Про память

в Python

Обо мне

Юлия Волкова (Iuliia Volkova)

Python Developer в  Grid Dynamics

 https://medium.com/@xnuinside

 xnuinside@gmail.com

 https://github.com/xnuinside

На каком уровне говорим

We are here >>>

Python Developer level

CPython - Злата Обуховская (Moscow Python)
https://www.youtube.com/watch?v=lSgoYx06L_s&list=PLv_zOGKKxVpi6BSAuySAtX5KyCa50PSCz&index=3

OS

Physical memory level

Дисклеймер

- Говорим про Python 3.7 (преимущественно) и CPython
- В целях соблюдения **NDA** любые намеки на реальный проект удалены, а все совпадения случайны
- Доклад о множестве нюансов, которые прячуться за кажущейся простотой Python, знание подобных нюансов и внимательность позволяет избавить себя от лишней боли и не пропускать на ревью потенциальные источники проблем
- Слайдов много, но я ужимала как могла
- Всё выложу после митапа, все ссылки на код семплы есть в слайдах

Чего в докладе не будет

1. Особенностей при работе с mutable типами, где можно получить граблями в лицо на практике*,* аналогично про lambda, циклы, контексты

2. Ошибки при оценки потребления памяти в алгоритмах, количествах комбинаций и т.д.

3. **Почти** не будет про профайлеры, особенности существующих инструментов, тонкости при работе, как использовать на что смотреть

4. GC

Когда важно думать про память

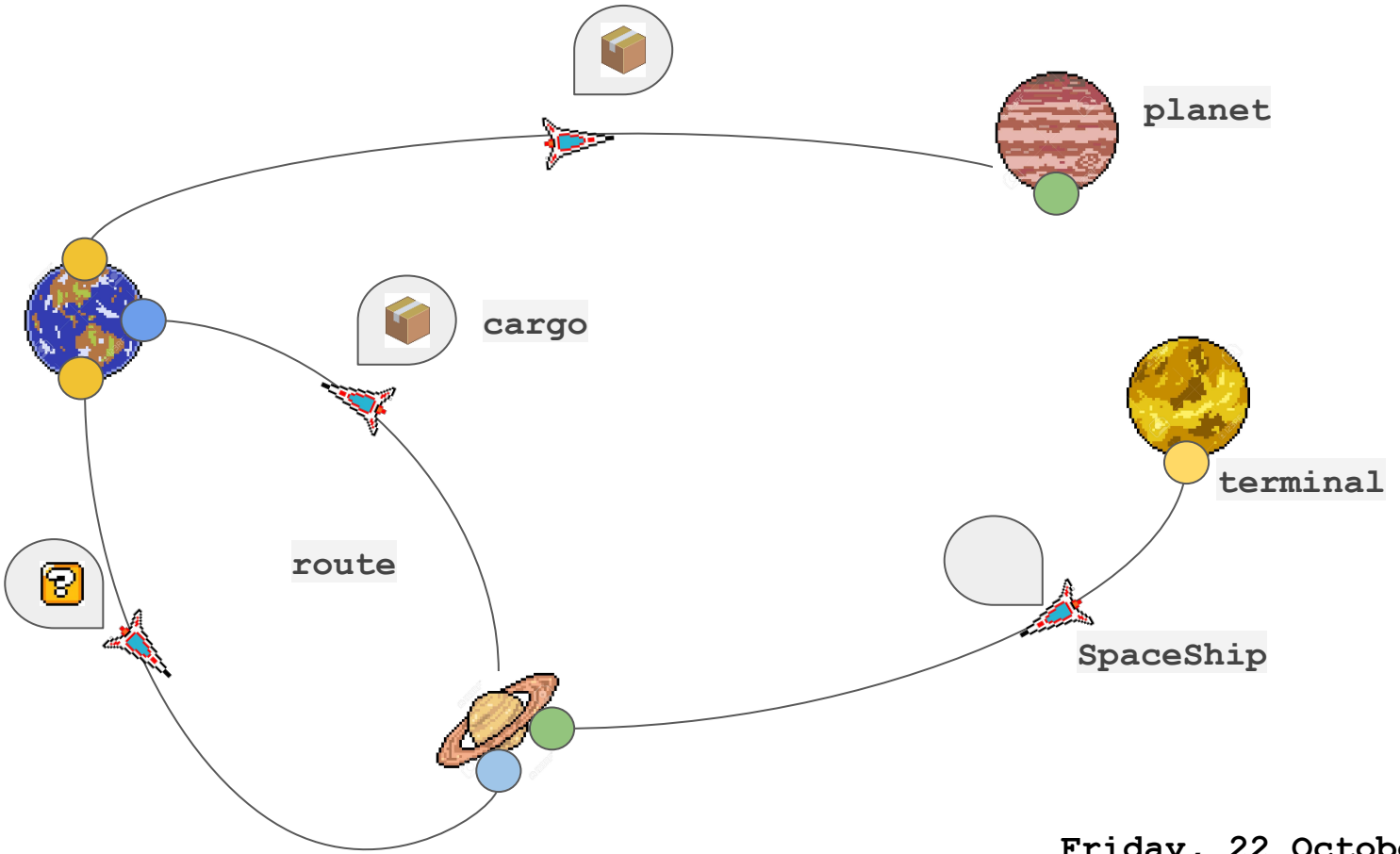
- Калькуляции вероятностей, комбинаций, анализ событий
- Обработка одновременно большого объема массива, связанных данных
- Любые операции, когда вы вынуждены хранить и “жонглировать” объектами в структурах языка и не можете дергать риалтайм из БД

