MIPT
MOSCOW INSTITUTE OF PHYSICS AND TECHNOLOGY

АТП кафедра Алгоритмов и Технологий Программирования

Исключения, модули, регулярные выражения.

Емельянов А. А.
login-const@mail.ru

2018

https://github.com/king-menin/python-course

Типы ошибок

- Синтаксические ошибки (ошибки ~~первого рода~~ разбора кода)

- Исключения (ошибки ~~второго рода~~ выполнения кода)

```
In [2]: 10 * (1/0)
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-9ce172bd90a7> in <module>()
----> 1 10 * (1/0)

ZeroDivisionError: division by zero
```

Ограничения на работу программы

- assertions – нужны для формулирования утверждений, которые никогда не должны нарушаться.

```
In [31]: def summ(*args):
             assert all(list(map(lambda x: isinstance(x, int), args))), \
                 "arguments must be int. Passed: {}".format(list(map(type, args)))
             return sum(args)
```

```
In [32]: summ(1, "1")
```

  - код выше эквивалентен следующему:

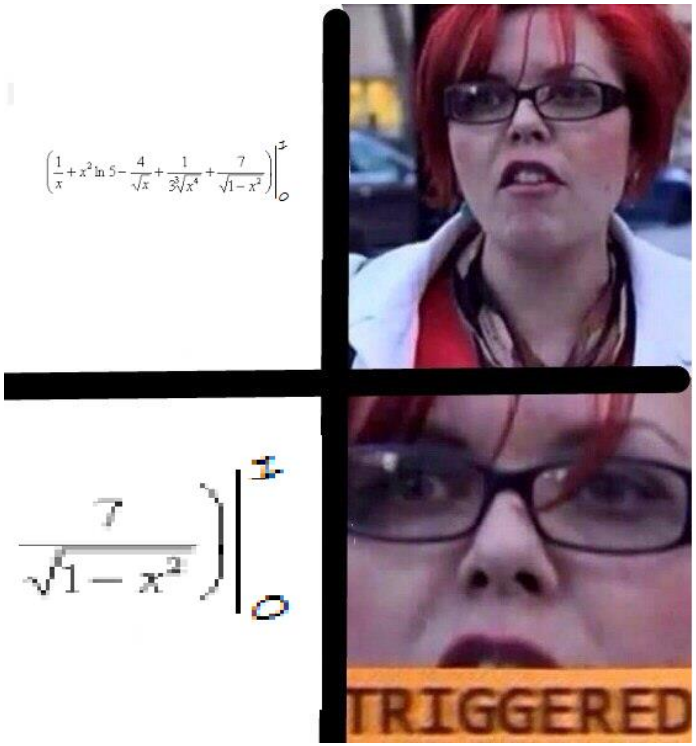
```
In [33]: def summ(*args):
             if __debug__:
                 if not all(list(map(lambda x: isinstance(x, int), args))):
                     raise AssertionError("arguments must be int. Passed: {}".format(list(map(type, args))))
             return a + b
```

```
In [34]: summ(1, "1")
```

- **__debug__** — встроенная константа, по умолчанию имеющая значение True. Если интерпретатор запущен в режиме оптимизации (с флагом командной строки -O), значение константы становится False, а генератор кода перестаёт производить байткод для рассматриваемой инструкции. Таким образом, отключив проверки, но не убирая их из кода, можно снизить неизбежные для них накладные расходы.

Исключения (Exceptions)

- Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution[1].



1. ВИКИПЕДИЯ: https://en.wikipedia.org/wiki/Exception_handling

Подходы к обработке исключений

- Look before you leap

In [64]:
```
def summ(*args):
    def process(args):
        processed = []
        for arg in args:
            if isinstance(arg, str):
                processed.append(int(arg))
            elif isinstance(arg, float):
                processed.append(int(arg))
            elif isinstance(arg, (list, dict, tuple)):
                processed.extend(process(arg))
            elif isinstance(arg, bool):
                processed.append(arg)
            elif isinstance(arg, int):
                processed.append(arg)
            # 100 elif
            else:
                print("Exclude undefined type: {}".format(type(arg)))
        return processed
    return sum(process(args))
```

