Лекция 11

Язык программирования Python.

Хайрулин Сергей Сергеевич

email: s.khairulin@g.nsu.ru, s.khayrulin@gmail.com

Ссылка на материалы

План

- Лекции/практические занятия
  - Тест
- Дифференцированный зачет в конце семестра
  - Защита задания

Литература

Начальный уровень

- Mark Pilgrim. Dive into Python - http://www.diveintopython.net/
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

Стандарт/Документация

- PEP-8 - https://www.python.org/dev/peps/pep-0008/
- https://www.python.org/
- https://github.com/python/cpython

Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги(ошибки) с января 2020 года поддержка прекращена.
- Python 3 в появился в 2008, является актуальной версией языка. Текущая стабильная версия 3.9, в разработке 3.10
  - Python 3 не гарантирует совместимости кода с Python 2

Summary

- Обработка исключительных ситуаций.
  - Конструкция: **try … except …**
  - Конструкция: **try … except … finally**
  - Пользовательские классы исключений
- Форматирование строк
- Магические методы
- Сборка мусора

Обработка аварийных/исключительных ситуаций.

Для того чтобы стабилизировать кодовую базу и предотвратить аварийные выходы программы, в том числе в связи с неправильными данными. Язык Python предоставляет программистам возможность обрабатывать аварийные ситуации с помощью механизма перехвата исключений.

Обработка аварийных/исключительных ситуаций.

```
In [1]: 1 / 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-bc757c3fda29> in <module>
----> 1 1 / 0

ZeroDivisionError: division by zero
```

```
In [2]: 1 + "abc"
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-87be4b1bc3f8> in <module>
----> 1 1 + "abc"

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

Конструкция: try … except …

Для того чтобы перехватить исключение в программном блоке блок помещается внутрь конструкции

```
try:
    do_smth
except …:
    do_smth_if_exception
```

Конструкция: try … catch …

exception.py > ...

```
1  if __name__ == '__main__':
2      try:
3          div = int(input("Input num: "))
4          print(1/div)
5      except ZeroDivisionError as err:
6          print(f"You're trying to dievice 1 by zero {err}")
7
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[14:45:09] serg :: serg-pc  ➜  ~/tmp/oop»
python3 exception.py
Input num: 10
0.1
[14:45:16] serg :: serg-pc  ➜  ~/tmp/oop»
python3 exception.py
Input num: 0
You're trying to dievice 1 by zero division by zero
[14:45:46] serg :: serg-pc  ➜  ~/tmp/oop»
```

Конструкция: try … except …

exception.py > ...

```
1   if __name__ == '__main__':
2       try:
3           div = int(input("Input num: "))
4           print(1/div)
5       except ZeroDivisionError as err:
6           print(f"You're trying to dievice 1 by zero {err}")
7       except ValueError as err:
8           print(f"Wrong value err: {err}")
9
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[14:55:18] serg :: serg-pc  ➜  ~/tmp/oop»
python3 exception.py
Input num: dsds
Wrong value err: invalid literal for int() with base 10: 'dsds'
```

Конструкция: try … catch …

```python
if __name__ == '__main__':
    try:
        div = int(input("Input num: "))
        print(1/div)
    except ZeroDivisionError as err:
        print(f"You're trying to dievice 1 by zero {err}")
    except ValueError as err:
        print(f"Wrong value err: {err}")
    except Exception:
        print(f"Some bad happend: {err}")
    except:  # <=> except BaseException
        print(f"Some REALY bad happend: {err}")
```

Конструкция: try … except …finally

Если нужно организовать работы некоторого программного блока в независимости от того было ли исключение или нет, то можно воспользоваться конструкцией **try … except … finally**. Блок кода определенный внутри finally будет вызван в любом случае было ли исключение или нет. Это позволяет в том числе организовать единое место освобождение ресурсов.

```
In [5]: try:
   ...:     db = connect_to_db()
   ...:     db.push_data('data')
   ...: except DataError as err:
   ...:     print(f"data error {err=}")
   ...: except DBError as err:
   ...:     print(f"Lost connection to DB {err=}")
   ...: finally:
   ...:     release_connect(db)
```

Ситуация

```
In [3]: def foo():
   ...:     x = 1
   ...:     try:
   ...:         print("In try block")
   ...:         return x
   ...:     except:
   ...:         print("In exception handling")
   ...:         return x + 2
   ...:     finally:
   ...:         print("In finally block")
   ...:         return x + 3
```

Пользовательские классы исключений

BaseException - системные ошибки.

Exception - Все встроенные исключения, не связанные с системными ошибками, являются производными от этого класса. Все пользовательские исключения также должны быть производными от этого класса.

Полный список встроенных исключений можно найти здесь.

StopIteraction

exception.py > MyIter > __next__

```python
class MyIter:
    def __init__(self, end, start=0, step=1):
        self.__start = start
        self.__end = end
        self.__step = step

    def __iter__(self):
        return self

    def __next__(self):
        if self.__start + self.__step >= self.__end:
            raise StopIteration("stop interation under my iter")
        self.__start += self.__step
        return self.__start

if __name__ == '__main__':
    for i in MyIter(5):
        print(i)
```

```
python exception.py
1
2
3
4
```

Пользовательские классы исключений

С помощью ключевого слова **raise** - можно возбуждать исключение, при этом программа останавливает выполнение и управление передается подходящему блоку catch - тип исключение соответствует типу обрабатываемого. Если подходящего блок не найден, то программа прекращает выполнение и останавливается, при этом в стандартный поток вывода, выводиться информация об ошибке.

