eebb

ЯЗЫК PYTHON

Лекция 14. Исключения, наследование классов

Мотивация

- Не всегда все идет по плану
- Сеть пропадает
- Файлы не находятся
- Пользователь передает странные параметры

Мотивация

- Вариант решения: возвращать специальное значение при ошибке
- Например, None
- Иногда так и делают
- Но есть минусы

Мотивация

- Первый минус: усложняем обработку результата вызова
- А от места вызова до обработки бывает далеко
- Второй минус: None может быть вариантом нормального результата
- Третий минус: нет детализации проблемы
- Четвертый минус: можно не проверить результат вызова

Исключения

- В Python есть механизм исключений
- Можно создать объект определенного класса
- И использовать специальную конструкцию - raise
- "Бросить исключение"

Исключения

- Выполнение функции/метода прекращается
- И вызывающей функции - тоже
- И так далее
- Пока все функции не прекратятся

Исключения

- Какого класса должен быть объект ?
- Первый вариант - Exception
- Есть такой встроенный
- Но одного класса маловато: скоро вернемся к этому

Пример брошенного исключения

```
1 print(123)
2 raise Exception()
3 print(234)
```

Пример брошенного исключения

```
1 def f():
2     print('abc')
3     raise Exception('I am exception')
4     print('bcd')
5
6 print(123)
7 f()
8 print(234)
```

Как обрабатывать

- Но мы же не хотим просто завершать программу при любой ошибке
- И начинали именно с обработки
- Есть специальная конструкция для обработки исключений
- Называется try/except

Как обрабатывать

- Если ожидаем исключения в блоке кода
- Начинаем его с try:
- Блок сдвигаем
- В конце пишем except:
- После except: - блок обработки со смещением

Как обрабатывать

- Если брошено исключение, смотрим, где оно брошено
- Если это место - непосредственно между try и except
- Исключение считаем пойманным
- Исполняем код после except

Как обрабатывать

- Если не непосредственно внутри try/except
- Прекращаем работу функции
- Идем в точку вызова
- Место вызова считаем за источник исключения
- Повторяем схему рекурсивно

Пример пойманного исключения

```
 1 def f():
 2     print('abc')
 3     try:
 4         raise Exception('I am exception')
 5     except:
 6         print("caught exception")
 7     print('bcd')
 8
 9 print(123)
10 f()
11 print(234)
```

Пример пойманного исключения

```
 1 def f():
 2     print('abc')
 3     raise Exception('I am exception')
 4     print('bcd')
 5
 6 print(123)
 7 try:
 8     f()
 9 except:
10     print("caught exception")
11 print(234)
```

Как обрабатывать

- После except в той же строчке можем написать Exception
- Пока это ничего нового не дает
- Но это синтаксически корректно
- Чуть позже увидим пользу

Пример пойманного исключения

```
 1 def f():
 2     print('abc')
 3     raise Exception('I am exception')
 4     print('bcd')
 5
 6 print(123)
 7 try:
 8     f()
 9 except Exception:
10     print("caught exception")
11 print(234)
```

Как обрабатывать

- Исключение - это объект
- Хотелось бы в обработчике исключения иметь доступ к исключению
- Надо после Exception написать as и имя идентификатора
- В обработчике под ним будет доступен объект-исключение

Как обрабатывать

- В свойствах объекта-исключения можно хранить детали происшедшего
- Установить их перед бросанием
- Почитать в обработчике
- Например, номер элемента списка
- Если суть исключения в том, что элемент списка не удовлетворяет требованиям

Пример использования объекта

```
 1 def compare(s):
 2     parts = s.split()
 3     if len(parts) != 2:
 4         raise Exception(len(parts))
 5     return parts[0] == parts[1]
 6
 7 try:
 8     compare('hello world')
 9     compare('helloworld')
10 except Exception as exc:
11     print("caught exception", exc)
```

Пример использования объекта

```
 1 def compare(s):
 2     parts = s.split()
 3     if len(parts) != 2:
 4         e = Exception('illegal # of parts')
 5         e.n_parts = len(parts)
 6         raise e
 7     return parts[0] == parts[1]
 8
 9 try:
10     compare('hello world')
11     compare('helloworld')
12 except Exception as exc:
13     print("caught exception", exc, exc.n_parts)
```

А если два исключения

- Например, хотим посчитать среднее списка чисел
- Одна ошибка: список пуст
- Другая: один из элементов - не число
- Возможны разные атрибуты, разная обработка

А если два исключения

- Напрашивается идея: дважды писать except
- Каждый раз - со своим именем исключения
- Но нам может потребоваться другая стратегия
- Обработать все исключения единообразно

А если много исключений

- А может потребоваться единообразно обработать группу исключений
- Например, все исключения про работу с файлами
- Или все арифметические
- Поможет механизм наследования

Механизм наследования

- Наследование классов - общий механизм
- Мы его изучим в контексте исключений
- Но полезен он не только здесь
- Основная идея: класс можно создавать не с нуля
- А унаследовать от другого класса

Пример наследования

- В нашем случае есть класс Exception
- Хотим создать два класса: EmptyListException и IllegalElementException
- Было бы логично в вызове __init__ класса наследника вызвать __init__ родителя
- И иметь доступ к свойствам, определенным в __init__ родителя

Синтаксис наследования

- После имени класса в скобках пишем класс-наследник
- Вообще говоря, наследников может быть несколько
- Но мы этот вариант не рассматриваем
- Он добавляет сложности

