



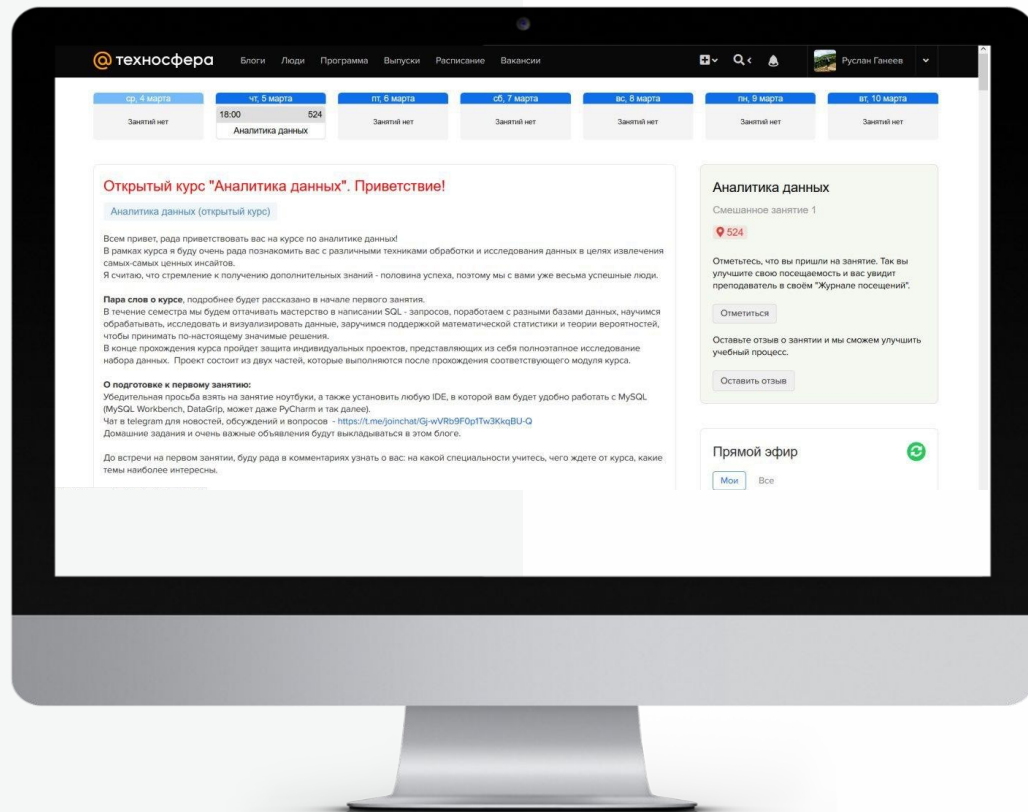
**Backend разработка на Python**

**Лекция 4**

**Работа с памятью, производительность**

Кандауров Геннадий





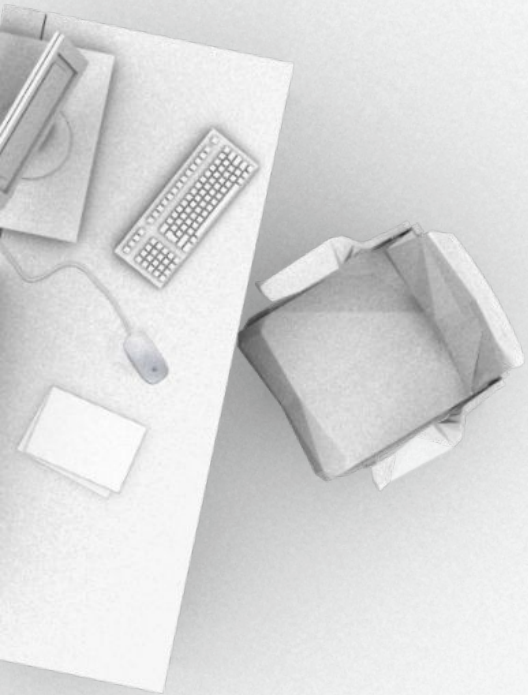
# Напоминание отметиться на портале

+ отзывы после лекции



## **Содержание занятия**

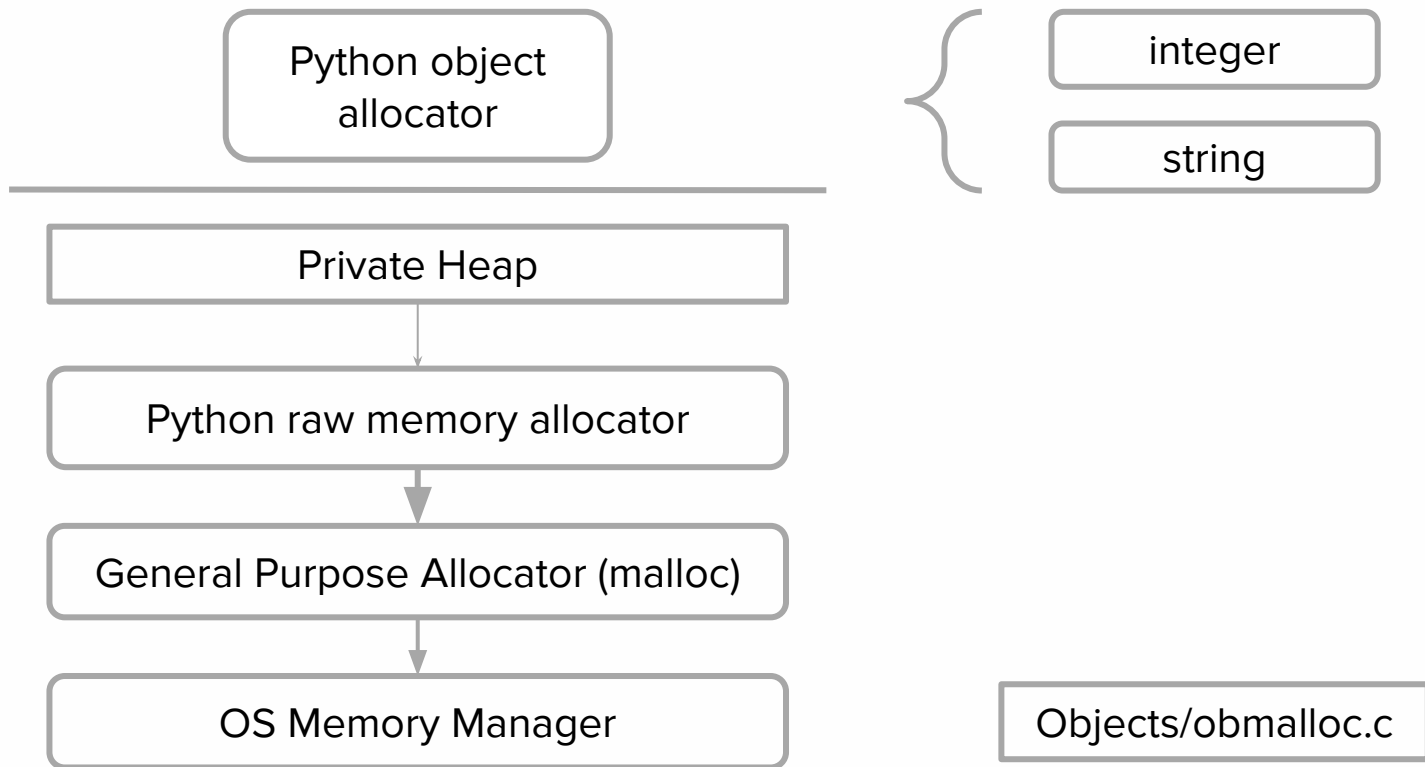
- Организация и выделение памяти
- Счетчик ссылок и сборщик мусора
- Weakref
- Slots
- Профилирование



# Устройство памяти


Выделение, очищение, управление

# Выделение памяти





## PyObject



```
typedef struct _object {  
    _PyObject_HEAD_EXTRA  
    Py_ssize_t ob_refcnt;  
    PyTypeObject *ob_type;  
} PyObject;
```



## Выделение памяти

- Большие объекты ( $> 512$  байт): C allocator;
- Меньшие объекты ( $\leq 512$  байт): арены, пулы, блоки;
  - Блок хранит один объект от 1 до 512 байт;
  - Пул хранит блоки, занимает одну страницу памяти (4Кб);
  - Арена хранит пулы, занимает 256Кб;

