**Отчет по самостоятельной работе №6**

**по дисциплине МДК 01.02 “Инструментальные средства разработки программного обеспечения”.**

Выполнил: студент

группы 319

Меркушин Евгений Романович

Дата 17.11.2024

**Цель работы**

Цель данной работы – изучить методы и подходы к обработке исключений, научиться применять конструкции try-catch, обрабатывать и регистрировать исключения для повышения стабильности программного обеспечения.

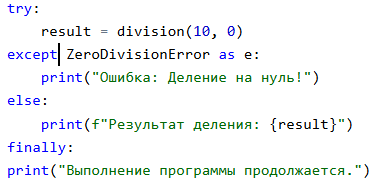
**Основная структура задания**

# Задание 1: Базовая обработка исключений

1. **Создать модуль с ошибкой деления на ноль**:



**Реализация обработки исключения с помощью try-catch**:

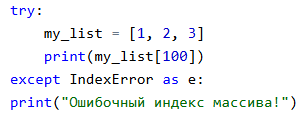


**Вывод сообщения об ошибке и продолжение выполнения программы**:

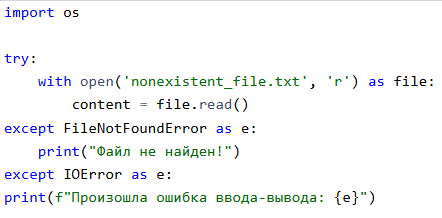
* + Пример выше демонстрирует, как можно вывести сообщение об ошибке, используя конструкцию except.
  + Блок finally гарантирует, что программа продолжит свое выполнение даже после возникновения ошибки.

# Задание 2: Иерархия исключений

1. **Изучаем иерархию стандартных исключений**: В Python, например, существует множество встроенных классов исключений, таких как ZeroDivisionError, IndexError, IOError и многие другие. Они все наследуются от базового класса Exception.
2. **Примеры обработки разных исключений**: Рассмотрим несколько примеров:
   * **Деление на ноль**: Мы уже рассмотрели этот случай в первом задании.
   * **Выход за пределы массива**:



**Ошибка ввода-вывода**:



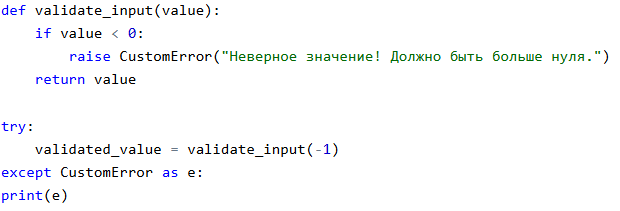
1. **Документирование различий и стратегия обработки**: Каждая категория исключений имеет свою природу и причину возникновения. Важно понимать эти различия и разрабатывать стратегии обработки, исходя из конкретных требований проекта. Например:
   * Для делений на ноль стоит предусмотреть проверку делителя на равенство нулю.
   * Ошибки индексации массива лучше обрабатывать путем проверки границ индексов.
   * Исключения ввода-вывода могут требовать перезапуска операции или уведомления пользователя.

# Задание 3: Генерация пользовательских исключений

1. **Создание пользовательского класса исключений**:



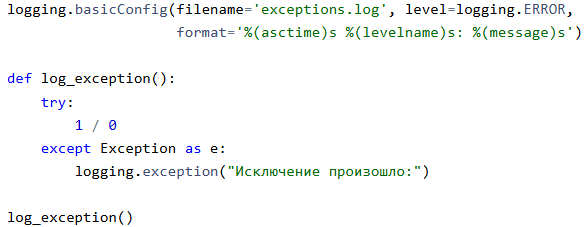
**Пример генерации и обработки исключения**:



**Обоснование необходимости пользовательского исключения**: Пользовательские исключения полезны тогда, когда стандартные классы исключений не покрывают специфические ситуации, характерные именно для вашего проекта. Это улучшает читаемость кода и облегчает диагностику ошибок.

# Задание 4: Логирование исключений

**Механизм логирования исключений в файл**: Используем стандартный модуль logging:

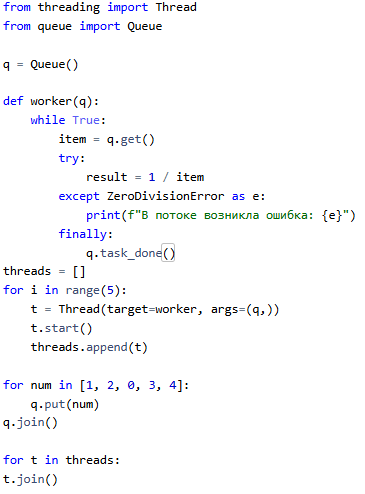


**Корректность логирования**: Необходимо убедиться, что каждая ошибка записывается с указанием времени, типа ошибки и контекста. Пример выше показывает, как это можно реализовать.

**Документация подходов к анализу и устранению исключений на основе логов**: Логи позволяют удобно анализировать ошибки постфактум. Важно документировать, какие действия следует предпринять при обнаружении той или иной ошибки. Например, для делений на ноль нужно искать места, где возможна эта ситуация, и добавлять соответствующую проверку.

# Задание 5: Обработка исключений в многопоточности

1. **Реализация обработки исключений в многопоточном окружении**: В Python можно использовать модуль threading для создания потоков и обработки исключений внутри них.



**Убеждаемся, что исключения одного потока не завершают программу**: Пример выше демонстрирует, как можно обработать исключение внутри потока, не прерывая работу остальных потоков.

1. **Синхронизация для корректной работы с исключениями**: Важно использовать механизмы синхронизации, такие как очереди (Queue), чтобы потоки могли безопасно обмениваться данными и сообщать друг другу о возникших исключениях.

**Вывод**

Из выполнения этих заданий можно сделать следующие выводы:

1. **Обработка исключений необходима для устойчивого функционирования программ**. Без правильного управления исключениями программа может аварийно завершаться при возникновении ошибок, что негативно сказывается на пользовательском опыте и надежности приложения.
2. **Иерархия исключений помогает четко классифицировать ошибки**. Понимание различных категорий исключений и их особенностей позволяет выбирать наиболее подходящую стратегию обработки для каждой конкретной ситуации.
3. **Пользовательские исключения полезны для специфичных проектов**. Создание собственных классов исключений делает код более понятным и удобным для диагностики, позволяя учитывать уникальные требования конкретного проекта.
4. **Логирование исключений играет важную роль в анализе и устранении ошибок**. Сохранение информации об ошибках в файлах или базе данных дает возможность анализировать сбои постфактум и принимать меры для их устранения.
5. **Многопоточность добавляет сложность в управление исключениями**. В многопоточных приложениях важно следить за тем, чтобы исключения в одном потоке не приводили к завершению всей программы, а также обеспечивать правильную синхронизацию для корректной обработки ошибок.