

Исключения

Исключения в программировании

это специальные события, которые возникают при выполнении программы и указывают на ошибки или необычные ситуации

01

ZeroDivisionError

Исключение ZeroDivisionError возникает, когда попытка деления на ноль выполняется в Python. Деление на ноль неопределено в математике и не может быть выполнено

Пример

```
...
numerator = 10
denominator = 0

result = numerator / denominator # Вызывает исключение ZeroDivisionError
print(result)
```

В этом примере переменная denominator равна нулю, и попытка выполнить деление numerator / denominator приводит к исключению ZeroDivisionError. Когда исключение возникает, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Чтобы избежать возникновения исключения ZeroDivisionError, необходимо обрабатывать случай деления на ноль программными средствами. Можно использовать условные операторы для проверки значения делителя перед выполнением деления

Пример

```
...
numerator = 10
denominator = 0

if denominator != 0:
    result = numerator / denominator
    print(result)
else:
    print("Ошибка: Деление на ноль!")
```

В этом примере перед выполнением деления проверяется, не является ли делитель равным нулю. Если данный случай обнаружен, выводится сообщение об ошибке. В противном случае, деление выполняется без ошибок

Обработка исключений `ZeroDivisionError` является важной частью программирования, чтобы предотвратить неопределенные результаты и обеспечить корректное выполнение программы

02

ValueError

Исключение `ValueError` возникает, когда функция ожидает аргумент с правильным типом данных, но получает аргумент некорректного значения или типа данных

Вот примеры ситуаций, которые могут вызвать исключение `ValueError`

- 01 Передача функции строкового значения, когда она ожидает целое число

```
...  
number = int("abc") # Вызывает исключение ValueError
```

- 02 Попытка преобразования строки в число, когда строка не может быть интерпретирована как число:

```
...  
number = int("10.5") # Вызывает исключение ValueError
```

- 03 Попытка разыменования строки, содержащей букву, в качестве списка

```
...  
my_string = "hello"  
letter = my_string[10] # Вызывает исключение ValueError
```

Во всех этих ситуациях функция ожидала получить корректные значения или тип данных, но получила что-то неподходящее, что привело к исключению `ValueError`. Когда возникает исключение `ValueError`, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Управление исключениями `ValueError` может включать в себя проверку исходных данных или использование конструкции `try-except` для обработки ошибки. Например:

```
...  
try:  
    number = int("abc")  
except ValueError:  
    print("Ошибка: Возникло исключение ValueError!")
```

В этом примере try-except блок позволяет перехватить исключение, вызванное некорректным преобразованием строки в число. Вместо завершения программы будет выведено сообщение об ошибке. Программа продолжит выполнение дальше после блока except

03

IndexError

Исключение IndexError возникает, когда код пытается обратиться к элементу последовательности (например, списку или строке) по индексу, который находится за пределами допустимого диапазона индексов

Вот примеры ситуаций, которые могут вызвать исключение IndexError

01 Попытка обратиться к элементу списка по недопустимому индексу

```
...  
my_list = [1, 2, 3]  
print(my_list[3]) # Вызывает исключение IndexError
```

02 Попытка обратиться к символу строки по недопустимому индексу

```
...  
my_string = "Hello"  
print(my_string[10]) # Вызывает исключение IndexError
```

02 Попытка обратиться к элементу списка вложенного списка по недопустимому индексу

```
...  
nested_list = [[1, 2, 3], [4, 5, 6]]  
print(nested_list[2][0]) # Вызывает исключение IndexError
```

Во всех этих ситуациях программа пытается обратиться к элементу последовательности по индексу, который выходит за пределы допустимого диапазона индексов. Когда возникает исключение IndexError, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Управление исключением IndexError может включать в себя проверку границы индекса перед обращением к элементам последовательности или использование конструкции try-except для обработки ошибки

Пример

```
...
my_list = [1, 2, 3]
try:
    print(my_list[3])
except IndexError:
    print("Ошибка: Возникло исключение IndexError!")
```

В этом примере try-except блок позволяет перехватить исключение, вызванное попыткой доступа к элементу списка по неправильному индексу. Вместо завершения программы будет выведено сообщение об ошибке. Программа продолжит выполнение дальше после блока except

04 KeyError

Исключение KeyError возникает, когда попытка обратиться к несуществующему ключу в словаре или некорректно использовать ключ в другой структуре данных, требующей ключ-значение-пары

Вот примеры ситуаций, которые могут вызвать исключение KeyError

01 Попытка обратиться к несуществующему ключу в словаре

```
...
my_dict = {"apple": "яблоко", "banana": "банан"}
print(my_dict["orange"]) # Вызывает исключение KeyError
```