In [65]:
```
summ(1, [1, 2, 3], "2")
```

Out[65]: 9

In [73]:
```
summ(1, [1, 2, 3], "2q")
```

```
Exclude undefined type: <class 'str'>
```

Out[73]: 7

Подходы к обработке исключений

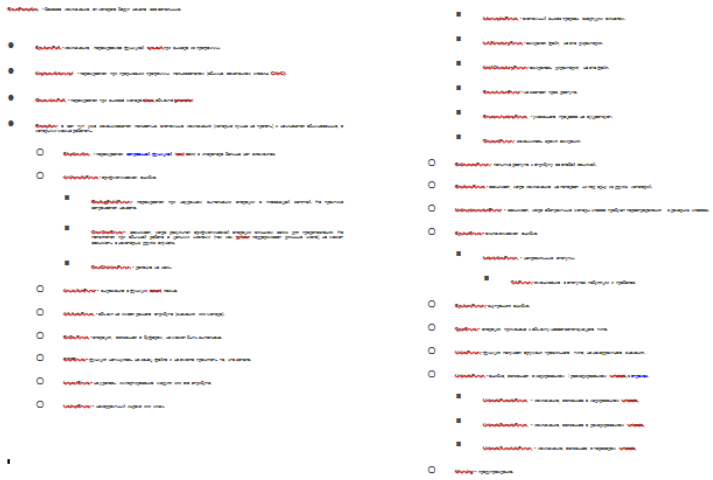
- Easier to ask for forgiveness than permission

```
In [71]: def summ(*args):
             def process(arg):
                 if isinstance(arg, (list, dict, tuple)):
                     return list(map(int, arg))
                 return [int(arg)]
             res = []
             for arg in args:
                 try:
                     res.append(sum(process(arg)))
                 except:
                     print("Exclude undefined type: {}".format(type(arg)))
             return sum(res)

In [75]: summ(1, [1, 2, 3], "2")

Out[75]: 9

In [74]: summ(1, [1, 2, 3], "2q")

         Exclude undefined type: <class 'str'>

Out[74]: 7
```

– так лучше ☺

Встроенные типы исключений

- Их очень много. Например: : MemoryError, ValueError, TypeError, ImportError, HTTPError, ZeroDivisionError, UnicodeDecodeError, NameError, AttributeError, NotImplementedError.



- **BaseException** - базовое исключение, от которого берут начало все остальные.
  - **Exception** - а вот тут уже заканчиваются полностью системные исключения (которые лучше не трогать) и начинаются обыкновенные, с которыми можно работать.

Обработка исключений

- Обработка исключений

```
In [76]: try:
             k = 1 / 0
         except ZeroDivisionError:
             k = 0
         print(k)
```

```
0
```

- Системные исключения

```
def train(self, x_train, y_train, x_test, y_test):
        train_info, test_info = [], []
        try:
            for epoch in range(self.n_epochs):
                start_time = time.time()
                self.__update_optimizer(epoch)
                self.__update_reg_weight(epoch)
                train_info.append(self.__iteration(x_train, y_train, self.train_func))
                test_info.append(self.__iteration(x_test, y_test, self.test_func))
                end_time = time.time()
                self.__print_iteration_info(start_time, end_time, train_info, test_info, epoch)
        except KeyboardInterrupt:
            print('Training was stoped.')
```

try .. except .. else

In [24]:

```
def zd(a):
    return a / 0

def ve(a):
    # ValueError
    return int(a)

def caller(f, *args):
    try:
        k = f(*args)
    except ZeroDivisionError:
        k = 0
        print("Handled")
    else:
        print(k)
```

raise

In [44]:
```
from functools import wraps
def calc_errors(func):
    @wraps(func)
    def res(*args):
        try:
            return func(*args)
        except :
            res.error_count += 1
            raise
    res.error_count = 0
    return res
@calc_errors
def zd(a):
    return a / 0
```

finally

In [56]:
```
from functools import wraps
from collections import Counter

def calc_call_stats(func):
    @wraps(func)
    def res(*args):
        try:
            result = func(*args)
        except :
            res.counts["errors"] += 1
            raise
        else:
            res.counts["successes"] += 1
        finally :
            res.counts["calls"] += 1
        return result
    res.counts = Counter()
    return res
@calc_call_stats
def zd(a):
    return 1 / a
```