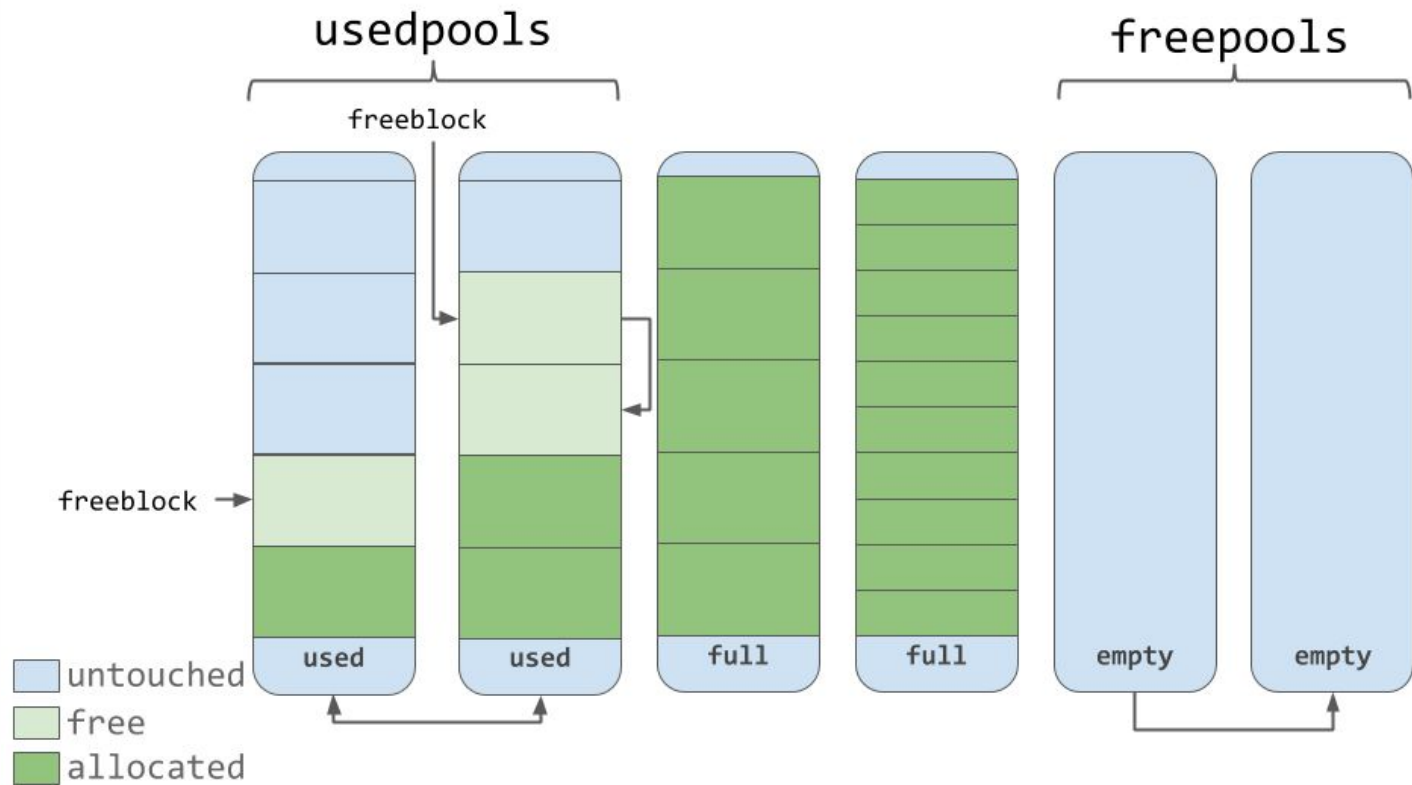
Только арена может освобождать память

## Выделение памяти: блок

* Request in bytes	Size of allocated block	Size class idx
* -----		
* 1-8	8	0
* 9-16	16	1
* 17-24	24	2
* 25-32	32	3
* 33-40	40	4
* 41-48	48	5
* 49-56	56	6
* 57-64	64	7
* 65-72	72	8
* ...	...	...
* 497-504	504	62
* 505-512	512	63
*		
* 0, SMALL_REQUEST_THRESHOLD + 1 and up: routed to the underlying		
* allocator.		

