:
            # ValueError возникает при ошибке преобразования float
            raise DataFormatError(filename, f"ошибка данных: {e}")
        except InvalidPriceError as e:
            raise DataFormatError(filename, f"некорректная цена: {e}")

    return products

```

Листинг программы:

```

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

import logging
import sys
from pathlib import Path
if __name__ == "__main__":
    project_root = Path(__file__).resolve().parents[2]
    if str(project_root) not in sys.path:

```

```

        sys.path.insert(0, str(project_root))
from tasks.task2.models import ProductCatalog
from tasks.task2.storage import ProductStorage
from tasks.task2.exceptions import *

def print_help() -> None:
    """Вывод справки по командам."""
    help_text = """
Доступные команды:
    add <товар> <магазин> <цена> - Добавить товар
    list                          - Показать все товары
    stores                        - Показать все магазины
    select <магазин>              - Показать товары в магазине
    save <файл.xml>               - Сохранить в XML
    load <файл.xml>               - Загрузить из XML
    help                          - Показать справку
    exit                          - Выйти
    """
    print(help_text)

def main() -> None:
    """Главная функция программы."""
    # Настройка логирования
    logging.basicConfig(
        filename='catalog.log',
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s'
    )

    catalog = ProductCatalog()

    print("Каталог товаров")
    print("=" * 40)
    print_help()

    while True:
        try:
            # Ввод команды
            command = input(">>> ").strip().lower()

            if command == 'exit':
                logging.info("Завершение работы.")
                print("Выход.")
                break

            elif command == 'help':
                print_help()
                logging.info("Вывод справки.")

```

```

elif command == 'list':
    print(catalog)
    logging.info("Вывод списка товаров.")

elif command == 'stores':
    stores = catalog.get_stores()
    if stores:
        print("Магазины:")
        for store in stores:
            print(f" - {store}")
        logging.info(f"Список магазинов: {len(stores)}")
    else:
        print("Нет магазинов.")
        logging.info("Нет магазинов.")

elif command.startswith('add '):
    parts = command.split(maxsplit=3)
    if len(parts) != 4:
        print("Используйте: add товар магазин цена")
        continue

    name, store = parts[1], parts[2]
    try:
        price = float(parts[3])
        catalog.add(name, store, price)
        print(f"Добавлен: {name} в {store} за {price} руб.")
        logging.info(f"Добавлен товар: {name}")
    except InvalidPriceError as e:
        print(f"Ошибка: {e}")
        logging.error(f"Некорректная цена: {parts[3]}")
    except ValueError as e:
        print(f"Ошибка: {e}")

elif command.startswith('select '):
    store = command.split(maxsplit=1)[1]
    try:
        products = catalog.select(store)
        print(f"Товары в '{store}':")
        for p in products:
            print(f" - {p.name}: {p.price} руб.")
        logging.info(f"Выборка магазина '{store}': {len(products)}
товаров")
    except StoreNotFoundError:
        print(f"Магазин '{store}' не найден.")
        logging.warning(f"Магазин не найден: {store}")

elif command.startswith('save '):
    filename = command.split(maxsplit=1)[1]
    ProductStorage.save(catalog.products, filename)
    print(f"Сохранено в {filename}")
    logging.info(f"Сохранение в {filename}")

```

```

elif command.startswith('load '):
    filename = command.split(maxsplit=1)[1]
    products = ProductStorage.load(filename)
    catalog.products = products
    catalog.products.sort(key=lambda p: p.store)
    print(f"Загружено из {filename}: {len(products)} товаров")
    logging.info(f"Загрузка из {filename}")

else:
    raise UnknownCommandError(command)

except KeyboardInterrupt:
    print("\nПрервано.")
    break

except UnknownCommandError as e:
    print(f"Ошибка: {e}")
    logging.error(f"Неизвестная команда: {e}")

except (InvalidPriceError, StoreNotFoundError, DataFormatError) as e:
    print(f"Ошибка: {e}")
    logging.error(f"Ошибка: {e}")

except Exception as e:
    print(f"Ошибка: {e}")
    logging.error(f"Ошибка: {e}")

if __name__ == '__main__':
    main()

```

```

@bickrosss →/workspaces/TM_oop_lab6 (main) $ python tasks/task2/main.py
Каталог товаров
=====

Доступные команды:
add <товар> <магазин> <цена> - Добавить товар
list                          - Показать все товары
stores                        - Показать все магазины
select <магазин>              - Показать товары в магазине
save <файл.xml>               - Сохранить в XML
load <файл.xml>               - Загрузить из XML
help                          - Показать справку
exit                          - Выйти

>>> exit
Выход.

```

Рисунок 2. Результат

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x.