Доступ к объекту исключения

In [63]:
```
try:
    k = 1 / 0
except Exception as e:
    print("something wrong {}.".format(e))
    raise
```

```
something wrong division by zero.

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-63-c60dd6dd26c5> in <module>()
      1 try:
----> 2     k = 1 / 0
      3 except Exception as e:
      4     print("something wrong {}.".format(e))
      5     raise

ZeroDivisionError: division by zero
```

Несколько except

In [65]:
```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as err:
    print("I/O error: {0}".format(err))
except ValueError:
    print("Не могу преобразовать данные в целое.")
except:
    print("Неожиданная ошибка:", sys.exc_info()[0])
    raise
```

```
I/O error: [Errno 2] No such file or directory: 'myfile.txt'
```

Порождение исключений

```
In [71]: test("2q")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-71-60c0f9ce90ab> in <module>()
----> 1 test("2q")

<ipython-input-70-4ae88af2265b> in test(x)
      1 def test(x):
      2     if not isinstance(x, int):
----> 3         raise TypeError("test() arg must be int")
      4     return x + 5

TypeError: test() arg must be int
```

Исключения, определённые пользователем

In [74]:
```
class MyError(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)
```

In [75]:
```
try:
    raise MyError(2*2)
except MyError as e:
    print('Поймано моё исключение со значением:', e.value)
```

```
Поймано моё исключение со значением: 4
```

Исключения, определённые пользователем

In [78]:
```
#Собственные исключения
class ShoeError(Exception):
    """Basic exception for errors raised by shoes"""
    pass
class UntiedShoelace(ShoeError):
    """You could fall"""
    pass

class WrongFoot(ShoeError):
    """When you miss left and right"""
    pass
```

- Хорошая библиотека должна содержать детальные исключения

raise from

```
In [80]: try:
             zd(0)
         except ZeroDivisionError as e:
             raise TypeError("actually, type error") from e
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-80-5bc0b43730c0> in <module>()
      1 try:
----> 2     zd(0)
      3 except ZeroDivisionError as e:

<ipython-input-56-64d9d2a404c5> in res(*args)
      7         try:
----> 8             result = func(*args)
      9         except :

<ipython-input-56-64d9d2a404c5> in zd(a)
     20 def zd(a):
---> 21     return 1 / a

ZeroDivisionError: division by zero

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
<ipython-input-80-5bc0b43730c0> in <module>()
      2     zd(0)
      3 except ZeroDivisionError as e:
----> 4     raise TypeError("actually, type error") from e

TypeError: actually, type error
```

Библиотеки для работы с исключениями

- sys.exc_info, sys.last_traceback
- модуль traceback

Контекстные менеджеры

- Ещё одна синтаксическая конструкция

```
In [ ]: with open("what_is_love.txt", "r") as infile:
            for line in infile:
                print("> {}".format(line))
```

- Нужны в первую очередь для управления ресурсами

Собственные контекстные менеджеры

In [110]:
```
class File():
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.open_file = open(self.filename, self.mode)
        return self.open_file

    def __exit__(self, *args):
        self.open_file.close()
```

In [ ]:
```
with File("foo.txt", "w") as infile:
    for _ in range(10000):
        infile.write('foo')
```

Модули в python

- это любой текстовый файл с кодом, имя которого заканчивается на .py
- внутри модуля доступна переменная __name__
- модуль можно импортировать, выполнить его, вернув ссылку на его глобальное пространство имён. При импорте переменная __name__ равна имени файла модуля
- модуль можно выполнить, тогда в переменной __name__ будет специально значение __main__
- при импорте интерпретатор пытается найти модуль в локальной папке и путях перечисленных в PATH и PYTHONPATH

Импорт модуля

- Файл pprint.py

```
In [ ]: windows_tmp = "Run this package on windows: \"{}\""


        def pprint(val):
            print(windows_tmp.format(val))


        if __name__ == "__main__":
            pprint("test")
```

- Использование

```
In [1]: import myprinter

In [2]: myprinter.pprint("Some text")
        Run this package on windows: "Some text"
```

Импорт модуля

```
In [5]: import myprinter

In [6]: import myprinter as mp

In [7]: from myprinter import pprint

In [8]: from myprinter import *
```

При

```
import a.b.c
```

будут последовательно выполнены

```
import a
import a.b
import a.b.c
```

Это нужно знать.