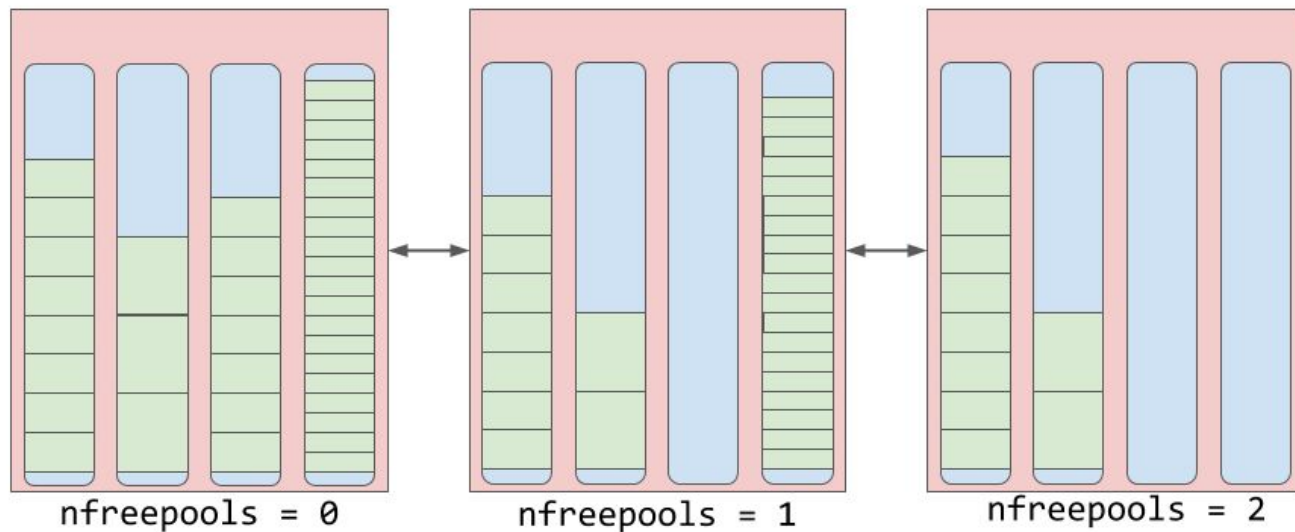


# Выделение памяти: пул



# Выделение памяти: арена

usable\_arenas





## Освобождение памяти



*“The only reliable way to free memory is to terminate the process”*

## Освобождение памяти

- счетчик ссылок, refcounter
- generational garbage collector, модуль gc (опционален)

```
>>> import sys
>>> foo = []
>>> print(sys.getrefcount(foo))
>>> def bar(a):
>>>     print(sys.getrefcount(a))
>>> bar(foo)
>>> print(sys.getrefcount(foo))
```

## Счетчик ссылок (refcount)



Память сразу можно  
очистить



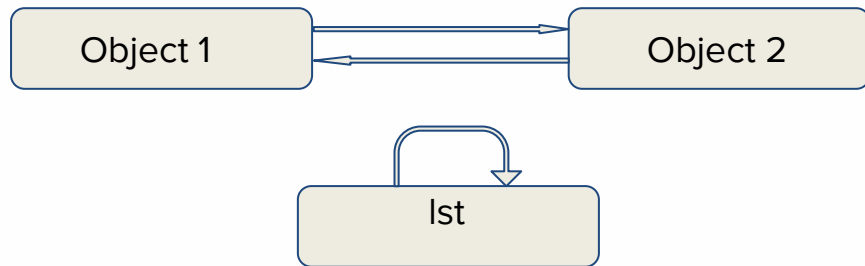
Циклические ссылки



Блокирование потоков




Доп расход CPU и RAM





## Сборка мусора

- 
- + Не нужно думать об очистке памяти
  - + Никаких double free ошибок
  - + Решение проблем утечек памяти

- Дополнительное использование CPU и RAM
- Момент сборки мусора непредсказуем

\_\_\_\_\_





# Сборщик мусора

GC следит только за объектами контейнерами, если они содержат тоже объекты-контейнеры

1. list
2. dict
3. tuple
4. class
5. Etc

<https://docs.python.org/3/library/gc.html>



## Сборщик мусора: отключение

1. Отключение gc

```
gc.disable()
```

```
gc.collect()
```

2. weakref (<https://docs.python.org/3/library/weakref.html>)

```
weakref.ref
```

```
WeakKeyDictionary, WeakValueDictionary, WeakSet,
```

```
WeakMethod;
```

```
finalize;
```

```
list, dict: только для подклассов;
```


```
tuple, int: не поддерживаются.
```

