



# Исключения



# Основы исключений

**Исключения** (Exceptions) - ещё один тип данных в Python. Исключения необходимы для того, чтобы сообщать об ошибках.

```
>>> 100/0
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> 2 + '1'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> int('Line')
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Line'
```

# Зачем нужны исключения?

Исключения позволяют перепрыгнуть через фрагмент программы произвольной длины.

Программа может перейти к обработчику исключения за один шаг, отменив все вызовы функций. После этого обработчик исключения может выполнить действия, соответствующие ситуации.

# Пример с функцией fetcher

```
1  def fetcher(obj, index):  
2      return obj[index]  
3  
4  
5  ex = 'word'  
6  # print(fetcher(ex, 2))  
7  # print(fetcher(ex, 10))  
8  
9  try:  
10     fetcher(ex, 15)  
11 except IndexError:  
12     print('Error!!!')  
13     print('But I am still working')  
14
```

# Возбуждение исключений

Чтобы возбудить исключение вручную, достаточно просто выполнить инструкцию `raise`.

Исключения, определяемые программой, перехватываются точно так же, как и встроенные исключения.

```
1  try:
2      raise IndexError
3  except IndexError:
4      print('O_o Error!')
5
```

# Иерархия встроенных в Python исключений

**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.

# Иерархия встроенных в Python исключений

**Exception** - а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать

- **StopIteration** - порождается встроенной функцией next, если в итераторе больше нет элементов
- **ArithmeticError** - арифметическая ошибка
  - **FloatingPointError** - порождается при неудачном выполнении операции с плавающей запятой
  - **OverflowError** - когда результат арифметической операции слишком велик для представления.
  - **ZeroDivisionError** - деление на ноль.
- **AssertionError** - выражение в функции assert ложно.
- **AttributeError** - объект не имеет данного атрибута (значения или метода)
- **ImportError** - не удалось импортирование модуля или его атрибута.
- **LookupError** - некорректный индекс или ключ.
  - **IndexError** - индекс не входит в диапазон элементов.
  - **KeyError** - несуществующий ключ
- **OSError** - ошибка, связанная с системой.
  - **ChildProcessError** - неудача при операции с дочерним процессом.
  - **ConnectionError** - базовый класс для исключений, связанных с подключениями.
  - **FileExistsError** - попытка создания файла или директории, которая уже существует.
  - **TimeoutError** - закончилось время ожидания.
- **RuntimeError** - возникает, когда исключение не попадает ни под одну из других категорий.
- **NotImplementedError** - возникает, когда абстрактные методы класса требуют переопределения в дочерних классах.
- **TypeError** - операция применена к объекту несоответствующего типа.
- **ValueError** - функция получает аргумент правильного типа, но некорректного значения.
- **UnicodeError** - ошибка, связанная с кодированием / декодированием unicode в строках.

## try - except

Для обработки исключений используется конструкция **try - except**.

```
1  try:
2      k = 1 / 0
3  except ZeroDivisionError:
4      k = 0
5
6  print(k)  # 0
7
8  try:
9      a = 1 / 0
10 except ArithmeticError:
11     a = 0
12
13 print(a)  # 0
```



# try, except, finally и else

Для обработки исключений используется конструкция **try - except**.


```
1  f = open('1.txt')
2  ints = []
3  try:
4      for line in f:
5          ints.append(int(line))
6  except ValueError:
7      print('Это не число. Выходим.')
8  except Exception:
9      print('Это что ещё такое?')
10 else:
11     print('Всё хорошо.')
12 finally:
13     f.close()
14     print('Я закрыл файл.')
15     # Именно в таком порядке: try, группа except, затем else, и только потом finally.
16
17
18 # Это не число. Выходим.
19 # Я закрыл файл.
20
21
```

# Свои собственные исключения с переопределениями и именами

```
1 class MyBad(Exception):
2     def __str__(self):
3         return 'Sorry - my mistake!'
4
5
6 try:
7     raise MyBad()
8 except MyBad as X:
9     print(X)
10
```

```
5 try:
6     raise ValueError('Oops')
7 except ValueError as X:
8     print(X) # Oops
```

Thanks for your  
attention

 [artezio\\_software](#)

 [info@artezio.com](mailto:info@artezio.com)

**[www.artezio.com](http://www.artezio.com)**