

Обработка ошибок и исключений в коде Python

Яндекс Взгляд · Опрос

Выберите 1 или несколько ответов

Какие сервисы проверки истории автомобилей вы знаете?

☐

Avtocod

☐

ПроАвто/Auto.ru

☐

Автотека/Авито

☐

Avinfobot

☐

Ни один из вариантов

1 из 3 вопросов

Продолжить

[Справочник по языку Python3.](#) / Обработка ошибок и исключений в коде Python

Существует, как минимум, два различных вида ошибок:

- 1. синтаксические ошибки;
- 2. исключения.

Синтаксические ошибки.

[Синтаксические ошибки](#), также известные как ошибки синтаксического анализа, являются, самой распространенной проблемой пока изучаете Python:

```
>>> while True print('Hello world')
# File "<stdin>", line 1
#   while True print('Hello world')
#           ^
# SyntaxError: invalid syntax
```

В примере ошибка обнаружена в [функции print\(\)](#), так как перед ней отсутствует двоеточие ':'. Имя файла и номер строки напечатаны, чтобы вы знали, где искать, если входные данные поступили из сценария.

Парсер повторяет ошибочную строку и отображает небольшую "стрелку", указывающую на самую раннюю точку в строке, где была обнаружена ошибка. Ошибка вызвана или, по крайней мере, обнаружена токеном, предшествующим стрелке.

Исключения.

Даже если оператор или выражение синтаксически верны, это может вызвать ошибку при попытке выполнить его. Ошибки, обнаруженные во время выполнения, [называются исключениями](#) и не являются безусловно фатальными. Однако большинство исключений не [обрабатываются](#) программами и приводят к сообщениям об ошибках, как показано здесь:

```
>>> 10 * (1/0)
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# ZeroDivisionError: division by zero
>>> 10 * 3
```

Вверх

```
# Traceback (most recent call last):  
#   File "<stdin>", line 1, in <module>  
# NameError: name 'spam' is not defined  
  
>>> '2' + 2  
# Traceback (most recent call last):  
#   File "<stdin>", line 1, in <module>  
# TypeError: Can't convert 'int' object to str implicitly
```

Последняя строка сообщения об ошибке указывает на то, что произошло. [Исключения бывают разных типов](#), и тип печатается как часть сообщения: типы в примере - это `ZeroDivisionError`, `NameError` и `TypeError`. Строка, напечатанная в качестве типа исключения, является именем возникшей [исключительной ситуации](#). Это верно для всех встроенных исключений, но не обязательно должно быть верно для пользовательских исключений, хотя это полезное соглашение. Стандартные имена исключений - это встроенные идентификаторы, не зарезервированные ключевые слова.

Остальная часть строки содержит подробности, основанные на типе исключения и причинах его возникновения.

Предыдущая часть сообщения об ошибке показывает контекст, в котором произошло исключение, в форме обратной трассировки стека. В общем случае он содержит трассировку стека, в которой перечислены исходные строки. Строки считанные из стандартного ввода - не отображаются.

[Встроенные исключения](#) перечисляют встроенные исключения и их значения.

[Обработка/перехват исключений try/except в Python](#)

Приемы перехвата ошибок конструкцией try/except. Оператор try может содержать несколько предложений except, чтобы указать обработчики для различных исключений. В этом случае будет выполнен только один обработчик.

[Инструкция finally, очистка внешних ресурсов в Python](#)

Если инструкция finally присутствует, то она будет выполняться как последняя задача перед завершением выполнения конструкции try. Инструкция finally выполняется независимо от того, создает ли блок кода в операторе try исключени

[Тонкости работы конструкции try/except/else/finally в Python](#)

В операторе try определяется обработчики исключений и/или код очистки для группы операторов.

[Создание пользовательского класса исключения в Python](#)

В Python пользователи могут определять свои собственные исключения, создавая новый класс. Этот класс исключений должен быть прямо или косвенно производным от встроенного класса `Exception`.

[Обработка группы исключений, оператор except* в Python](#)

Новое в Python 3.11. Группы исключений используются try/except, как и со всеми другими исключениями. Кроме того, они распознаются оператором except*, которое соответствует их подгруппам на основе типов содержащихся исключений.

[Эффективная обработка исключений в Python](#)

Если не обрабатывать возможные ошибки, то в любой момент может все сломаться. Необходимо правильно и активно реагировать на любую ситуацию, чтобы минимизировать ошибки при обработке данных.

[Инструкция raise, принудительный вызов исключений в Python](#)

Инструкция raise позволяет программисту принудительно вызвать одно исключение в любое время и в любом месте кода. Повторно вызвать исключение, которое было перехвачено try/except. Создавать исключения, когда выполнение программы бессмысленно или не может продолжаться.

[Отладочные утверждение assert в Python](#)

Утверждения Assert - это удобный способ вставить отладочные утверждения в программу.

[Улучшения сообщений об ошибках в Python 3.10](#)

Улучшения коснулись сообщений об ошибке `SyntaxErrors`, `IndentationErrors`, `AttributeErrors` и `NameErrors`. В материале рассматриваются отличия сообщений об ошибках в примерах.

[Улучшения сообщений об ошибках в Python 3.11](#)

При печати трассировки интерпретатор теперь будет указывать точное выражение, вызвавшее ошибку, а не только строку.

- [ОБЗОРНАЯ СТРАНИЦА РАЗДЕЛА](#)
- [Обработка/перехват исключений try/except](#)
- [Инструкция finally, очистка внешних ресурсов](#)
- [Тонкости работы конструкции try/except/else/finally](#)
- [Создание пользовательского класса исключения](#)
- [Обработка группы исключений, оператор except*](#)
- [Эффективная обработка исключений](#)
- [Инструкция raise, принудительный вызов исключений](#)
- [Отладочные утверждение assert](#)
- [Улучшения сообщений об ошибках 3.10](#)
- [Улучшения сообщений об ошибках 3.11](#)

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

РЕКЛАМА



12+ Сроки проведения с 18.09.23 по 30.09.23. Товары и условия их распродажи определяются продавцами. Такие товары и отмечены значком «Распродажа». Представлены собирательные образы товаров.

Вверх