**Отчет по практической работе №**6

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

Выполнил: студент

группы 319

Муртазин Руслан Шамилевич

Дата 22.10.2024

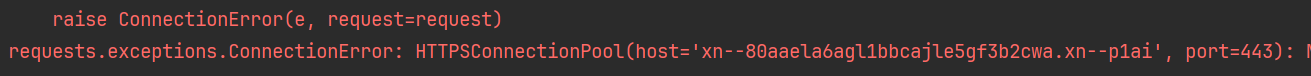
**Цель работы:** изучить методы и подходы к обработке исключений, научиться применять конструкции try-except, обрабатывать и регистрировать исключения для повышения стабильности программного обеспечения.

**Основная структура задания**

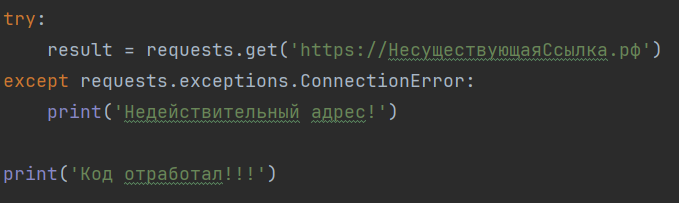
**Базовая обработка исключений**

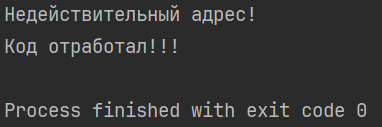
Для базовой обработки исключений создадим простой программный модуль:





При отправке запроса получаем ошибку, чтобы предотвратить аварийное завершение программы выведем сообщение об ошибке и продолжим выполнение программы:



****

**Иерархия исключений**

В Python есть иерархия исключений, которая начинается с базового класса BaseException. Давайте рассмотрим стандартные исключения и их наследников:

1. BaseException

BaseException является базовым классом для всех встроенных исключений. Обычно с ним напрямую не работают, так как для обработки исключений предназначен его подкласс Exception.

2. Exception

Exception — базовый класс для всех стандартных ошибок, возникающих в ходе выполнения программы. Большинство исключений, с которыми вы сталкиваетесь, наследуются от этого класса. Обрабатывая Exception, можно отловить практически все ошибки, кроме системных.

3. Стандартные исключения и их подтипы

* ArithmeticError — базовый класс для арифметических ошибок. Подклассы:
  + ZeroDivisionError: возникает, когда происходит деление на ноль.
  + OverflowError: выбрасывается при переполнении чисел.
  + FloatingPointError: ошибка при операциях с плавающей точкой.
* LookupError — базовый класс для ошибок, связанных с доступом к данным. Подклассы:
  + IndexError: возникает при обращении к несуществующему индексу в последовательности.
  + KeyError: выбрасывается при доступе к несуществующему ключу в словаре.
* ValueError — выбрасывается, когда объект имеет неверное значение (например, когда функция получает аргумент правильного типа, но неподходящего значения).
* TypeError — возникает при попытке выполнить операцию с несовместимыми типами данных (например, сложение числа и строки).
* ImportError — ошибка, связанная с неудачным импортом модуля или атрибута. У ImportError есть подкласс ModuleNotFoundError, который возникает, если не найден модуль для импорта.

Несколько примеров обработки различных исключений:

1. Обработка деления на ноль (ZeroDivisionError)

try:  
 a = 10  
 b = 0  
 result = a / b  
except ZeroDivisionError:  
 print("Ошибка: деление на ноль невозможно!")  
else:  
 print("Результат деления:", result)

ZeroDivisionError возникает при попытке деления на ноль, что нарушает выполнение программы. Чтобы предотвратить ошибку, следует проверять делитель перед операцией или устанавливать условие для значений, которые могут быть равны нулю, чтобы вернуть, например, None или предложить пользователю изменить значение.

1. Обработка выхода за границы массива (IndexError)

try:  
 my\_list = [1, 2, 3]  
 print(my\_list[5])  
except IndexError:  
 print("Ошибка: индекс выходит за границы списка!")

IndexError появляется, когда код обращается к индексу за пределами диапазона последовательности (например, списка). Чтобы избежать этой ошибки, рекомендуется проверять длину последовательности перед обращением к элементу или обрабатывать ее с помощью конструкции try-except, предлагая безопасное значение или исправление индекса.

1. Обработка неправильного типа данных (TypeError)
2. try:  
    number = 10  
    text = "строка"  
    result = number + text  
   except TypeError:  
    print("Ошибка: несоответствие типов данных!")

TypeError возникает при применении операции к объекту неподходящего типа, например, при сложении числа и строки. Для избежания ошибки рекомендуется проверять типы данных перед операцией или сообщать пользователю об ожидаемых типах, улучшая устойчивость программы и информативность вывода.