02 Попытка обратиться к ключу в словаре, которого там нет в некоторых случаях

```
...
my_dict = {"apple": "яблоко", "banana": "банан"}
key = "orange"

if key in my_dict:
    print(my_dict[key])
else:
    print("Ошибка: Ключ не найден!")
```

03 Попытка использовать ключ в другой структуре данных, например, при извлечении элемента из множества по ключу:

```
...  
my_set = {"apple", "banana", "orange"}  
print(my_set["apple"]) # Вызывает исключение KeyError
```

Во всех этих ситуациях код пытается использовать ключ, который не существует в структуре данных или не является корректным для этой операции. Когда возникает исключение `KeyError`, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Управление исключением `KeyError` может включать в себя проверку наличия ключа в структуре данных перед его использованием или использование конструкции `try-except` для обработки ошибки

Пример

```
...  
my_dict = {"apple": "яблоко", "banana": "банан"}  
try:  
    print(my_dict["orange"])  
except KeyError:  
    print("Ошибка: Возникло исключение KeyError!")
```

В этом примере `try-except` блок позволяет перехватить исключение, вызванное попыткой обращения к несуществующему ключу словаря. Вместо завершения программы будет выведено сообщение об ошибке. Программа продолжит выполнение дальше после блока `except`

05

OverflowError

Исключение `OverflowError` возникает, когда попытка выполнить операцию приводит к переполнению численного типа данных

Вот примеры ситуаций, которые могут вызвать исключение `OverflowError`

01 Переполнение целочисленного типа данных

```
...  
result = 2 ** 1000 # Вызывает исключение OverflowError
```

02 Переполнение при выполнении математических операций

```
...  
import math  
result = math.exp(1000) # Вызывает исключение OverflowError
```

02 Переполнение при выполнении операций с плавающей запятой

```
...  
result = float("inf") * 1000 # Вызывает исключение OverflowError
```

Во всех этих ситуациях код пытается выполнить операцию, которая приводит к результату, выходящему за пределы допустимого диапазона для соответствующего типа данных. Когда возникает исключение `OverflowError`, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Работа с исключением `OverflowError` может включать в себя проверку результатов операций на предмет возможного переполнения или использование конструкции `try-except` для обработки ошибки

Пример

```
...  
try:  
    result = 2 ** 1000  
except OverflowError:  
    print("Ошибка: Возникло исключение OverflowError!")
```

В этом примере `try-except` блок позволяет перехватить исключение, вызванное попыткой выполнить операцию, приводящую к переполнению численного типа данных. Вместо завершения программы будет выведено сообщение об ошибке. Программа продолжит выполнение дальше после блока `except`

06

FileNotFoundException

Исключение `FileNotFoundException` возникает, когда попытка открыть или выполнить операции с файлом, который не существует или не может быть найден

Вот примеры ситуаций, которые могут вызвать исключение FileNotFoundError

01 Попытка открыть несуществующий файл для чтения

```
...  
file = open("nonexistent_file.txt", "r") # Вызывает исключение FileNotFoundError
```

02 Попытка выполнить операции с файлом, указывая неправильное имя или путь к файлу

```
...  
import os  
  
file_path = "path/to/nonexistent_file.txt"  
file_size = os.path.getsize(file_path) # Вызывает исключение FileNotFoundError
```

03 Попытка выполнить операции с файлом в недоступной директории

```
...  
import os  
file_path = "/root/secret_file.txt"  
file_size = os.path.getsize(file_path) # Вызывает исключение FileNotFoundError (если у вас нет прав доступа)
```

Во всех этих ситуациях код пытается работать с файлом, который не существует или не может быть найден по указанному пути. Когда возникает исключение FileNotFoundError, выполнение программы останавливается, и на консоль выводится сообщение об ошибке

Управление исключением FileNotFoundError может включать в себя проверку существования файла перед его открытием или использование конструкции try-except для обработки ошибки

Пример

```
...  
try:  
    file = open("nonexistent_file.txt", "r")  
except FileNotFoundError:  
    print("Ошибка: Файл не найден!")
```

В этом примере try-except блок позволяет перехватить исключение, вызванное попыткой открытия несуществующего файла. Вместо завершения программы будет выведено сообщение об ошибке. Программа продолжит выполнение дальше после блока except

Итоги

В этой теме мы разобрали понятие исключение. Изучение и обработка различных исключений, таких как `ZeroDivisionError`, `ValueError`, `IndexError`, `KeyError`, `OverflowError` и `FileNotFoundException`, позволяет более эффективно управлять ошибками, обеспечивать корректное выполнение программы и повышать ее надежность. Это важные инструменты, которые помогают программистам создавать более устойчивый и безопасный код.