~~from sklearn import *~~

Атрибуты модулей

- __name__: str - полное имя модуля. Путь от начала с точками как разделителями. Например, 'xml.dom' или 'xml.dom.minidom‘.
- __doc__: str - описание (так называемый *docstring*).
- __file__: str - полный путь к файлу, из которого модуль был создан (загружен).
- __path__: [str] - список файловых путей, в которых находится пакет. Существует только для пакетов. Об этом атрибуте я расскажу чуть позже более подробно.
- __cached__: str - [3.2+] нововведение, появившееся в Python 3.2. Путь к .pyc файлу.
- __package__: str - [2.5+] имя пакета, в котором лежит модуль (пустая строка для модулей верхнего уровня). Появился для поддержки относительного импорта from . import a.
- __loader__: Loader - [2.3+] ссылка на объект, который выполнял загрузку данного модуля. Присутствует только для тех модулей, которые были обработаны через механизм расширения импорта.

Пакеты

- **Пакет** (*package*) - разновидность модуля, используемая для собирания модулей в иерархическую древовидную структуру. Классические папки с __init__.py внутри.

```
printer
        linux2
                printer.py
                __init__.py
        win32
                printer.py
                __init__.py
                __pycache__
                        printer.cpython-36.pyc
        __init__.py
        __pycache__
                __init__.cpython-36.pyc
```

Пакеты: пример

- Поддержка разных операционных систем. Структура пакета:

```
printer
        linux2
                printer.py
                __init__.py
        win32
                printer.py
                __init__.py
                __pycache__
                        printer.cpython-36.pyc
        __init__.py
        __pycache__
                __init__.cpython-36.pyc
```

Пакеты: пример

- Мы кладем "общий" код непосредственно в package, а платформозависимый разносим по вложенным (технически они могут находится где угодно) папкам. Обратите внимание - linux2 и win32 не содержат __init.py__ и не являются вложенными пакетами.
- А в __init__.py пишем что-то вроде:

```
[n [ ]:  import sys
         from os.path import join, dirname
         __path__.append(join(dirname(__file__), sys.platform))
```

- Наслаждаемся:

```
In [43]: import os
         from printer.printer import pprint

In [44]: pprint("test")

         Run this package on windows: "test"
```

Регулярные выражения

- Регулярные выражения используют два типа символов:
  - специальные символы: как следует из названия, у этих символов есть специальные значения. Аналогично символу *, который как правило означает «любой символ» (но в регулярных выражениях работает немного иначе, о чем поговорим ниже);
  - литералы (например: a, b, 1, 2 и т. д.).

- В Python для работы с регулярными выражениями есть модуль re.

- Вот наиболее часто используемые из них:
  - re.match, re.search, re.findall, re.split, re.sub, re.compile

Регулярные выражения: match

- re.match(*pattern*, *string*):
- Этот метод ищет по заданному шаблону в начале строки. Например, если мы вызовем метод match() на строке «AV Analytics AV» с шаблоном «AV», то он завершится успешно. Однако если мы будем искать «Analytics», то результат будет отрицательный.

```
In [49]: import re
         res = re.match(r'AV', 'AV Analytics Vidhya AV')
         print(res)
         print(res.group(0))

         <_sre.SRE_Match object; span=(0, 2), match='AV'>
         AV

In [51]: print(re.match(r'Analytics', 'AV Analytics Vidhya AV'))

         None
```

