

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины
«Объектно-ориентированное программирование»
Вариант 1

Выполнил:
Бабенко Артём Тимофеевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Ассистент департамента цифровых,
робототехнических систем и
электроники Богданов С.С

(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 добавьте возможность работы с исключениями и логгирование.

Результат работы программы:

```
>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
load <имя_файла> - загрузить данные из файла;
save <имя_файла> - сохранить данные в файл;
help - отобразить справку;
exit - завершить работу с программой.
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
>>> add
Фамилия и инициалы? Иванов И.И.
Должность? Программист
Год поступления? 1998
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Иванов И.И.          |      Программист      |      1998      |
+-----+-----+-----+-----+
>>> save
save -> Unknown command
>>> save log_1.txt
>>> exit
```

Рисунок 1 – Результат работы программы

Код программы:

```
#!/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,
            '-' * 20,
            '-' * 8
        )
        table.append(line)

```

```

table.append(
    '{:^4} | {:^30} | {:^20} | {:^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} | {:<20} | {:>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 - worker.year >= period:
            result.append(worker)
    return result

def load(self, filename):
    with open(filename, 'r', encoding='utf8') as fin:
        xml = fin.read()

    parser = ET.XMLParser(encoding="utf8")
    tree = ET.fromstring(xml, parser=parser)
    self.workers = []

    for worker_element in tree:
        name, post, year = None, None, None

        for element in worker_element:
            if element.tag == 'name':
                name = element.text

            elif element.tag == 'post':
                post = element.text

```

```

        elif element.tag == 'year':
            year = int(element.text)

        if name is not None and post is not None \
            and year is not None:
            self.workers.append(
                Worker(
                    name=name,
                    post=post,
                    year=year
                )
            )

def save(self, filename):
    root = ET.Element('workers')

    for worker in self.workers:
        worker_element = ET.Element('worker')
        name_element = ET.SubElement(worker_element, 'name')
        name_element.text = worker.name
        post_element = ET.SubElement(worker_element, 'post')
        post_element.text = worker.post
        year_element = ET.SubElement(worker_element, 'year')
        year_element.text = str(worker.year)
        root.append(worker_element)

    tree = ET.ElementTree(root)
    with open(filename, 'wb') as fout:
        tree.write(fout, encoding='utf8', xml_declaration=True)

if __name__ == '__main__':
    # Выполнить настройку логгера.
    logging.basicConfig(
        filename='workers.log',
        level=logging.INFO
    )
    # Список работников.
    staff = Staff()
    # Организовать бесконечный цикл запроса команд.
    while True:
        try:
            # Запросить команду из терминала.
            command = input(">>> ").lower()
            # Выполнить действие в соответствие с командой.
            if command == 'exit':
                break

            elif command == 'add':
                # Запросить данные о работнике.
                name = input("Фамилия и инициалы? ")
                post = input("Должность? ")

```

```

year = int(input("Год поступления? "))
# Добавить работника.
staff.add(name, post, year)
logging.info(
    f'Добавлен сотрудник: {name}, {post}, '
    f'поступивший в {year} году.'
)

elif command == 'list':
    # Вывести список.
    print(staff)
    logging.info("Отображен список сотрудников.")

elif command.startswith('select '):
    # Разбить команду на части для выделения номера года.
    parts = command.split(maxsplit=1)
    # Запросить работников.
    selected = staff.select(parts[1])
    # Вывести результаты запроса.
    if selected:
        for idx, worker in enumerate(selected, 1):
            print(
                '{:>4}: {}'.format(idx, worker.name)
            )
        logging.info(
            f'Найдено {len(selected)} работников со '
            f'стажем более {parts[1]} лет."
        )
    else:
        print("Работники с заданным стажем не найдены.")
        logging.warning(
            f'Работники со стажем более {parts[1]} лет не найдены."
        )

elif command.startswith('load '):
    # Разбить команду на части для имени файла.
    parts = command.split(maxsplit=1)
    # Загрузить данные из файла.
    staff.load(parts[1])
    logging.info(f"Загружены данные из файла {parts[1]}")

elif command.startswith('save '):
    # Разбить команду на части для имени файла.
    parts = command.split(maxsplit=1)
    # Сохранить данные в файл.
    staff.save(parts[1])
    logging.info(f"Сохранены данные в файл {parts[1]}")

elif command == 'help':
    # Вывести справку о работе с программой.
    print("Список команд:\n")