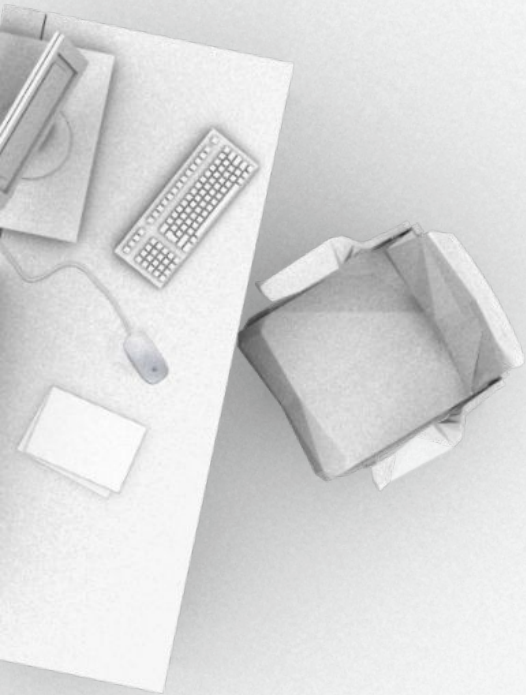
## weakref

```
>>> import weakref
>>> class Object:
...     pass
...
>>> o = Object()
>>> r = weakref.ref(o)
>>> o2 = r()
>>> o is o2
True
>>> del o, o2
>>> print(r())
None
```



## `__slots__`

- 
1. Экономия памяти, побочный эффект - ускорение
  2. Объект не хранит `__dict__`
  3. Допускается объявление `__dict__` в `__slots__` (может свести экономию памяти на нет)
  4. Не наследуется, в подклассе будет `__dict__`
  5. `__weakref__`



# Мониторинг потребления ресурсов



## top/atop

**top** - консольная команда, которая выводит список работающих в системе процессов и информацию о них.

**PID** - идентификатор процесса

**USER** - пользователь, под которым запущен процесс

**VIRT** - объем виртуальной памяти, занимаемой процессом

**RES** - текущее использование RAM

**%CPU** - процент доступного времени процессора

**atop** - продвинутый интерактивный полноэкранный монитор производительности, написанный для Linux.

```
atop -r /var/log/atop/atop_<date> [-b hh:mm]
```



## iotop/iostat

**iotop** - утилита, выводящая данные по использованию жесткого диска.

- `iotop -o` (активные процессы)
- `iotop -o -a` (собрать статистику за время)

**iostat** - утилита, предназначенная для мониторинга использования дисковых разделов.

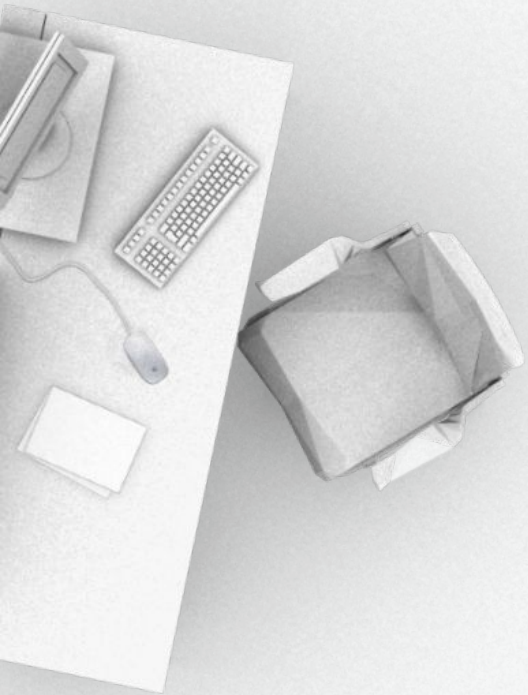
`iostat -d -t -p sda -x`

`-c` вывести отчет по CPU

`-d` вывести отчет по использованию диска

`-t` интервал, за который усредняются значения

`-x` вывести расширенную статистику



# Профилирование

сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т. д.