**Генерация пользовательских исключений**

# определение пользовательских исключений  
class ValueTooSmallError(Exception):  
 *"""Вызывается, когда входное значение мало"""* pass  
  
class ValueTooLargeError(Exception):  
 *"""Вызывается, когда входное значение велико"""* pass  
  
# оценка, которую нужно угадать  
number = 5  
  
# игра продолжается до тех пор,  
# пока пользователь его не угадает  
print('Угадай оценку Руслана за эту практическую!')  
while True:  
 try:  
 i\_num = int(input("Ввести число: "))  
 if i\_num < number:  
 raise ValueTooSmallError  
 elif i\_num > number:  
 raise ValueTooLargeError  
 break  
 except ValueTooSmallError:  
 print("Это число меньше загаданного, попробуйте еще раз!\n")  
 except ValueTooLargeError:  
 print("Это число больше загаданного, попробуйте еще раз!\n")  
  
print("Поздравляю! Вы правильно угадали.")

Этот код реализует игру, в которой пользователь пытается угадать заданное число (мою оценку за практическую работу) путем ввода чисел. Если введенное число меньше загаданного, выбрасывается пользовательское исключение ValueTooSmallError, а если больше — ValueTooLargeError, и пользователю предлагается попробовать снова. Когда число угадано правильно, программа поздравляет пользователя с победой.

Пользовательские исключения, такие как ValueTooSmallError и ValueTooLargeError, позволяют чётко указывать причину ошибки, делая логику программы понятнее и упрощая её поддержку.

**Логирование исключений**

Вот как можно реализовать механизм логирования исключений в файл с указанием времени, типа ошибки и контекста:

from datetime import datetime  
  
# определение пользовательских исключений  
class ValueTooSmallError(Exception):  
 *"""Вызывается, когда входное значение мало"""* pass  
  
class ValueTooLargeError(Exception):  
 *"""Вызывается, когда входное значение велико"""* pass  
  
# оценка, которую нужно угадать  
number = 5  
  
  
# Функция для записи логов  
def log\_error(error\_type, context):  
 with open('logs.txt', 'a', encoding='utf-8') as f:  
 log\_entry = f'{datetime.now().strftime("%Y-%m-%d %H-%M-%S")}: {error\_type} - {context}\n'  
 f.write(log\_entry)  
  
  
# игра продолжается до тех пор,  
# пока пользователь его не угадает  
print('Угадай оценку Руслана за эту практическую!')  
while True:  
 try:  
 i\_num = int(input("Введите число: "))  
 if i\_num < number:  
 log\_error("ValueTooSmallError", f"Введено: {i\_num}")  
 raise ValueTooSmallError  
 elif i\_num > number:  
 log\_error("ValueTooLargeError", f"Введено: {i\_num}")  
 raise ValueTooLargeError  
 break  
 except ValueTooSmallError:  
 print("Это число меньше загаданного, попробуйте еще раз!\n")  
 except ValueTooLargeError:  
 print("Это число больше загаданного, попробуйте еще раз!\n")  
  
print("Поздравляю! Вы правильно угадали.")

Для анализа и устранения исключений на основе логов необходимо регулярно проверять файл логов, чтобы выявлять повторяющиеся ошибки и их коренные причины. Выявление паттернов ошибок поможет адаптировать логику обработки, добавляя подсказки и улучшая пользовательский интерфейс для снижения вероятности ввода некорректных данных. Также важно документировать типичные ошибки и методы их предотвращения, а также собирать обратную связь от пользователей для дальнейшего улучшения программы.

**Обработка исключений в многопоточности**

Для обработки исключений в многопоточной среде важно правильно организовать код, чтобы ошибки в одном потоке не влияли на другие потоки или на основную программу. Вот шаги по реализации обработки исключений в многопоточном приложении с использованием модуля threading в Python:

import threading

import random

# Функция для выполнения потока

def worker(thread\_id):

try:

# Случайная ошибка для демонстрации

if random.choice([True, False]):

raise ValueError(f"Ошибка в потоке {thread\_id}")

print(f"Поток {thread\_id} выполняется успешно.")

except Exception as e:

print(f"Исключение в потоке {thread\_id}: {e}")

# Создание и запуск потоков

threads = []

for i in range(3): # Запускаем 3 потока

thread = threading.Thread(target=worker, args=(i,))

threads.append(thread)

thread.start()

# Ожидание завершения всех потоков

for thread in threads:

thread.join()  
print("Все потоки завершены.")

**Вывод работы**

В ходе практической работы я изучил механизмы обработки исключений в Python, включая создание пользовательских исключений и логирование ошибок. Я реализовал примеры, в которых обрабатывались различные исключения, такие как деление на ноль и выход за пределы массива. Логирование ошибок в файл дало возможность сохранить информацию о возникших проблемах с указанием времени и контекста, что значительно упростило анализ и устранение ошибок. Кроме того, я освоил обработку исключений в многопоточной среде, что является критически важным для обеспечения стабильности приложения. Я научился использовать блоки try-except для обработки ошибок в каждом потоке, а также применять синхронизацию для предотвращения конфликтов. Все эти навыки помогут мне разрабатывать более надежные и устойчивые программы в будущем.