```

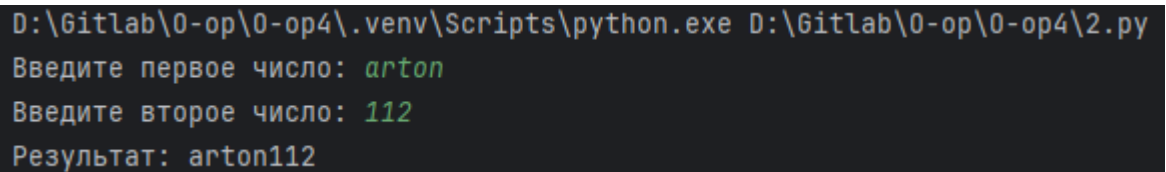
```

print("add - добавить работника;")
print("list - вывести список работников;")
print("select <стаж> - запросить работников со стажем;")
print("load <имя_файла> - загрузить данные из файла;")
print("save <имя_файла> - сохранить данные в файл;")
print("help - отобразить справку;")
print("exit - завершить работу с программой.")
else:
    raise UnknownCommandError(command)
except Exception as exc:
    logging.error(f"Ошибка: {exc}")
    print(exc, file=sys.stderr)

```

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.

Результат работы программы:



```

D:\Gitlab\0-op\0-op4\.venv\Scripts\python.exe D:\Gitlab\0-op\0-op4\2.py
Введите первое число: arton
Введите второе число: 112
Результат: arton112

```

Рисунок 1 – Результат работы программы

Код программы:

```

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

if __name__ == '__main__':
    n1 = input("Введите первое число: ")
    n2 = input("Введите второе число: ")
    try:
        print(f"Результат: {float(n1)+float(n2)}")
    except:
        print(f"Результат: {str(n1)+str(n2)}")

```

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.

Результат работы программы:

```
Введите количество строк: 5
Введите количество столбцов: 3
Введите нижнюю границу диапазона: 5
Введите верхнюю границу диапазона: 12
11 9 6
10 11 8
7 12 11
7 8 8
11 11 10
```

Рисунок 1 – Результат работы программы

Код программы:

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

from random import randint

if __name__ == '__main__':
    try:
        n = int(input("Введите количество строк: "))
        m = int(input("Введите количество столбцов: "))

        d1 = int(input("Введите нижнюю границу диапазона: "))
        d2 = int(input("Введите верхнюю границу диапазона: "))

        matrix = []
        for i in range(0,n):
            matrix.append([])
            for j in range(0,m):
                matrix[i].append(randint(d1,d2))
        for i in matrix:
            print(*i)
    except:
        print("Введены неправильные значения")
```

Индивидуальное задание. Выполнить индивидуальное задание 1 лабораторной работы 2.19, добавив возможность работы с исключениями и логгирование. Изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды.

Результат работы программы:


```
Ввод данных с клавиатуры:
Введите дробное число (first): 2.5
Введите целое число (second): 8
Pair(first=2.5, second=8)
Результат возведения в степень: 1525.87890625

Перегрузка операторов:
pair1 ** pair2: 1.3572088082974532
pair1 + pair2: 15.736111111111111
pair1 - pair2: 15.513888888888889
pair1 * pair2: 1.7361111111111111
pair1 / pair2: 140.625
pair1 == pair2: False
pair1 > pair2: True