И тд

Проект-вдохновение

Total revenue: 5 billions cosmo $

revenue

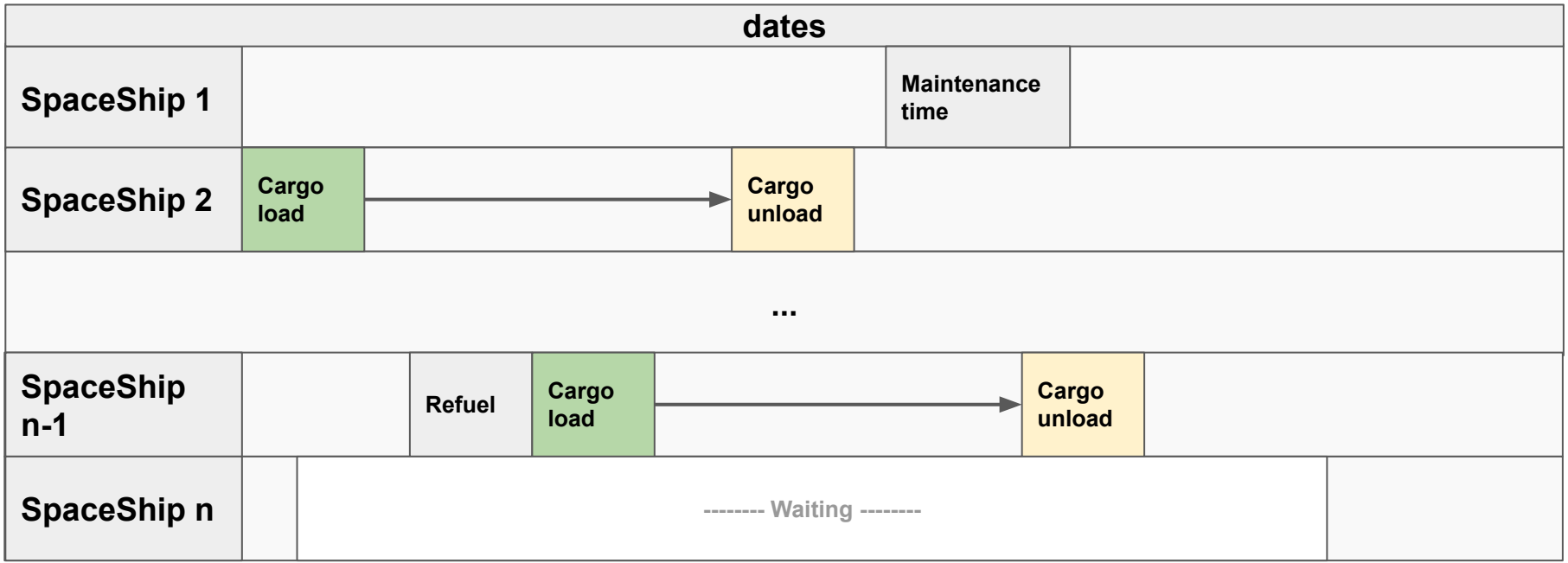


date

Friday, 22 October, 10:30 PM

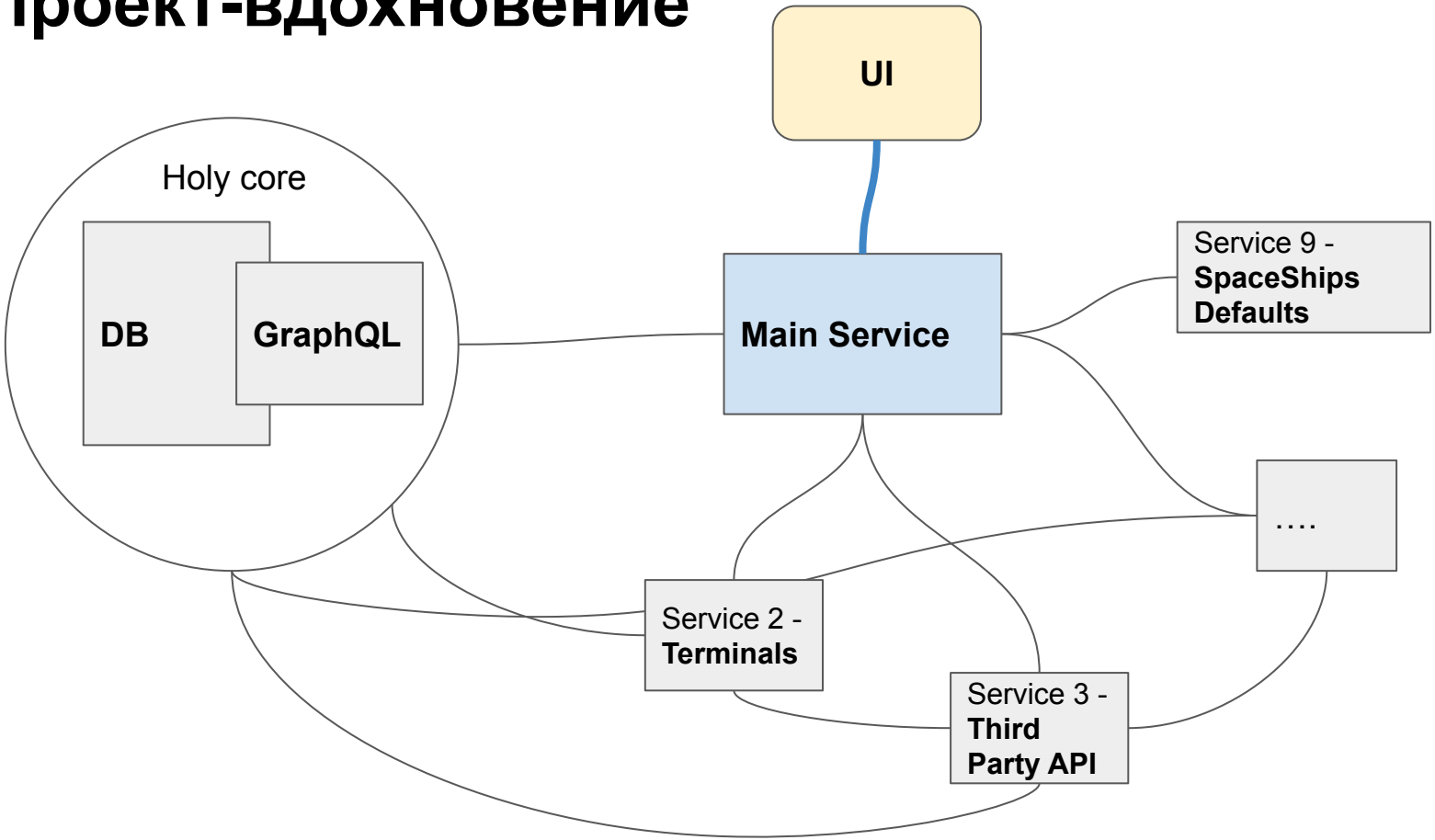
На выходе

Possible risks, cosmo pirates and etc.

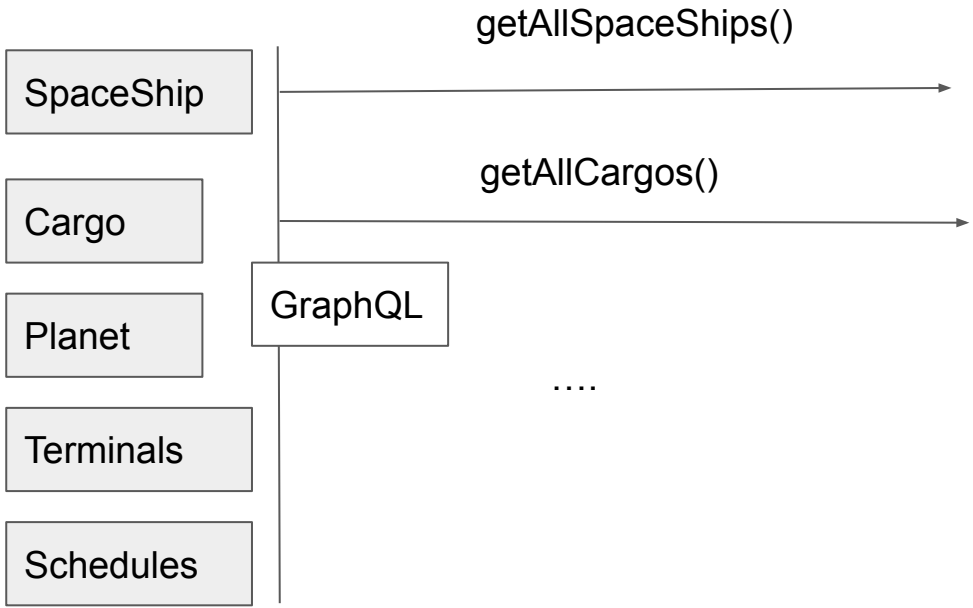


Total revenue:

Проект-вдохновение



Проект-вдохновение




```
# Cargo

{
        ‘productType’: {
                ‘name’: ‘ProductName’
                ‘category’: {
                        ‘id’: ‘GreatProductCategory’
                }
        }

}
```

For example, SpaceShip: ~ 15 fields, most part of them - nested dicts, lists with dicts and etc.