# Профилирование

Цель:

- найти узкие места в коде

Основные способы:

- CPU
- Память
- Частота/продолжительность вызовов функций

Методы:

- Статистический (сэмплирование)
- Детерминированный (инструментирование)



## Профилирование python

- ***cProfile*** - написанная на C, быстрая реализация профилировщика
- ***profile*** - нативная реализация профилировщика на чистом python, значительно медленнее

```
python -m cProfile -o output.txt ptest.py
```

```
import pstats
```

```
p = pstats.Stats('output.txt')
```

```
p.strip_dirs().sort_stats(-1).print_stats()
```

# Профилирование python

```
import cProfile, pstats, io
```

```
pr = cProfile.Profile()
```

```
pr.enable()
```

```
# ... do something ...
```

```
pr.disable()
```

```
s = io.StringIO()
```

```
sortby = 'cumulative'
```

```
ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
```

```
ps.print_stats()
```

```
print(s.getvalue())
```

# Профилирование python

1567629 function calls (1166637 primitive calls) in 809.730 seconds

Ordered by: cumulative time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.164	0.164	809.738	809.738	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/ioloop.py:568(start)
4961	806.444	0.163	806.444	0.163	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/platform/kqueue.py:66(poll)
9982/8005	0.086	0.000	3.095	0.000	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/stack_context.py:269(wrapped)
5657	0.011	0.000	2.767	0.000	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/ioloop.py:471(_run_callback)
6766/2479	0.083	0.000	1.869	0.001	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:507(run)
2445	0.009	0.000	1.775	0.001	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:567(inner)
2445	0.005	0.000	1.764	0.001	/Users/project/src/.env3/lib/python3.7/site-packages/tornado/gen.py:497(set_result)
430	0.008	0.000	0.902	0.002	/Users/project/src/gekko/net/resolver.py:414(resolve)
75	0.000	0.000	0.669	0.009	/Users/project/src/gekko/handlers2/executor.py:93(callback)
75	0.000	0.000	0.669	0.009	/Users/project/src/gekko/handlers2/executor.py:72(_handler_callback)
48	0.000	0.000	0.669	0.014	/Users/project/src/gekko/handlers2/executor.py:114(_done)
72	0.000	0.000	0.612	0.009	/Users/project/src/gekko/location2.py:266(_call_location_method)
60	0.000	0.000	0.610	0.010	/Users/project/src/gekko/location2.py:91(create_gen_tasks)
63	0.000	0.000	0.609	0.010	/Users/project/src/gekkoapps/gosearch/locations/ajax_web.py:27(get)
9	0.000	0.000	0.576	0.064	/Users/project/src/gekkoapps/common/locations/base.py:104(create_response)
9	0.001	0.000	0.572	0.064	/Users/project/src/gekkoapps/common/locations/base.py:97(render_view)
9	0.000	0.000	0.242	0.027	/Users/project/src/gekkoapps/common/locations/base.py:173(get_data_from_view)
9	0.000	0.000	0.242	0.027	/Users/project/src/gekkoapps/common/views/base.py:136(get_data)
9	0.000	0.000	0.239	0.027	/Users/project/src/gekkoapps/gosearch/v1/web/view/compat.py:14(create_location_data)
9	0.000	0.000	0.238	0.026	/Users/project/src/gekkoapps/gosearch/v1/web/view/produce.py:518(get_data)
9	0.000	0.000	0.220	0.024	/Users/project/src/gekkoapps/common/locations/base.py:183(render_json)
9	0.000	0.000	0.220	0.024	/Users/project/src/gekko/template/helpers.py:148(do_json)
9	0.013	0.001	0.220	0.024	/Users/project/src/.env3/lib/python3.7/site-packages/simplejson/encoder.py:371(encode)
3626	0.030	0.000	0.214	0.000	/Users/project/src/gekko/net/resolver.py:185(resolve)
27	0.000	0.000	0.209	0.008	/Users/project/src/gekkoapps/common/views/serp/v1/creator.py:23(create)

## Профилирование памяти

```
pip install memory_profiler
```

```
# run.py  
from memory_profiler import profile
```

```
@profile  
def some_func():  
    lst1 = []  
    lst2 = "1" * 100000
```

```
python -m memory_profiler run.py
```

