http://pythonworld.ru/tipy-dannyx-v-python/isklyucheniya-v-python-konstrukciya-try-except-dlya-obrabotki-isklyuchenij.html

Исключения (exceptions) - ещё один тип данных в python. Исключения необходимы для того, чтобы сообщать программисту об ошибках.

Самый простейший пример исключения - деление на ноль:

>>> 100 / 0

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

100 / 0

ZeroDivisionError: division by zero

В данном случае интерпретатор сообщил нам об исключении ZeroDivisionError, то есть делении на ноль. Также возможны и другие исключения:

>>> 2 + '1'

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

2 + '1'

TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> int('qwerty')

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

int('qwerty')

ValueError: invalid literal for int() with base 10: 'qwerty'

В этих двух примерах генерируются исключения TypeError и ValueError соответственно. Подсказки дают нам полную информацию о том, где порождено исключение, и с чем оно связано.

Рассмотрим иерархию встроенных в python исключений, хотя иногда вам могут встретиться и другие, так как программисты могут создавать собственные исключения. Данный список актуален для [python 3.3](http://pythonworld.ru/novosti-mira-python/chto-novogo-v-python-33.html), в более ранних версиях есть незначительные изменения.

* **BaseException** - базовое исключение, от которого берут начало все остальные.
  + **SystemExit** - исключение, порождаемое функцией sys.exit при выходе из программы.
  + **KeyboardInterrupt** - порождается при прерывании программы пользователем (обычно сочетанием клавиш Ctrl+C.
  + **GeneratorExit** - порождается при вызове метода close объекта generator.
  + **Exception** - а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать.
    - **StopIteration** - порождается [встроенной функцией](http://pythonworld.ru/osnovy/vstroennye-funkcii.html) next, если в итераторе больше нет элементов.
    - **ArithmeticError** - арифметическая ошибка.
      * **FloatingPointError** - порождается при неудачном выполнении операции с плавающей запятой. На практике встречается нечасто.
      * **OverflowError** - возникает, когда результат арифметической операции слишком велик для представления. Не появляется при обычной работе с числами (так как python поддерживает длинные числа), но может возникать в некоторых других случаях.
      * **ZeroDivisionError** - деление на ноль.
    - **AssertionError** - выражение в функции assert ложно.
    - **AttributeError** - объект не имеет данного атрибута (значения или метода).
    - **BufferError** - операция, связанная с буфером, не может быть выполнена.
    - **EOFError** - функция input() наткнулась на конец файла и не смогла при этом ничего прочитать.
    - **ImportError** - не удалось импортирование модуля или его атрибута.
    - **LookupError** - некорректный индекс или ключ.
      * **IndexError** - индекс не входит в диапазон элементов.
      * **KeyError** - несуществующий ключ (в [словаре](http://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html), [множестве](http://pythonworld.ru/tipy-dannyx-v-python/mnozhestva-set-i-frozenset.html) или другом объекте).
    - **MemoryError** - недостаточно памяти.
    - **NameError** - не найдено переменной с таким именем.
      * **UnboundLocalError** - сделана ссылка на локальную переменную в функции, но переменная не определена ранее.
    - **OSError** - ошибка, связанная с системой.
      * **BlockingIOError**
      * **ChildProcessError** - неудача при операции с дочерним процессом.
      * **ConnectionError** - базовый класс для исключений, связанных с подключениями.
        + **BrokenPipeError**
        + **ConnectionAbortedError**
        + **ConnectionRefusedError**
        + **ConnectionResetError**
      * **FileExistsError** - попытка создания [файла](http://pythonworld.ru/tipy-dannyx-v-python/fajly-rabota-s-fajlami.html) или директории, которая уже существует.
      * **FileNotFoundError** - файл или директория не существует.
      * **InterruptedError** - системный вызов прерван входящим сигналом.
      * **IsADirectoryError** - ожидался файл, но это директория.
      * **NotADirectoryError** - ожидалась директория, но это файл.
      * **PermissionError** - не хватает прав доступа.
      * **ProcessLookupError** - указанного процесса не существует.
      * **TimeoutError**
    - **ReferenceError** - попытка доступа к атрибуту со слабой ссылкой.
    - **RuntimeError** - возникает, когда исключение не попадает ни под одну из других категорий.
    - **NotImplementedError** - возникает, когда абстрактные методы класса требуют переопределения в дочерних классах.
    - **SyntaxError** - синтаксическая ошибка.
      * **IndentationError** - неправильные отступы.
        + **TabError** - смешивание в отступах табуляции и пробелов.
    - **SystemError** - внутренняя ошибка.
    - **TypeError** - операция применена к объекту несоответствующего типа.
    - **ValueError** - функция получает аргумент правильного типа, но некорректного значения.
      * **UnicodeError** - ошибка, связанная с кодированием / раскодированием unicode в [строках](http://pythonworld.ru/tipy-dannyx-v-python/stroki-literaly-strok.html).
        + **UnicodeEncodeError** - исключение, связанное с кодированием unicode.
        + **UnicodeDecodeError** - исключение, связанное с декодированием unicode.
        + **UnicodeTranslateError** - исключение, связанное с переводом unicode.
    - **Warning** - предупреждение.

Теперь, зная, когда и при каких обстоятельствах могут возникнуть исключения, мы можем их обрабатывать. Для обработки исключений используется конструкция **try - except**.

Первый пример применения этой конструкции:

>>> try:

k = 1 / 0

except ZeroDivisionError:

k = 0

>>> print(k)

0

В блоке try мы выполняем инструкцию, которая может породить исключение, а в блоке except мы перехватываем их. При этом перехватываются как само исключение, так и его потомки. Например, перехватывая ArithmeticError, мы также перехватываем FloatingPointError, OverflowError и ZeroDivisionError.

>>> try:

k = 1 / 0

except ArithmeticError:

k = 100

>>> print(k)

100

Также возможна инструкция except без аргументов, которая перехватывает вообще всё (и прерывание с клавиатуры, и системный выход и т. д.). Поэтому в такой форме инструкция except практически не используется, а используется except Exception. Однако чаще всего перехватывают исключения по одному, для упрощения отладки (вдруг вы ещё другую ошибку сделаете, а except её перехватит).

Ещё две инструкции, относящиеся к нашей проблеме, это **finally** и else. Finally выполняет блок инструкций в любом случае, было ли исключение, или нет (применима, когда нужно непременно что-то сделать, к примеру, закрыть файл). Инструкция else выполняется в том случае, если исключения не было.

f = open('1.txt')

ints = []

try:

for line in f:

ints.append(int(line))

except ValueError:

print('Это не число. Выходим.')

except Exception:

print('Это что ещё такое?')

else:

print('Всё хорошо.')

finally:

f.close()

print('Я закрыл файл.')

*# Именно в таком порядке: try, группа except, затем else, и только потом finally.*

Это не число. Выходим.

Я закрыл файл.

Начало формы

Конец формы