Начнем с семпла


```
[{"id": 1326380, "product": {"name": "Duper", "uid": "elec109ui"},
  "quantity": {"value": 1165, "id": "mt"}, "dates": {"start":
"2200-10-13T00:00:00",
  "end": "2200-11-13T00:00:00"}},

 {"id": 1710124, "product": {"name": "Elec'sOil", "uid": "duper123"},
  "quantity": {"value": 1225, "id": "mt"},
  "dates": {"start": "2200-10-12T00:00:00", "end":
"2200-11-12T00:00:00"}},

 {"id": 1013489, "product": {"name": "Elec'sOil", "uid": "elec109ui"},
  "quantity": {"value": 1905, "id": "mt"}, "dates": {"start":
"2200-10-13T00:00:00",
  "end": "2200-11-12T00:00:00"}} ...]
```


На что смотреть?

1. Процесс создания объектов
2. И только потом на размер

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/data/cargo.json

1 объект или несколько?

C

```
int a = 1;
```


```
a = 2;
```


```
int b = a;
```

`variables`

Python

```
a = 1
```


```
a = 2
```


```
b = a
```

`names`

Картиночки отсюда и рекомендую к прочтению:
http://foobarnbaz.com/2012/07/08/understanding-python-variables/

используем id()

Сколько объектов?

```
a = 12
b = 12
c = 12

print(id(a))
print(id(b))
print(id(c))
```

```
def function_1_with_low_ints(call_number):
    d = 12
    f = 12
    g = 12

    print(id(a), id(b), id(c))


function_1_with_low_ints(1)
function_1_with_low_ints(2)
```

1 и тот же объект в функциях и в глобальном скоупе

для int от -5 до 256

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

А для остальных int?

```
>>> a = 256
>>> b = 256
>>> a is b
True
```

This is not the case if the object referred to is outside the range:

```
>>> a = 257
>>> b = 257
>>> a is b
False
```

Только в REPL!!!

```
>>> a=1234 ; b=1234
>>> a is b
True
>>>
```

```
>>> a = "python super string"
>>> b = "python super string"
>>> a is b
False
>>> a = "python super string" ; b = "python super string"
>>> a is b
True
```

Скрин из:

https://medium.com/@tyastropheus/tricky-python-i-memory-management-for-mutable-immutable-objects-21507d1e5b95

(это не единичный пример)

Так что же с int

```
MacBook-Air:~ xnu$ python3 /Users/xnu/meetup-examples/meetup_examples/realbehavior_example.py
True
True
True
4323700368 4323700368 4323700368 4323700368 4323700368 4323700368
MacBook-Air:~ xnu$ cat /Users/xnu/meetup-examples/meetup_examples/realbehavior_example.py
a = 257
b = 257
c = 257
d = 257
f = 257
g = 257

print(a is g)
print(f is c)
print(b is d)
print(id(a), id(b), id(c), id(d), id(f), id(g))
MacBook-Air:~ xnu$ python2.7 /Users/xnu/meetup-examples/meetup_examples/realbehavior_example.py
True
True
True
(140651125589752, 140651125589752, 140651125589752, 140651125589752, 140651125589752, 140651125589752)
MacBook-Air:~ xnu$
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

1 объект в рамках 1 скоупа

```python
def function_1_with_ints(call_number):

    print(f"Function 1 call number {call_number}")
    a = 2577
    b = 2577
    ....
    print(id(a), id(b), id(c), id(d), id(f), id(g))

function_1_with_ints(1)
function_1_with_ints(2)


def function_2_with_ints(call_number):
    print(f"Function 2 call number {call_number}")
    ....

function_2_with_ints(1)
function_2_with_ints(2)
```

1

1

```
4564591952 4564591952 ...

Function 1 call number 1
4564592144 4564592144 ...
Function 1 call number 2
4564592144 4564592144 ...

Function 2 call number 1
4564592336 4564592336 ...
Function 2 call number 2
4564592336 4564592336 ...
```

1 объект для каждой области инициализации

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

Что со строками?

```
str1 = "name"
str2 = "your"
str3 = "python memory"
str4 = "omg this is amazing python memory"
print(id(str1), id(str2), id(str3), id(str4))

str1_1 = "name"
str2_1 = "your"
str3_1 = "python memory"
str4_1 = "omg this is amazing python memory"
print(id(str1_1), id(str2_1), id(str3_1),
id(str4_1))
```

Тот же id в рамках скоупа, аналогично int

В гитхабе - пример кода с функциями, id будут по 1 штуке для каждого скоупа

```
4389975216 4391070384 4391070000 4391430064
4389975216 4391070384 4391070000 4391430064
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

Немного магии

```
c = 500
d = 500
f = 200 + 300

# in 3.6 f will be different from c and d,
in 3.7 - will be the same all 3 ids
print(id(f), id(c), id(d))
```

```
4554954640 4554954640 4554954640
```

```
# mathematical operations
a = 200
b = 300
c = 500
d = 500
# pay attention to this line
f = a + b

# in 3.6 and in 3.7!! f will be different
from c and d, interning does not  working
print(id(f), id(c), id(d))
```

```
4554953872 4554954640 4554954640
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

Конкатенация строк

