# Лекция 5 часть 1 "Обработка исключений"

### Финансовый университет при Правительстве РФ, лектор С.В. Макрушин

v 0.6 16.08.2021

## Разделы:



-


- к оглавлению


In [3]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[3]:

## Обработка исключений

-


- к оглавлению


**Исключения** - это извещения интерпретатора, возбуждаемые в случае возникновения ошибки в программном коде или при наступлении какого-либо события. Если в коде не предусмотрена обработка исключения, то программа прерывается и выводится сообщение об ошибке.

Существуют три типа ошибок в программе:

- *синтаксические* - это ошибки в синтаксисе языка, например: в имени оператора или функции, отсутствие закрывающей или открывающей кавычек и т. д. Как правило, интерпретатор предупредит о наличии ошибки, а программа не будет выполняться совсем. Пример синтаксической ошибки:

In [1]:

```
print("Нет  завершающей кавычки!)
```

```
  File "<ipython-input-1-f572aab57a7b>", line 1
    print("Нет  завершающей кавычки!)
                                      ^
SyntaxError: EOL while scanning string literal
```

- *ошибки времени выполнения, смысловые (семантические)* - это ошибки в логике работы программы, которые возникают во время работы скрипта. Как правило эти ошибки можно выявить только по результатам работы скрипта: интерпретатор не предупреждает о наличии ошибки, а программа будет выполняться вплоть до возникновения ошибки, т. к. не содержит синтаксических ошибок. Классическим примером служит деление на ноль

In [1]:

```
print(10/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-fe01563e1bc6> in <module>
----> 1 print(10/0)

ZeroDivisionError: division by zero
```

Необходимо заметить, что в языке Python исключения возбуждаются не только при ошибке, но и как уведомление о наступлении каких-либо событий. Например, метод index () возбуждает исключение ValueError, если искомый фрагмент не входит в строку:

In [3]:

```
"Строка".index("текст")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-0a329fdc99ca> in <module>()
----> 1 "Строка".index("текст")

ValueError: substring not found
```

## Инструкция try ... except ... else ... finally

-

- к оглавлению

Для обработки исключений предназначена инструкция try. Формат инструкции:

```
try:
    <Блок, в котором перехватьшаются исключения>
[ except [ <Исключение1> [ as <Объект исключения>] ] :
    <Блок, вьтолняемьм при возникновении исключения>
[ ...
except [<ИсключениеN>[ as <Объект исключения>]]:
    <Блок, вьтолняемьм при возникновении исключения>]]
[else:
    <Блок, вьтолняемьм, если исключение не возникло>]
[finally:
    <Блок, вьтолняемьм в любом случае>]
```

Инструкции, в которых перехватываются исключения, должны быть расположены внутри блока `try` . В блоке `except` в параметре <Исключение1> указывается класс обрабатываемого исключения.

Например, обработать исключение, возникающее при делении на ноль, можно так:

In [2]:

```
x = 0
try: # Перехватьшаем исключения
    x = 1/0 # Ошибка: деление на 0
except ZeroDivisionError: # Указьшаем класс исключения
    print("Обработали деление на 0")
    # x = 0
print(x)
```

```
Обработали деление на 0
0
```

1. Если в блоке `try` возникло исключение, то управление передается блоку except.
2. В случае, если исключение не соответствует указанному классу, управление передается следующему блоку `except` .
3. Если ни один блок except не соответствует исключению, то исключение "всплывает" к обработчику более высокого уровня.
4. Если исключение нигде не обрабатывается в программе, то управление передается обработчику по умолчанию, который останавливает выполнение программы и выводит стандартную информацию об ошибке.

In [3]:

```
x = 0
try: # Обрабатыаем исключения
    try: # Вложенным обработчик
        x = 1/0 # Ошибка: деление на 0
    except NameError:
        print("Неопределенный идентификатор")
    except IndexError:
        print("Несуществующий индекс")
    print("Выражение после вложенного обработчика")
except  ZeroDivisionError:
    print("Обработка деления на 0")
    x = 0
print(x) # Выведет: 0
```

```
Обработка деления на 0
0
```

В инструкции except можно указать сразу несколько исключений, перечислив их через запятую внутри круглых скобок:

In [6]:

```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except (NameError, IndexError, ZeroDivisionError):
    x = 0
print(x) # Выведет: 0
```

```
0
```

Получить информацию об обрабатываемом исключении можно через второй параметр в инструкции except:

In [4]:

```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except (NameError, IndexError, ZeroDivisionError) as err:
    print(err.__class__.__name__) #Название  класса  исключения
    print(err) # Текст сообщения об ошибке
```

```
ZeroDivisionError
division by zero
```

Для получения информации об исключении можно воспользоваться функцией exc_info() из модуля sys, которая возвращает кортеж из трех элементов: типа исключения, значения и объекта с трассировочной информацией. Преобразовать эти значения в удобочитаемый вид позволяет модуль traceback.