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы



The image shows a Windows file explorer window titled "program_log.log - Блокнот". The window displays a list of files and folders, including "Блокнот" and "program_log.log". The file "program_log.log" is selected, and its contents are displayed in the right pane. The log file contains a series of entries, each starting with a timestamp and a log level, followed by a message. The messages are related to the execution of a program, including input from the user, the creation of a Pair object, and the results of various arithmetic operations performed on the Pair object. The log entries are as follows:

```
2025-02-19 01:36:24.352 - INFO - Вызван метод display с аргументами () и ключевыми аргументами {}. Результат: None
2025-02-19 01:36:24.352 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:24.352 - INFO - Вызван метод make_pair с аргументами (-2,) и ключевыми аргументами {}. Результат: <__main__.Pair object at 0x000001D3875CD810>
2025-02-19 01:36:24.352 - INFO - Вызван метод display с аргументами () и ключевыми аргументами {}. Результат: None
2025-02-19 01:36:24.352 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.667 - INFO - Вызван метод read с аргументами () и ключевыми аргументами {}. Результат: None
2025-02-19 01:36:39.667 - INFO - Вызван метод display с аргументами () и ключевыми аргументами {}. Результат: None
2025-02-19 01:36:39.667 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 1525.87890625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __pow__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: 1.3572088082974532
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __add__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: 15.736111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __sub__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: 15.513888888888889
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __mul__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: 1.7361111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __truediv__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: 140.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.668 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.668 - INFO - Вызван метод __eq__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: False
2025-02-19 01:36:39.669 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 15.625
2025-02-19 01:36:39.669 - INFO - Вызван метод power с аргументами () и ключевыми аргументами {}. Результат: 0.1111111111111111
2025-02-19 01:36:39.669 - INFO - Вызван метод __lt__ с аргументами (<__main__.Pair object at 0x000001D3875CD810>) и ключевыми аргументами {}. Результат: False
```

Рисунок 3 – Файл с логами

Код программы:

```
import logging
```

```
from datetime import datetime
```

```
# Настройка логгирования
```

```
logging.basicConfig(
```

```
    filename='program_log.log',
```

```
    level=logging.INFO,
```

```
    format='%(asctime)s.%(msecs)03d - %(levelname)s - %(message)s',
```

```
datefmt='%Y-%m-%d %H:%M:%S'
)
```

```
def log_method(func):
```

```
    """
```

```
    Декоратор для логгирования вызова метода.
```

```
    """
```

```
    def wrapper(*args, **kwargs):
```

```
        try:
```

```
            result = func(*args, **kwargs)
```

```
            logging.info(f'Вызван метод {func.__name__} с аргументами {args[1:]} и  
ключевыми аргументами {kwargs}. Результат: {result}')
```

```
            return result
```

```
        except Exception as e:
```

```
            logging.error(f'Ошибка при вызове метода {func.__name__}: {str(e)}')
```

```
            raise
```

```
    return wrapper
```

```
class Pair:
```

```
    def __init__(self, first: float, second: int):
```

```
        """
```

```
        Инициализация объекта Pair.
```

```
        :param first: дробное число (float).
```

```
        :param second: целое число (int), показатель степени.
```

```
        """
```

```
        if not isinstance(first, (float, int)):
```

```
            raise ValueError("Поле 'first' должно быть дробным числом.")
```

```
        if not isinstance(second, int):
```

```
            raise ValueError("Поле 'second' должно быть целым числом.")
```

```
        self.first = float(first)
```

```
        self.second = second
```

```
    @log_method
```

```
    def power(self):
```

```
        """
```

Возведение числа first в степень second.

:return: результат возведения в степень.

"""

return self.first ** self.second

@log_method

def read(self):

"""

Ввод значений полей с клавиатуры.

"""

try:

self.first = float(input("Введите дробное число (first): "))

self.second = int(input("Введите целое число (second): "))

except ValueError as e:

logging.error(f"Ошибка ввода данных: {str(e)}")

print("Ошибка ввода. Проверьте, что введены корректные значения.")

raise

@log_method

def display(self):

"""

Вывод значений полей на экран.

"""

print(f"Pair(first={self.first}, second={self.second})")

Перегрузка оператора возведения в степень (**)

@log_method

def __pow__(self, other):

if isinstance(other, Pair):

return self.power() ** other.power()

elif isinstance(other, (int, float)):

return self.power() ** other

else:

raise TypeError("Операнд должен быть числом или объектом класса Pair.")

```
# Перегрузка операторов сравнения
```

```
@log_method
```

```
def __eq__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() == other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() == other
```

```
    else:
```

```
        raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def __ne__(self, other):
```

```
    return not self.__eq__(other)
```

```
@log_method
```

```
def __lt__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() < other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() < other
```

```
    else:
```

```
        raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def __le__(self, other):
```

```
    return self.__lt__(other) or self.__eq__(other)
```

```
@log_method
```

```
def __gt__(self, other):
```

```
    return not self.__le__(other)
```

```
@log_method
```

```
def __ge__(self, other):
```

```
    return not self.__lt__(other)
```

```
# Перегрузка арифметических операторов
```

```
@log_method
```

```
def __add__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() + other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() + other
```

```
    else:
```

```
        raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def __sub__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() - other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() - other
```

```
    else:
```

```
        raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def __mul__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() * other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() * other
```

```
    else:
```

```
        raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def __truediv__(self, other):
```

```
    if isinstance(other, Pair):
```

```
        return self.power() / other.power()
```

```
    elif isinstance(other, (int, float)):
```

```
        return self.power() / other
```

```
    else:
```

```
raise TypeError("Операнд должен быть числом или объектом класса Pair.")
```

```
@log_method
```

```
def make_pair(first: float, second: int) -> Pair:
```

```
    """
```

```
    Создание объекта Pair с проверкой параметров.
```

```
    :param first: дробное число.
```

```
    :param second: целое число.
```

```
    :return: объект Pair.
```

```
    """
```

```
    try:
```

```
        return Pair(first, second)
```

```
    except ValueError as e:
```

```
        logging.error(f"Ошибка создания Pair: {e}")
```

```
        print(f"Ошибка создания Pair: {e}")
```

```
        exit(1)
```

```
if __name__ == '__main__':
```

```
    try:
```

```
        # Демонстрация работы класса Pair
```

```
        print("Создание объекта Pair через конструктор:")
```

```
        pair1 = Pair(2.5, 3)
```

```
        pair1.display()
```

```
        print(f"Результат возведения в степень: {pair1.power()}")
```

```
        print("\nСоздание объекта Pair через функцию make_pair():")
```

```
        pair2 = make_pair(3.0, -2)
```

```
        pair2.display()
```

```
        print(f"Результат возведения в степень: {pair2.power()}")
```

```
        print("\nВвод данных с клавиатуры:")
```

```
        pair3 = Pair(0, 0) # Создаем пустой объект
```

```
        try:
```

```
            pair3.read()
```

```
            pair3.display()
```

```

    print(f"Результат возведения в степень: {pair3.power()}")
except ValueError:
    print("Ошибка при вводе данных.")

print("\nПерегрузка операторов:")
print(f"pair1 ** pair2: {pair1 ** pair2}") # Возведение в степень
print(f"pair1 + pair2: {pair1 + pair2}") # Сложение
print(f"pair1 - pair2: {pair1 - pair2}") # Вычитание
print(f"pair1 * pair2: {pair1 * pair2}") # Умножение
print(f"pair1 / pair2: {pair1 / pair2}") # Деление
print(f"pair1 == pair2: {pair1 == pair2}") # Сравнение
print(f"pair1 > pair2: {pair1 > pair2}") # Сравнение

except Exception as e:
    logging.error(f"Необработанная ошибка: {str(e)}")
    print(f"Произошла ошибка: {str(e)}")

```

Ответы на контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

Существуют следующие виды ошибок: синтаксические ошибки в Python, исключения в Python.

2. Как осуществляется обработка исключений в языке программирования Python?

В Python обработка исключений осуществляется с помощью конструкции try и except. Код, который может вызвать исключение, помещается в блок try, а обработка исключения производится в блоке except.

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

Блок finally выполняется всегда. Блок else выполняется, если в блоке try не произошло никаких исключений.

4. Как осуществляется генерация исключений в языке Python?

Исключения в Python могут быть сгенерированы с помощью оператора raise.

5. Как создаются классы пользовательских исключений в языке Python?

Создание пользовательских исключений в Python осуществляется путем создания нового класса, который наследует от встроенного класса `Exception`.

6. Каково назначение модуля `logging`?

Модуль `logging` в Python предназначен для ведения журналов (логов) событий, возникающих во время выполнения программы.

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

Пользователю доступны разные типы сообщений, такие как предупреждение, информация или ошибка (`debug`, `info`, `warning`, `error`, `critical`). Сообщения типа `debug` могут использоваться при отладке работы программы, сообщения типа `info` могут быть полезны при взаимодействии с пользователем, `warning` используется как предупреждение, сообщения типа `error` и `critical` говорят о наличии ошибки.

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