```
str_name = "name"
str_3 = "3"
str1_3_concat = str_name + str3

str1_3_c_1 = "name" + str(3)

str1_3_c = "name" + "3"
str_c = "name3"

print(id(str1_3_concat), id(str1_3_c_1),
      id(str1_3_c), id(str_c))
```

```
4447882816 4447682992 4447682736 4447682736
```

"name" + str(3)

```
78          0 LOAD_CONST               1 ('name')
            2 LOAD_GLOBAL              0 (str)
            4 LOAD_CONST               2 (3)
            6 CALL_FUNCTION            1
            8 BINARY_ADD
           10 RETURN_VALUE
```

"name" + "3"

```
80          0 LOAD_CONST               1 ('name3')
            2 RETURN_VALUE
```

"name3"

```
82          0 LOAD_CONST               1 ('name3')
            2 RETURN_VALUE
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

```
test_dict = {'key1': 1, 'key2': 135673,
             'key3': 'python', 'key4': 'very long and strange string',
             'key1_1': 1, 'key2_2': 135673,
             'key3_3': 'python', 'key4_4': 'very long and strange string'}
```

```
1   4504432400 4504432400
135673   4507093328 4507093328
```

```
python   4507705904 4507705904
very long and strange string   4507121184 4507121184
```

```
test_list = ['python', 1, 135673, 1, ['very long and strange string'], 135673,
             'very long and strange string', 'very long and strange string',  'python']
```

```
very long and strange string   4507121184 4507121184
very long and strange string inside nested list   4507121184
```

```
python   4507705904 4507705904
135673   4507093328 4507093328
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/variables_objects_init_example.py

Инициализация типов-контейнеров

list

```
a = [1, 2]
b = [1, 2]
d = [1, 2]
c = [1, 2]
print(id(a), id(b),
id(d), id(c))
```

```
4450065536
4450065696
4450065376
4437969808
```

dict

```
a = {'1': 1}
b = {'1': 1}
d = {'1': 1}
c = {'1': 1}
print(id(a), id(b),
id(d), id(c))
```

```
4331742784
4331742624
4331742544
4331047184
```

tuple

```
a = (1, 2,)
b = (1, 2,)
d = (1, 2,)
c = (1, 2,)
print(id(a), id(b),
id(d), id(c))
```

Python3.6

```
4346801224 4346801288
4346801352 4346801416
```

Python3.7

```
4330278496 4330278496
4330278496 4330278496
```


Размер: чем будем смотреть?

Pympler - asizeof

Последний релиз - Last released: Apr 5, 2019

Pympler is a development tool to measure, monitor and analyze the memory behavior of **Python objects** in a running Python application.

https://github.com/pympler/pympler

Внутри много разных модулей mupy для онлайн мониторинга и поиска утечек, модули для работы с хипом и т.д.

sys.getsizeof - показывает, только память под сам объект, не учитывает nested объекты

Сколько байт

Для Python 3.6

28	int	+4 bytes about every 30 powers of 2
37	bytes	+1 byte per additional byte
49	str	+1-4 per additional character (depending on max width)
48/56 (in py37)/ 56(py2.7)	tuple	+8 per additional item
64	list	+8 for each additional
224	set	5th increases to 736; 21nd, 2272; 85th, 8416; 341, 32992
240	dict	6th increases to 368; 22nd, 1184; 43rd, 2280;

86th, 4704; 171st, 9320

Уточню по системе

x86_64
Darwin Kernel Version 17.7.0: Sun Jun 2 20:31:42 PDT 2019;
root:...-4570.71.46~1/RELEASE_X86_64
Darwin-17.7.0-x86_64-i386-64bit

x86_64

Darwin Kernel Version 18.7.0: Thu Jun 20 18:42:21 PDT 2019;
root:...-4903.270.47~4/RELEASE_X86_64

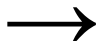
Darwin-18.7.0-x86_64-i386-64bit

JSON

File https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/data/cargo.json

```
File size 1344182 bytes
File size 1.28 mbytes
```

читаем в строку



```
Txt massive 1344232 bytes
Txt massive 1.2819595336914062 mbytes
```

json.load()

```
8317088 bytes
7.93 mbytes
```

orjson.loads()

```
8316688 bytes
7.93 mbytes
```

ujson.load()

```
11724576 bytes
11.18 mbytes
```

eval(str(json_dict))
python dict by Python

```
6238824 bytes
5.95 mbytes
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/json_memory_overuse.py

Давайте найдем причину

```
global_1 = 1
global_2 = 1
global_1345_1 = 1345
global_1345_2 = 1345
global_str_1 = "python"
global_str_2 = "python"
global_long_string_1 = "python is a very funny language"
global_long_string_2 = "python is a very funny language"

with open("data/small_json.json", 'r') as small_json:
  small_json_dict = json.load(small_json)
  in_context_int = 1
  in_context_1345 = 1345
  in_context_str = "python"
  in_context_long_str = "python is a very funny language"
```


```
[{"1": 1, "11": 1, "2": "python", "21": "python",
"long_str": "python is a very funny language",
"long_str1": "python is a very funny language",
"1345": 1345, "13451": 1345},
{"1": 1, "11": 1, "2": "python", "21": "python",
"long_str": "python is a very funny language",
"long_str1": "python is a very funny language",
"1345": 1345, "13451": 1345}]
```


Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/json_memory_overuse.py

Делаем принт

```
print("python", id(global_str_1), id(global_str_2), id(in_context_int)) ...
print('python', id(small_json_dict[0]['2']), id(small_json_dict[0]['21'])) ...
```

global vars and context

1	4530581264	4530581264	4530581264
1345	4533240560	4533240560	4533240560
Python	4533542960	4533542960	4533542960
python is a..	4533269168	4533269168	4533269168

	dict1 from json.load()		dict2 from json.load()	
1	4530581264	4530581264	4530581264	4530581264
1345	4698025840	4698025840	4698025616	4698025616
Python	4698074672	**4698074800**	4698079280	**4698079344**
python is a..	4698070272	**4698070352**	4698070512	**4698070592**

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/json_memory_overuse.py

Вернемся к проекту

```
cargo_example = {
                "id": 12318912,
                "product": {"name": "Elec'sOil",
                            "uid": "elec109ui"},
                 "quantity": {"value": 2380, "id": "mt"},
                 "dates": {"start": "2200-10-14T00:00:00",
                           "end": "2200-11-11T00:00:00"}
                }
```

Мы убедились, что наш дикт не содержит дублирующих int и str, проверили либы

Size of dict **1896**
Size of dict 'product' **488**
Size of dict 'quantity' **448**
Size of dict 'dates' **504**

```
# product
{"name": "Elec'sOil", "uid": "elec109ui"}
```

Size of dict 'product' **488**

Начнем с оптимизации этого куска

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Попробуем другие контейнеры

	Размер bytes	Доступ через [‘key’]	
dict	**488**	**да**	`{"name": "Elec'sOil", "uid": "elec109ui"}`
frozendict	800	**да**	frozendict(`{"name": "Elec'sOil", "uid": "elec109ui"}`)
namedtuple	200	**нет**	`product = namedtuple("ProductNamedTuple", ['name', 'uid'])` `pr_named_tuple = product("Elec'sOil", "elec109ui")`
tuple	200	**нет**	`("Elec'sOil", "elec109ui")`
object	**424**	**да**	**class Product -->**

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Посмотрим поближе на класс

424

```
class Base:
    def __setitem__(self, key, value):
        self.__dict__[key] = value

    def __getitem__(self, key):
        return self.__dict__[key]


class Product(Base):

    def __init__(self, name, uid):
        self.name = name
        self.uid = uid
```

→

328

```
class Base:
    def __setitem__(self, key, value):
        self.__dict__[key] = value

    def __getitem__(self, key):
        return self.__dict__[key]


class ProductSlots(Base):
    __slots__ = ['name', 'uid']

    def __init__(self, name, uid):
        self.name = name
        self.uid = uid
```

```
print("Size of one Product __dict__",
asizeof(one_product.__dict__)) -> Size of one Product __dict__ 360
```

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Можем лучше

328

```
class Base:
    def __setitem__(self, key, value):
        self.__dict__[key] = value

    def __getitem__(self, key):
        return self.__dict__[key]


class ProductSlots(Base):
    __slots__ = ['name', 'uid']

    def __init__(self, name, uid):
        self.name = name
        self.uid = uid
```

→

192

```
class BaseSlots:

    __slots__ = []

    def __setitem__(self, key, value):
        self.__setattr__(key, value)

    def __getitem__(self, key):
        return self.__getattribute__(key)


class ProductSlotsWithBaseSlots(BaseSlots):

    __slots__ = ['name', 'uid']

    def __init__(self, name, uid):
        self.name = name
        self.uid = uid
```

dict 488 → object 192

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Переведем всё в объекты

```
{"id": 12318912,
 "product": {"name": "Elec'sOil",
             "uid": "elec109ui"},
 "quantity": {"value": 2380, "id": "mt"},
 "dates": {"start": "2200-10-14T00:00:00",
           "end": "2200-11-11T00:00:00"}}
```

```
Cargo("123123",
      Dates('2010-03-10', '2010-03-10'),
      ProductSlotsWithBaseSlots("Elec'sOil",
           "elec109ui"),
      OptimizedQuantity(450, "mt")
      )
```

1896 → 688

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Посмотрим на листе с 5 строками

```
[{"id": 12318912, "product": {"name": "Elec'sOil", "uid": "elec109ui"},
  "quantity": {"value": 2380, "id": "mt"},
  "dates": {"start": "2200-10-14T00:00:00", "end":
"2200-11-11T00:00:00"}},
  {"id": 22318912, "product": {"name": "Elec'sOil", "uid": "elec109ui"},
   "quantity": {"value": 380, "id": "mt"},
   "dates": {"start": "2200-11-14T00:00:00",
             "end": "2200-12-11T00:00:00"}}...]
```

6928 → 2840

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Добавим init только одного инстанса

```python
class BaseSlots:

    __slots__ = []

    def __setitem__(self, key, value):
        self.__setattr__(key, value)

    def __getitem__(self, key):
        return self.__getattribute__(key)
```

→

```python
class OptimisedBaseUniq:

    __slots__ = []

    _instances = {}

    def __new__(cls, *args, **kwargs):
        # you can make it better!
        instance_args = str(list(args) + list(kwargs.values()))
        if instance_args in cls._instances:
            return cls._instances[instance_args]
        else:
            obj = super(OptimisedBaseUniq, cls).__new__(cls)
            cls._instances[instance_args] = obj
            return obj

    def __getitem__(self, key):
        return self.__getattribute__(key)
```

1 объект на 1 сет аргументов

2840 → 2648

Пока не понятно..