Вызов __init__ родителя

- Класс-родитель часто называют суперклассом
- Отсюда название полезной встроенной функции - super
- Нам интересен вариант без параметров
- Ее можно вызывать только из метода класса
- Возвращает суперкласс

Вызов __init__ родителя

- Можем вызвать super() из __init__ наследника
- Дописать .__init__
- И в скобках - параметры, передаваемые в __init__ родителя
- Первый из них - self

Пример определения класс-наследника

```
 1 class EmptyListException(Exception):
 2
 3     def __init__(self):
 4         super().__init__(self, "List is empty")
 5
 6 class IllegalElementException(Exception):
 7
 8     def __init__(self, index, value):
 9         super().__init__(self, f"illegal element at index {
10         self._index = index
11         self._value = value
```

Как ловить наследников Exception

- (Тут становится понятным смысл имени исключения в except)
- Можем написать except и имя исключения
- Если надо - назначить имя переменной
- Можно написать несколько except

Обработка групп исключений

- Любой except ловит исключение того класса, который указан
- Если не указан - подразумевается Exception
- Или любого подкласса
- Возникает вопрос: а если исключение подходит под несколько ?

Выбор except

- Идем по списку except
- Смотрим класс, связанный с текущим except
- Если исключение - объект того же класса или подкласса - except подходит
- Как нашли первый подходящий - исключение считается пойманным
- Исполняется код под этим except
- Следующие except не проверяются

Выбор except

- Следующие except не проверяются
- Если не нашли подходящий except - исключение остается необработанным
- Если внутри обработчика бросается новое исключение
- Перебор except в том же try/except не продолжается

Встроенные исключения

- Python уже содержит много исключений
- Все наследуются от Exception
- Прямо или косвенно
- Одно из них - ValueError

Переброс исключения

- Реализуем функцию, которая принимает строку
- Разбивает ее по пробельным последовательностям
- Ожидает, что слова преобразуются в числа
- И посчитаем среднее
- И бросим два уже определенных исключения в соответствующих ситуациях

Переброс исключения

- Применим технику переброса исключения
- Есть встроенная функция int
- Преобразует строку в целое
- Бросает ValueError, если не может преобразовать
- А нам-то нужен IllegalElementException

Переброс исключения

- Обернем int в try/except
- Поймаем ValueError
- В обработчике используем raise IllegalElementException
- И в этом try/except он уже не будет обработан

Пример перебрасывания

```
 1 def str_mean(s):
 2     def convert(index, word):
 3         try:
 4             return int(word)
 5         except ValueError:
 6             raise IllegalElementException(index, word)
 7 
 8     if not s:
 9         raise EmptyListException()
10     words = s.split()
11     if not words:
12         raise EmptyListException()
13 
14     return sum(convert(index, w) for index, w in enumerate(
```

Пример общего обработчика

```
1 def f(s):
2     try:
3         print(str_mean(s))
4     except Exception as exc:
5         print('caught', exc)
6
7 f('')
8 f('123 45')
9 f('iq123 45')
```

Пример отдельных обработчиков

```
 1 def f(s):
 2     try:
 3         print(str_mean(s))
 4     except EmptyListException as exc:
 5         print('caught', exc)
 6     except IllegalElementException as exc:
 7         print('caught', exc, exc._index, exc._value)
 8
 9 f('')
10 f('123 45')
11 f('iq123 45')
```

Пример смешанных обработчиков

```
 1 def f(s):
 2     try:
 3         print(str_mean(s))
 4     except IllegalElementException as exc:
 5         print('caught', exc, exc._index, exc._value)
 6     except Exception as exc:
 7         print('caught', exc)
 8
 9
10 f('')
11 f('123 45')
12 f('iq123 45')
```

Пример недоступного обработчика

```
 1 def f(s):
 2     try:
 3         print(str_mean(s))
 4     except Exception as exc:
 5         print('caught', exc)
 6     except IllegalElementException as exc:
 7         print('caught', exc, exc._index, exc._value)
 8
 9
10 f('')
11 f('123 45')
12 f('iq123 45')
```

else

- После всех except может идти отдельная ветка else
- В нее попадаем, если исключений не было
- Это не совсем то же, что и поместить код в конец try/except
- Исключение, брошенное здесь, летит дальше

finally

- В конце может идти ключевое слово finally с блоком кода
- Если не было исключений, то после основного блока и else исполнится finally
- Если было исключение и его поймали, после обработика выполнится блок finally
- Есть есть непойманное исключение, то перед тем, как оно полетит дальше, исполнится finally

Пример на else

```
 1 def f(s):
 2     try:
 3         print(str_mean(s))
 4     except IllegalElementException as exc:
 5         print('caught', exc, exc._index, exc._value)
 6     except Exception as exc:
 7         print('caught', exc)
 8     else:
 9         print('ok')
10
11
12 f('')
13 f('123 45')
14 f('iq123 45')
```

Пример на finally

```
 1 def f(s):
 2     try:
 3         print(str_mean(s))
 4     except IllegalElementException as exc:
 5         print('caught', exc, exc._index, exc._value)
 6     except Exception as exc:
 7         print('caught', exc)
 8     else:
 9         print('ok')
10     finally:
11         print('done')
12
13
14 f('')
15 f('123 45')
```

Интересный случай

- Исключение может бросить любой код
- В том числе и тот, который внутри finally-блока
- А если было непойманное исключение, а в finally бросилось новое ?
- "Победит" то, которое в finally

Пример

```
try:
    try:
        int('dcqdcqc')
    finally:
        print(1/0)
except Exception as exc:
    print('got', exc)
```