In [8]:

```
import sys, traceback
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except ZeroDivisionError as err:
    etype, value, trace = sys.exc_info()
    print("Type: {}, Value: {}, Trace: {} ".format(etype, value, trace))
    print("\n", "print_exception() ".center(40,  "-"))
    traceback.print_exception(etype, value, trace, limit=5, file=sys.stdout)
    print("\n",  "print_tb()".center(40,  "-"))
    traceback.print_tb(trace, limit=1, file=sys.stdout)
    print("\n",  "format_exception()".center(40,  "-"))
    print(traceback.format_exception(etype, value, trace, limit=5))
    print("\n",  "format_exception_only()".center(40,"-"))
    print(traceback.format_exception_only(etype, value))
```

```
Type: <class 'ZeroDivisionError'>, Value: division by zero, Trace: <traceback
object at 0x000000A73CA4A388>

 -----------print_exception() -----------
Traceback (most recent call last):
  File "<ipython-input-8-bdbf3e19a700>", line 4, in <module>
    x = 1/0 # Ошибка: деление на 0
ZeroDivisionError: division by zero

 ---------------print_tb()---------------
  File "<ipython-input-8-bdbf3e19a700>", line 4, in <module>
    x = 1/0 # Ошибка: деление на 0

 -----------format_exception()-----------
['Traceback (most recent call last):\n', '  File "<ipython-input-8-bdbf3e19a7
00>", line 4, in <module>\n    x = 1/0 # Ошибка: деление на 0\n', 'ZeroDivisi
onError: division by zero\n']

 --------format_exception_only()---------
['ZeroDivisionError: division by zero\n']
```

Если в инструкции except не указан класс исключения, то такой блок перехватывает все исключения. На практике следует избегать пустых инструкций except, т. к. можно перехватить исключение, которое является лишь сигналом системе, а не ошибкой.

In [9]:

```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except:
    x = 0
print(x) # Выведет: 0
```

```
0
```

Если в обработчике присутствует блок else, то инструкции внутри этого блока будут выполнены только при отсутствии ошибок. При необходимости *выполнить какие-либо завершающие действия* вне зависимости от того, возникло исключение или нет, следует воспользоваться *блоком finally*.

In [10]:

```
x = 0
try: # Обрабатьшаем исключения
#     x = 1/0 # Ошибка: деление на 0
    x = 1/10
except ZeroDivisionError:
    print('Деление на 0')
    x = 0
else:
    print('Блок else')
finally:
    print('Блок finally')
print(x) # Выведет: 0
```

```
Блок else
Блок finally
0
```

Необходимо заметить, что при наличии исключения и отсутствии блока except инструкции внутри блока finally будут выполнены, но исключение не будет обработано. Оно продолжит "всплывание" к обработчику более высокого уровня. Если пользовательский обработчик отсутствует, то управление передается обработчику по умолчанию, который прерывает выполнение программы и выводит сообщение об ошибке.

In [6]:

```
x = 0
try: # Обрабатьшаем исключения
    x = 1/0 # Ошибка: деление на 0
    x = 1/10
finally:
    print('Блок finally')
```

```
Блок finally

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-6-9f07fde92c29> in <module>
      1 x = 0
      2 try: # Обрабатьшаем исключения
----> 3     x = 1/0 # Ошибка: деление на 0
      4     x = 1/10
      5 finally:

ZeroDivisionError: division by zero
```

## Классы встроенных исключений

-

- к оглавлению

- BaseException
  - GeneratorExit

- Keyboardinterrupt
- SystemExit
- Exception
  - Stopiteration
  - Warning
    - BytesWarning, ResourceWarning,
    - DeprecationWarning, FutureWarning, ImportWarning,
    - PendingDeprecationWarning, RuntimeWarning, SyntaxWarning,
    - UnicodeWarning, UserWarning
  - ArithmeticError
    - FloatingPointError, OverflowError, ZeroDivisionError
  - AssertionError
  - AttributeError
  - BufferError
  - EnvironmentError
    - IOError
    - OSError
      - WindowsError
  - EOFError
  - ImportError
  - LookupError
    - IndexError, KeyError
  - MemoryError
  - NameError
    - UnboundLocalError
  - ReferenceError
  - RuntimeError
    - NotimplementedError
  - SyntaxError
    - IndentationError
      - TabError
  - SystemError
  - TypeError
  - ValueError
    - UnicodeError
      - UnicodeDecodeError, UnicodeEncodeError
      - UnicodeTranslateError

Основное преимущество использования классов для обработки исключений заключается в возможности указания базового класса для перехвата всех исключений соответствующих классов-потомков. Например, для перехвата деления на ноль мы использовали класс ZeroDivisionError. Если вместо этого класса указать базовый класс ArithrneticError, то будут перехватываться исключения классов FloatingPointError, OverflowError и ZeroDivisionError.

