
# Обработка ошибок

## Технологии и языки программирования

Юдинцев В. В.

Кафедра теоретической механики

25 марта 2019 г.

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

# Синтаксические ошибки

```
1 for i in range(5)
2     print(i)
```

```
SyntaxError: invalid syntax
```

# Ошибки времени выполнения

```
1 a = 1.0
2 for i in range(5):
3     print(a/i)
```

```
...
ZeroDivisionError: division by zero
```

# Обработка ошибок

Для самостоятельной обработки ошибок внутри программы, возникающих во время выполнения, используются ключевые слова `try ... except`:

```
1 try:
2     a = 1.0
3     for i in range(5):
4         print(a/i)
5 except:
6     print('Произошло деление на ноль')
```

Любая ошибка во время выполнения программы внутри блока `try` приведет к выполению кода в блоке `except`.

# Обработка ошибки определённого типа

После ключевого слова `except` можно указать тип ошибки

```
try:
    f = open("datafile.txt", "r")
    a = f.readline()
except IOError:
    print('Невозможно открыть или прочитать файл')
```

- Блок `except IOError:` выполнится только если произойдёт ошибка, связанная с вводом/выводом.
- Ошибки других типов будут обрабатываться объемлющим кодом.

# Дополнительная информация об ошибке

В блоке `except` можно указать имя переменной, которая будет иметь тип ошибки и содержать информацию об ошибке:

```
try:
    f = open("datafile.txt", "r")
    a = f.readline()
except IOError as err:
    print('Невозможно открыть или прочитать файл')
    print('Имя файла:', err.filename)
```

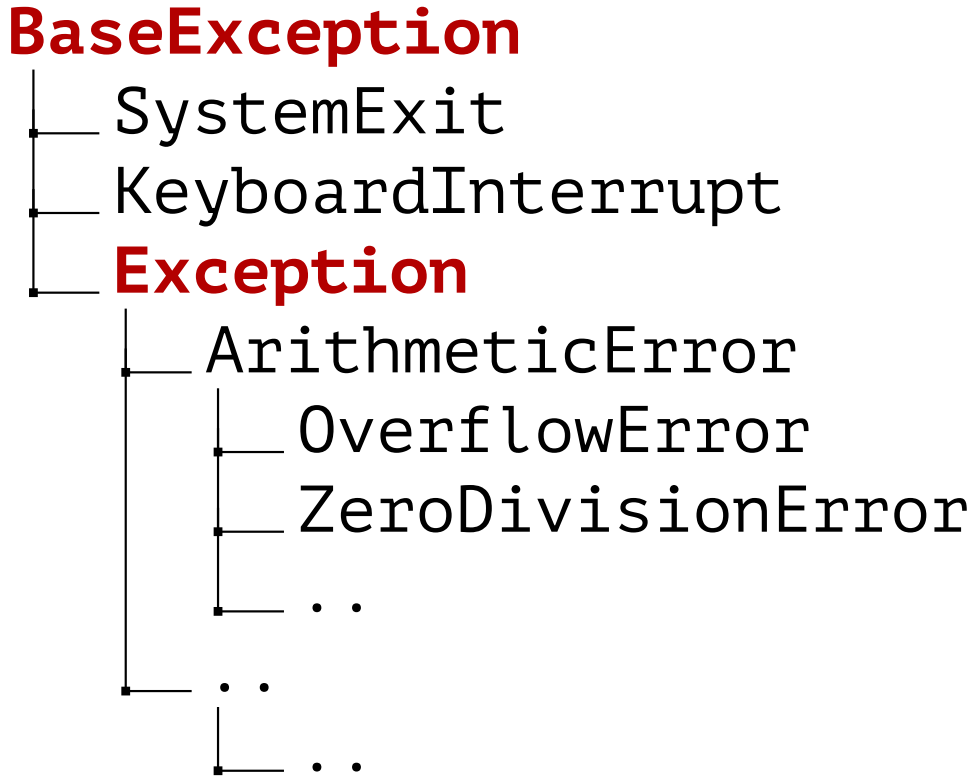
```
Невозможно открыть или прочитать файл
Имя datafile.txt
```

# Типы исключений при работе с файлами

Другие типы исключений для работы с файлами:

- `FileNotFoundError`
  Открываемый файл или каталог не существует
- `FileExistsError`
  Создаваемый файл или каталог уже существует
- `PermissionError`
  Доступ к файлу или каталогу при недостаточном уровне прав

# Иерархия исключений




Все исключения кроме `SystemExit` и `KeyboardInterrupt` являются потомками базового класса `Exception`.

# Конструкция try ... except ... else

В “защищаемом” участке кода делается попытка открыть файл для чтения. Если файл не существует, но генерируется исключение и управление передаётся блоку `except`, иначе выполняется блок `else`:

```
try:
    f = open("datafile.txt", "r")
except FileNotFoundError as err:
    print('Невозможно открыть или прочитать файл')
    print('Имя файла:', err.filename)
else:
    a = f.readline()
```

Переменная `f`, объявленная в блоке `try`, доступна и в блоке `else`.

# Несколько блоков except

Блок `try` может вызывать ошибки различных типов. Для каждого типа ошибки можно создать свой блок `except`, указав её тип:

```
try:
    f = open("datafile.txt", "r")
    str_value = f.readline()
    a = int(str_value)
except FileNotFoundError as err:
    print("Невозможно открыть или прочитать файл")
except ValueError as err:
    print("Ошибка преобразования")
except:
    print("Неизвестная ошибка")

```

# Блок except для нескольких исключений

```
try:
    f = open("datafile.txt", "r")
    str_value = f.readline()
    a = int(str_value)
except (FileNotFoundError, ValueError):
    print("Ошибка загрузки данных из файла")
except:
    print("Неизвестная ошибка")
```

# Блок finally

После блоков `except` и `else` может быть определёт блок `finally`, который выполняется влбом случае:

```
f = open("datafile.txt", "r")
try:
    str_value = f.readline()
    a = int(str_value)
except ValueError as err:
    print("Ошибка преобразования")
finally:
    f.close()
```

Файл закроется при любом исходе.

# Ввод данных с клавиатуры

```
1 s = float(input('Введите основание треугольника'))
2 h = float(input('Введите высоту треугольника'))
3
4 print('Площадь треугольника равна {:1.2f}'.format(0.5*s*h))
```

При вводе не числовых значений программа сообщит об ошибке и остановится:

```
Введите основание треугольника: a
...
...
ValueError: could not convert string to float: 'a'
```

Это плохая реакция программы на ошибку: нет возможности исправить ошибку не перезапуская программу.

# Контроль ввода данных

```
1  def input_as(text, type_of_value):
2      status = True
3      while status:
4          try:
5              val = input(text)
6              val = type_of_value(val)
7              status = False
8          except ValueError as err:
9              print("Это не "+type_of_value.__name__+", попробуйте еще раз.")
10             status = True
11     return val
12
13 val = input_as("Введите целое число: ", int)
14 print("Введено значение", val)
```

# Команда `raise`

Если необходимо после обработки ошибки передать управление обработчику ошибок верхнего уровня, необходимо использовать команду `raise`:

```
1 def divide(a, b):
2     try:
3         res = a/b
4     except:
5         print('divide: b=0!')
6         raise
7     return res
```

```
1 try:
2     a = float(input('a='))
3     b = float(input('b='))
4     divide(a, b):
5 except
6     print('Неверные исходные данные!')
```

```
a: 1
b: 2
divide: b=0!
Неверные исходные данные!
```