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/dict_optimize_memory.py

Загрузим файл побольше

```
File size: 1344182 bytes (1.2819 mbytes) cargo.json

Dicts size:      6238824 bytes 5.9498 mbytes
Non-unique:      Size with optimized objects 1918712 bytes 1.8298 mbytes
Unique objects: Size with optimized unique product objects 1535736 bytes
                                                          1.4646 mbytes
```

Но надо очень аккуратно с _instances

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/perfomance_test.py

Так, стоп, а что с perfomance

10000 runs

```python
def find_all_dates_in_multi_cargo_example():
    dates = [event['dates'] for event in multi_cargo_example]
    return dates

def find_product_with_name_dict():
    for i in multi_cargo_example:
        if i['product']['name'] == "Elec'sOil":
            return i

def while_products_all_dates_in_multi_cargo_example():
    while cargo_copy_dict:
        i = cargo_copy_dict.pop()
        if i['product']['name'] == "Elec'sOil":
            return i

def get_id_in_multi_cargo_example():
    for i in multi_cargo_example:
        return i['id']
```

Find all dates and return list

Dict **0.3583587479999999**

Object **1.4361160179999999**

Find first product with name with for

Dict **0.0028282829999999315**

Objects by keys **0.011392252999999908**

Search first elem with product name == Elec'sOil with while and pop

Dict **0.0033425770000001798**

Objects by keys **0.004955137000000054**

Return id of first element

Dict **0.0017288610000000137**

Objects by keys **0.004175465999999961**

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/perfomance_test.py

А если атрибуты

```python
def find_all_dates_in_optimized_access_by_arg():
    dates = [event.dates for event in
optimized_multi_cargo_unique_product]
    return dates

def find_first_product_byid_in_optimized():
    for i in optimized_multi_cargo_unique_product:
        if i.product.name == "Elec'sOil":
            return i

def while_all_dates_in_optimized_with_arg():
    while cargo_copy_optimized:
        i = cargo_copy_optimized.pop()
        if i.product.name == "Elec'sOil":
            return i

def get_id_by_arg_optimized():
    for i in optimized_multi_cargo_unique_product:
        return i.id
```

10000 runs

Find all dates and return list
Dict **0.3583587479999999**
Objects by id **0.2514159760000001**

Find first product with name with **for**
Dict **0.0028282829999999315**
Objects by id **0.0030798729999998997**

Search first elem with product name == Elec'sOil with **while** and pop
Dict **0.0033425770000001798**
Objects by id **0.0011348099999999306**

Return id of first element
Dict **0.0017288610000000137**
Objects by id **0.0015311040000001164**

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/perfomance_test.py

А давайте последний тест

```
def find_product_with_name_dict_list():
    return [i for i in
optimized_multi_cargo_unique_product if
i['product']['name'] == "Elec'sOil"]


def find_first_product_byid_in_optimized_list():
    return [i for i in
optimized_multi_cargo_unique_product
     if i.product.name == "Elec'sOil"]
```

10000 runs

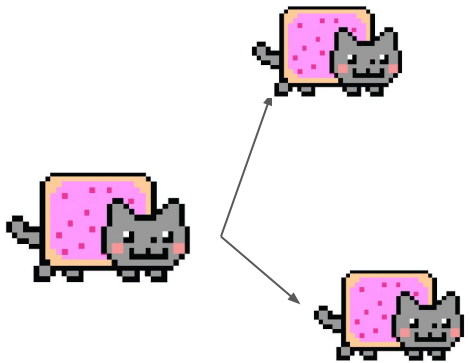
Find all elems with product == name with **for** and return list

Dict 6.373839168999999

Objects by dict keys 4.485599703

Sample:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/perfomance_test.py

Дерево родни для кисы



```
def cat_relative_ids(cat_id, cat_family=[]):
    [cat_family.append(x) for x in range(10)]
    return cat_family


cat_relative_ids(1)
ids = cat_relative_ids(2)
print(len(ids))
```

create_tree → cat_relative_ids

__defaults__

```
def cat_relative_ids(cat_id, cat_family=[]):
    cat_family_from_db = db.query(
        Cat, Cat.relations, cat_id)
    return cat_family
```

