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 13

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

_____
(подпись)

Проверил:
Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Работа с исключениями в языке Python

Цель: приобретение навыков по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее. Ссылка на гит: https://github.com/AkselSukub/OOP_4

2. Выполнил индивидуальное задание №1:

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

Код индивидуального задания №1:

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

def main() -> None:
    result: int | str
    try:
        a = input("Введите первое число: ")
        b = input("Введите второе число: ")
        c = int(a)
        d = int(b)
        result = c + d
    except ValueError:
        result = f"{a}{b}"
    except Exception as e:
        result = "Непредвиденная ошибка: " + str(e)
    finally:
        print(f"Результат: {result}")

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

4. Выполнил индивидуальное задание №2:

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

Код индивидуального задания №2:

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


import random
from typing import Generator
```

```
class Matrix:
    def __init__(self, rows: int, columns: int, start: int, end: int) -> None:
        self.rows = rows
        self.columns = columns
        self.start = start
        self.end = end
        self.matrix: list[list[int]] = []

    def generate_matrix(self) -> None:
        for name, value in self.items():
            if value <= 0:
                raise NumberNotPositiveError(name, value)

        if self.start > self.end:
            raise StartGreaterThanEndError(self.start, self.end)

        self.matrix = [
            [random.randint(self.start, self.end) for _ in range(self.columns)]
            for _ in range(self.rows)
        ]

    def items(self) -> Generator[tuple[str, int], None, None]:
        for name in ["rows", "columns"]:
            yield name, getattr(self, name)

    def __str__(self) -> str:
        if not self.matrix:
            return "Матрица пока не сгенерирована"
        string = ""
        for row in self.matrix:
            string += "|\t" + "\t".join(map(str, row)) + "\t|\n"
        return string

class StartGreaterThanEndError(Exception):
    def __init__(
        self,
        start: int,
        end: int,
        message: str = "Начало диапазона больше конца",
    ) -> None:
        self.start = start
        self.end = end
```

```
        self.message = message
        super(StartGreaterThanEndError, self).__init__(message)

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

class NumberNotPositiveError(Exception):
    def __init__(
        self,
        name: str,
        number: int,
        message: str = "Значение не является положительным",
    ):
        self.name = name
        self.number = number
        self.message = message
        super(NumberNotPositiveError, self).__init__(message)

    def __str__(self) -> str:
        return f"{self.message}: {self.name} = {self.number} (ожидалось > 0)"

def main() -> None:
    try:
        matrix = Matrix(
            int(input("Введите количество строк: ")),
            int(input("Введите количество столбцов: ")),
            int(input("Введите начало диапазона: ")),
            int(input("Введите конец диапазона: ")),
        )
        matrix.generate_matrix()
        print(matrix)

    except Exception as e:
        print("Ошибка: ", e)

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

Ответы на контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

В Python существуют два основных вида ошибок: Синтаксические ошибки: возникающие при нарушении синтаксиса языка и определяющиеся на этапе парсинга программы. Исключения: ошибки, возникающие во время выполнения программы при корректном синтаксисе кода, такие как деление на ноль, отсутствие файла и т.п.

2. Как осуществляется обработка исключений в языке программирования Python?

Обработка исключений в Python выполняется с помощью блока try…except. Код, который может вызвать исключение, помещается в блок try, а возможные исключения обрабатываются в блоке except. Это позволяет программе продолжать выполнение после обработки ошибки.

Если в программе происходит ошибка, она завершится аварийно. Чтобы этого избежать, мы используем обработку исключений через блоки try, except, else, finally.

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

finally: выполняется в любом случае, возникло исключение или нет. Он полезен для закрытия ресурсов (файлы, соединения) независимо от результата выполнения кода.

else: используется для выполнения кода, если в блоке try не возникло исключений. Это позволяет отделить основной рабочий код от кода, который должен выполняться только при успешном выполнении блока try.

4. Как осуществляется генерация исключений в языке Python?

Исключения в Python можно создавать вручную с помощью оператора raise, который генерирует исключение определенного типа и позволяет передать сообщение об ошибке.

5. Как создаются классы пользовательский исключений в языке Python?

Для создания пользовательских исключений создается класс, наследующий от базового класса исключений, например, Exception. Это позволяет задать специфичное поведение или сообщение для исключений в рамках программы.

6. Каково назначение модуля logging?

Модуль logging предназначен для регистрации (логгирования) событий, возникающих во время выполнения программы. Он позволяет записывать сообщения об ошибках, предупреждениях, информацию о работе и т.п. в файл или консоль.

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

Модуль logging поддерживает несколько уровней:

DEBUG: для отладки, используется для детализированной информации о ходе выполнения.

INFO: для записи общей информации, например, о начале и завершении работы.

WARNING: для потенциально опасных ситуаций, например, о приближении к лимитам использования ресурсов.

ERROR: для ошибок, которые не приводят к завершению работы программы, например, невозможность открыть файл.

CRITICAL: для ошибок, которые могут привести к аварийному завершению программы, например, потеря доступа к базе данных.

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