```
class MyException(Exception):
    def __str__(self):
        return "HaHa you're on the wrong way"


if __name__ == '__main__':
    try:
        raise MyException()
    except MyException as e:
        print(e)
```

Пользовательские классы исключений

```python
from math import sqrt


class SQRTError(Exception):
    def __str__(self):
        return "Check you're input case we're not working with complex number"


def f(y):
    if y == 0:
        raise ZeroDivisionError("Y is equal to 0")
    return 1/y


def my_sqrt(n):
    if n < 0:
        raise SQRTError()
    return sqrt(n)


if __name__ == '__main__':
    try:
        f(2)
        my_sqrt(-1)
    except ZeroDivisionError as e:
        print(e)
    except SQRTError as e:
        print(e)

```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

[15:09:26] serg :: serg-pc  → ~/tmp/oop»
python3 exception.py
Y is equal to 0
[15:09:27] serg :: serg-pc  → ~/tmp/oop»
python3 exception.py
Check you're input case we're not working with complex number
```

Форматирование строк.

Старый стиль форматирования строк, иногда может быть полезен..

```
In [2]: var = "world"

In [3]: "Hello %s"%(var)
Out[3]: 'Hello world'

In [4]: var2 = "It's me"

In [5]: "Hello %s %s"%(var, var2)
Out[5]: "Hello world It's me"
```

Форматирование строк (format).

```
In [9]: "Hello {0} It's {1}".format("world", "me")
Out[9]: "Hello world It's me"
```

```
In [10]: "Hello {k1} It's {k2}".format(k1="world", k2="me")
Out[10]: "Hello world It's me"
```

Форматирование строк (f string).

fstring появились с релизом python 3.4 - позволяют форматировать строки на лету подставляя текущее значение переменной.

```
In [13]: who = "me"

In [14]: f"Hello {var} It's {who}"
Out[14]: "Hello world It's me"
```

!Переменные должны быть определены на момент формирования строки. Иначе это приведет к ошибкам.

Магические методы.

Магические методы (Dunder methods) - это подход в python к *перегрузке операторов*, позволяющий классам определять свое поведение в *отношении операторов языка*. Подобные методы добавляются в реализацию класса и должны называться определенным образом

def __<meth_name>__(args....):

Со всем списков методов и описанием можно ознакомиться в этой статье.

Магические методы.

```
In [5]: class A:
   ...:     def __init__(self, data):
   ...:         self.__acum = data
   ...:     def __str__(self):
   ...:         return str(self.__acum)
   ...:     def __add__(self, other):
   ...:         self.__acum += other
   ...:

In [6]: a1 = A(10)

In [7]: a1 + 5

In [8]: print(a1)
15
```

Магические методы.

matrix.py > Matrix > __mul__

```python
class Matrix:
    def __init__(self, data):
        """
        :param data: list[list[float]]
        """

        self.matrix = data

    def __str__(self):
        result = ""
        for row in self.matrix:
            tmp = ""
            for item in row:
                tmp += str(item) + " "
            result += tmp + "\n"
        return result

    def mult_scalar(self, scalar):
        for i in range(len(self.matrix)):
            for j in range(len(self.matrix[i])):
                self.matrix[i][j] *= scalar

    def __mul__(self, m):
        if isinstance(m, (float, int)):
            for i in range(len(self.matrix)):
                for j in range(len(self.matrix[i])):
                    self.matrix[i][j] *= m
            return self

        elif isinstance(m, Matrix):
            raise NotImplementedError("It's en exircise!")
```

Магические методы.

```
In [10]: class A:
    ...:     def __init__(self, data):
    ...:         self.__acum = data
    ...:     def __str__(self):
    ...:         return str(self.__acum)
    ...:     def __int__(self):
    ...:         return self.__acum
    ...:

In [11]: a = A(10)

In [12]: print(a)
10

In [13]: int(a)
Out[13]: 10
```

Освобождение ресурсов - garbage collector.

В python реализован алгоритм сборки мусора, который удаляет объекты(освобождает память занятую этими объектами) из памяти. Такими образом вам не нужно заботиться об утечках памяти, до тех пор **пока вы сами ее не сделаете.**

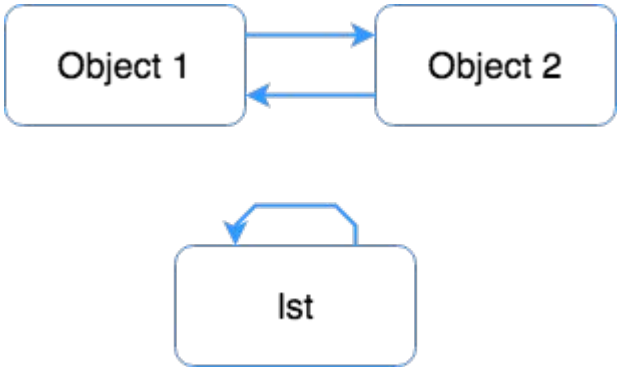
Об алгоритме можно почитать здесь

Освобождение ресурсов - garbage collector.

На самом деле алгоритма 2

1. Подсчет ссылок

2. Сканирование на наличее циклических ссылок



Спасибо за внимание. Вопросы?