Чаще это ‘{}’ в defaults

```
 27           0 LOAD_CLOSURE             0 (cat_family)
              2 BUILD_TUPLE              1
              4 LOAD_CONST               1 (<code object
<listcomp> at 0x10402c780, file
".../meetup_examples/realbehavior_example.py", line 27>)
              6 LOAD_CONST               2
('cat_relative_ids.<locals>.<listcomp>')
              8 MAKE_FUNCTION            8
             10 LOAD_GLOBAL              0 (range)
             12 LOAD_CONST               3 (10)
             14 CALL_FUNCTION            1
             16 GET_ITER
             18 CALL_FUNCTION            1
             20 POP_TOP

 28          22 LOAD_DEREF               0 (cat_family)
             24 RETURN_VALUE
```

Что мы использовали

1. Pympler - asizeof
2. id()
3. dis.dis()

Что есть ещё

1. Зоопарк модулей Pympler-а (included Muppy, SummaryTracker, ClassTracker, Heap и тд) https://github.com/pympler/pympler
2. Heapy из Guppy3 https://github.com/zhuyifei1999/guppy3/ (сейчас посмотрим один из отчетов, что он дает)
3. memory_profiler (очень много НО) https://github.com/pympler/pympler
4. Objgraph (Python Object Graphs https://mg.pov.lt/objgraph/)
5. Tracemalloc (Standard Library)
6. Pysizer (мёртв?) Python 2.5?
7. Если что забыла - докиньте в канал

memory_profiler magic

```
def funct_to_profile():
    a = lambda: 10
    b = lambda: 10
    print(h.heap())
    a()
    b()
    big_list_check = [{'1': 1} for _
in range(100000)]
    print("asizeof ",
asizeof(big_list_check))
    print(h.heap())
    c = big_list_check
```

```
from guppy import hpy

h = hpy()
print(h.heap())
```

Давайте, просто на всяяяяякий случай возьмем другой профайлер

```
Line #    Mem usage    Increment   Line Contents
================================================
    10     31.4 MiB     31.4 MiB   def funct_to_profile():
    11     33.2 MiB      0.0 MiB       a = lambda: 10
    12     33.2 MiB      0.0 MiB       b = lambda: 10
    13     33.2 MiB      1.8 MiB       print(h.heap())
    14     33.2 MiB      0.0 MiB       a()
    15     33.2 MiB      0.0 MiB       b()
    16     60.0 MiB      0.7 MiB       big_list_check = [{'1': 1} for _ in range(100000)]
    17     67.6 MiB      7.5 MiB       print("asizeof ", asizeof(big_list_check))
    18     70.2 MiB      2.6 MiB       print(h.heap())
    19     70.2 MiB      0.0 MiB       c = big_list_check
```

Неару в тот же момент, в том же вызове

До создания листа `big_list_check = [{'1': 1} for _ in range(100000)]`

```
Partition of a set of 111079 objects. Total size = 15374593 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  30718  28  4556540  30   4556540  30 str
     1  29611  27  2483568  16   7040108  46 tuple
     2  15103  14  1134864   7   8174972  53 bytes
     3   7572   7  1094416   7   9269388  60 types.CodeType
     4   7152   6  1029888   7  10299276  67 function
     5   1050   1   913208   6  11212484  73 type
     6   1722   2   736400   5  11948884  78 dict (no owner)
     7    290   0   601224   4  12550108  82 dict of module
     8    949   1   482824   3  13032932  85 dict of type
     9      2   0   262512   2  13295444  86 _io.BufferedWriter
```

После (обратите внимание также на asizeof)

```
asizeof  25624560
Partition of a set of 211093 objects. Total size = 40999745 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0 101722  48 25536528  62  25536528  62 dict (no owner)
     1  30718  15  4556540  11  30093068  73 str
     2  29616  14  2483928   6  32576996  79 tuple
     3  15103   7  1134864   3  33711860  82 bytes
     4   7572   4  1094416   3  34806276  85 types.CodeType
     5   7152   3  1029888   3  35836164  87 function
     6   1050   0   913208   2  36749372  90 type
     7    433   0   913096   2  37662468  92 list
     8    290   0   601224   1  38263692  93 dict of module
     9    949   0   482824   1  38746516  95 dict of type
```

А вот хип до `from pympler.asizeof import asizeof`

```
Partition of a set of 36070 objects. Total size = 4587406 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0  10299  29   911404  20    911404  20 str
     1   8834  24   717136  16   1628540  36 tuple
     2   2331   6   337000   7   1965540  43 types.CodeType
```

Sampler:
https://github.com/xnuinside/meetup_code_samples/blob/master/meetup_examples/profilers_memory_fun.py

Заключение

1. Проверяйте профайлеры профайлерами. У нас тут Python.
2. Не забывайте про то, что ваши переменные это ссылки, а объекты порой могут жить своей жизнью. Убивайте за mutable в дефолтных значениях функций.
3. Пользуйтесь id(), dis.dis() и pymler. Все свои предположения проверяйте, потому что вполне возможно всё совсем не так
4. Не верьте либам, которыми пользуетесь, скорее всего до вас вообще никто не думал про память
5. Вода мокрая, огонь горячий и т.д.

Вопросы?