Регулярные выражения: search

- re.search(*pattern*, *string*):
- Этот метод похож на match(), но он ищет не только в начале строки. В отличие от предыдущего, search() вернет объект, если мы попытаемся найти «Analytics».
  Метод search() ищет по всей строке, но возвращает только первое найденное совпадение.

```
In [52]: re.search(r'Analytics', 'AV Analytics Vidhya AV')
Out[52]: <_sre.SRE_Match object; span=(3, 12), match='Analytics'>
```

Регулярные выражения: findall

- re.findall(*pattern*, *string*):
- Этот метод возвращает список всех найденных совпадений. У метода findall() нет ограничений на поиск в начале или конце строки. Если мы будем искать «AV» в нашей строке, он вернет все вхождения «AV». Для поиска рекомендуется использовать именно findall(), так как он может работать и как re.search(), и как re.match().

```
In [53]: re.findall(r'AV', 'AV Analytics Vidhya AV')
Out[53]: ['AV', 'AV']
```

Регулярные выражения: split

- re.split(*pattern*, *string*, *[maxsplit=0]*):
- Этот метод разделяет строку по заданному шаблону.

```
In [54]: re.split(r'y', 'Analytics')
Out[54]: ['Anal', 'tics']
```

- В примере мы разделили слово «Analytics» по букве «y». Метод split() принимает также аргумент maxsplit со значением по умолчанию, равным 0. В данном случае он разделит строку столько раз, сколько возможно, но если указать этот аргумент, то разделение будет произведено не более указанного количества раз.

```
In [56]: print(re.split(r'i', 'Analytics Vidhya'))
         print(re.split(r'i', 'Analytics Vidhya',maxsplit=1))

         ['Analyt', 'cs V', 'dhya']
         ['Analyt', 'cs Vidhya']
```

Регулярные выражения: sub

- re.sub(*pattern*, *repl*, *string*):
- Этот метод ищет шаблон в строке и заменяет его на указанную подстроку. Если шаблон не найден, строка остается неизменной.

```
In [57]: re.sub(r'India', 'the World', 'AV is largest Analytics community of India')
Out[57]: 'AV is largest Analytics community of the World'
```

Регулярные выражения: compile

- re.compile(*pattern*, *repl*, *string*):
- Мы можем собрать регулярное выражение в отдельный объект, который может быть использован для поиска. Это также избавляет от переписывания одного и того же выражения.

```
In [58]: pattern = re.compile('AV')
         result = pattern.findall('AV Analytics Vidhya AV')
         print(result)
         result2 = pattern.findall('AV is largest analytics community of India')
         print(result2)

         ['AV', 'AV']
         ['AV']
```

Регулярные выражения: основные символы

Оператор	Описание
`.`	Один любой символ, кроме новой строки \n.
`?`	0 или 1 вхождение шаблона слева
`+`	1 и более вхождений шаблона слева
`*`	0 и более вхождений шаблона слева
`\w`	Любая цифра или буква (\W — все, кроме буквы или цифры)
`\d`	Любая цифра [0-9] (\D — все, кроме цифры)
`\s`	Любой пробельный символ (\S — любой непробельнй символ)
`\b`	Граница слова
`[..]`	Один из символов в скобках ([^..] — любой символ, кроме тех, что в скобках)
`\`	Экранирование специальных символов (\. означает точку или \+ — знак «плюс»)
`^` и `$`	Начало и конец строки соответственно
`{n,m}`	От n до m вхождений ({,m} — от 0 до m)
`a|b`	Соответствует a или b
`()`	Группирует выражение и возвращает найденный текст
`\t, \n, \r`	Символ табуляции, новой строки и возврата каретки соответственно

- Более подробная информация: https://docs.python.org/3/library/re.html

Домашнее задание 5

- **Целью этого задания является знакомство с контекстными менеджерами и регулярными выражениями.**
- **Deadline (получение полных баллов):** 05.04.2018

- **Адрес:** login-const@mail.ru

- Задание состоит из трех частей (разных декораторов):
  - cached,
  - checked,
  - Logger.

- **Текс условия доступен по** ссылке.

СПАСИБО ЗА ВНИМАНИЕ