In [7]:

```
x = 0
try: # Обрабатыаем исключения
    x = 1/0 # Ошибка: деление на 0
except ArithmeticError: # Указываем базовый класс
    print('Обработка деления на 0')
    x = 0
print(x) # Выведет: 0
```

```
Обработка деления на 0
0
```

Рассмотрим основные классы встроенных исключений:

- BaseException - является классом самого верхнего уровня;
- Exception - именно этот класс, а не BaseException, необходимо наследовать при создании пользовательских классов' исключений;
- AssertionError - возбуждается инструкцией assert;
- AttributeError - попытка обращения к несуществующему атрибуту объекта;
- EOFError - возбуждается функцией input() при достижении конца файла;
- IOError - ошибка доступа к файлу;
- ImportError - невозможно подключить модуль или пакет;
- IndentationError - неправильно расставлены отступы в программ е;
- IndexError - указанный индекс не существует в последовательности;
- KeyError - указанный ключ не существует в словаре;
- KeyboardInterrupt - нажата комбинация клавиш +;
- NameError - попытка обращения к идентификатору до его определения;
- Stoprteration - возбуждается методом _ next _ () как сигнал об окончании итераций;
- SyntaxError - синтаксическая ошибка;
- TypeError - тип объекта не соответствует ожидаемому;
- UnboundLocalError - внутри функции переменной присваивается значение после обращения к одноименной глобальной переменной;
- UnicodeDecodeError - ошибка преобразования последовательности байтов в строку;
- UnicodeEncodeError - ошибка преобразования строки в последовательность байтов;
- ValueError - переданный параметр не соответствует ожидаемому значению;
- ZeroDivisionError - попьпка· деления на ноль.

# Пользовательские исключения

-


- к оглавлению


Инструкция raise возбуждает указанное исключение. Она имеет несколько форматов:

- raise <Экземпляр класса>
- raise <Название класса>
- raise <Экземпляр или название класса> from <Объект исключения>
- raise

В первом формате инструкции `raise` указывается экземпляр класса возбуждаемого исключения. При создании экземпляра можно передать данные конструктору класса. Эти данные будут доступны через второй параметр в инструкции except.

In [8]:

```
try:
    raise ValueError("Описание исключения")
except ValueError  as  msg:
    print(msg) # Выведет: Описание исключения
```

```
Описание исключения
```

In [9]:

```
try:
    raise ValueError
except ValueError:
    print('Сообщение об ошибке')
```

```
Сообщение об ошибке
```

# Инструкция assert

-


- к оглавлению


Инструкция assert возбуждает исключение AssertionError, если логическое выражение возвращает значение False. Инструкция имеет следующий формат:

assert <Логическое выражение> [, <Сообщение>]

Инструкция assert эквивалентна следующему коду:

```
if __debug__:
    if not <Логическое  выражение>:
        raise  AssertionError(<Сообщение>)
```

Если при запуске программы используется флаг -o, то переменная __debug__ будет иметь ложное значение. Таким образом можно удалить все инструкции assert из байт-кода.

In [10]:

```
try:
    x = -3
    assert x >= 0, "Сообщение об ошибке"
except AssertionError as err:
    print(err)  # Выдает: Сообщение об ошибке
```

```
Сообщение об ошибке
```

In [11]:

```
def factorial(n):
    """Возвращает Факториал числа n.
    Аргумент n - не отрицательное целое число."""
    assert n >= 0, 'Аргумент n должен быть больше 0!'
    assert n % 1 == 0, 'Аргумент n должен быть целым!'
    f = 1
    for i in range(2, n+1):
        f *= i
    return f
```

In [12]:

```
factorial(-1)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-12-5aae425d6a8b> in <module>
----> 1 factorial(-1)

<ipython-input-11-4fb2c5db8c0e> in factorial(n)
      2     """Возвращает Факториал числа n.
      3     Аргумент n - не отрицательное целое число."""
----> 4     assert n >= 0, 'Аргумент n должен быть больше 0!'
      5     assert n % 1 == 0, 'Аргумент n должен быть целым!'
      6     f = 1

AssertionError: Аргумент n должен быть больше 0!
```

In [20]:

```
factorial(5.5)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-20-347bc4de69b9> in <module>()
----> 1 factorial(5.5)

<ipython-input-18-8ec99d38a0cf> in factorial(n)
      3     Аргумент n - не отрицательное целое число."""
      4     assert n >= 0, 'Аргумент n должен быть больше 0!'
----> 5     assert n % 1 == 0, 'Аргумент n должен быть целым!'
      6     f = 1
      7     for i in range(2, n+1):

AssertionError: Аргумент n должен быть целым!
```

In [21]:

```
factorial(0)
```

Out[21]:

```
1
```

In [23]:

```
factorial(1)
```

Out[23]:

```
1
```

In [24]:

```
factorial(2)
```

Out[24]:

```
2
```

In [25]:

```
factorial(5)
```

Out[25]:

```
120
```