## Достойны упоминания: pdb, psutil

```
python3 -m pdb script.py
```

или

```
# script.py
```

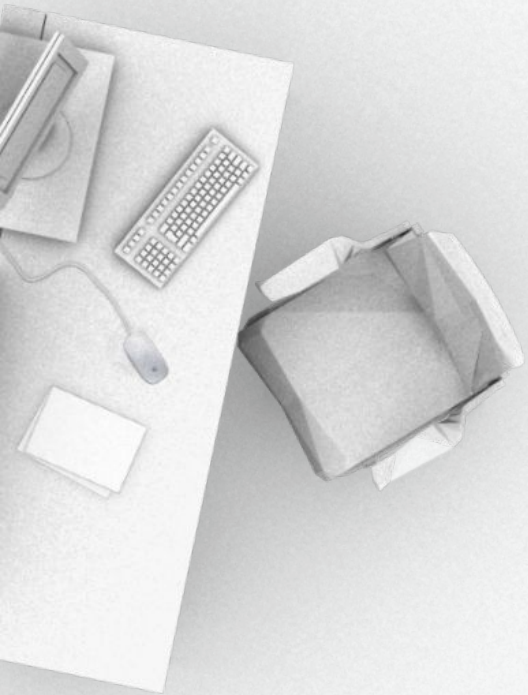
```
def some_func():
```

```
    lst1 = []
```

```
    import pdb; pdb.set_trace()
```

```
    lst2 = "1" * 100000
```


```
python script.py
```



# Логирование



# Логирование

- 
- loggers
  - handlers
  - filters
  - formatters

# Логирование

```
import logging
logging.basicConfig(
    filename='example.log',
    level=logging.DEBUG
)
logging.debug('This message should go to the log file')
logging.info('So should this')
logging.warning('And this, too')
```

```
$ cat example.log
DEBUG:root:This message should go to the log file
INFO:root:So should this
WARNING:root:And this, too
```





## Домашнее задание по лекции #4

ДЗ #4

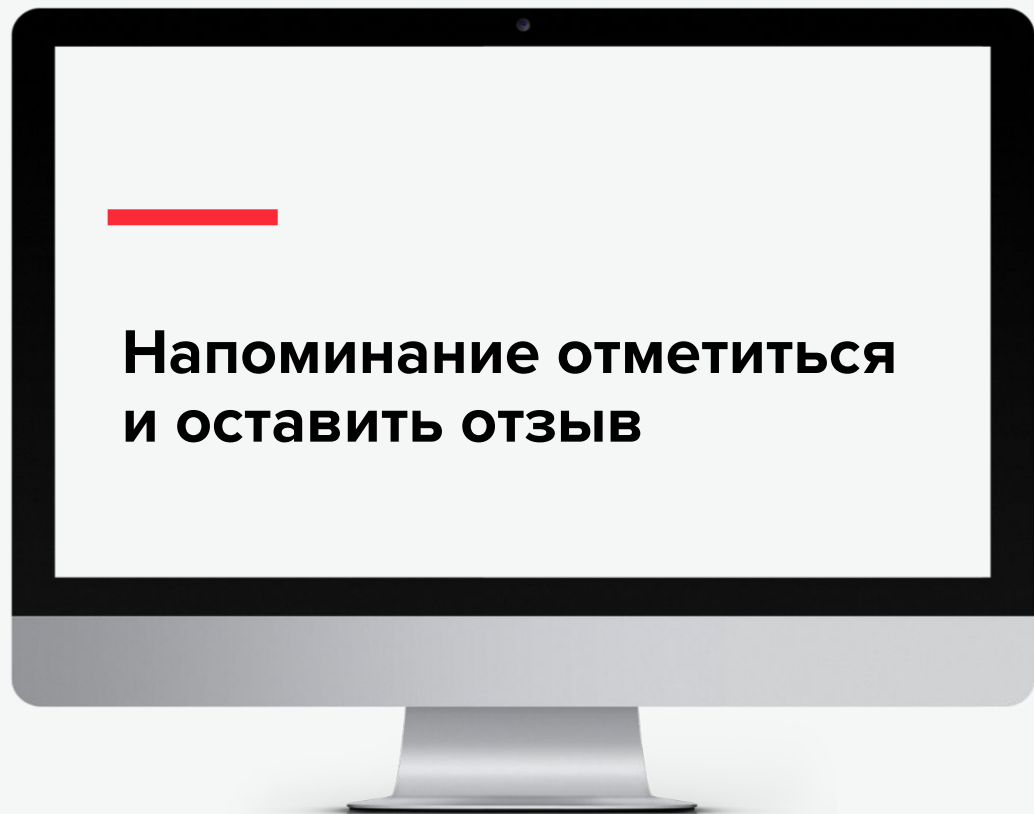
8

30.11.2020

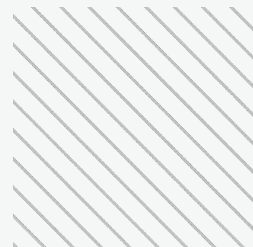
баллов за  
задание

срок сдачи

- Реализовать функцию для вычисления числа возможных вариантов подъема по лестнице из  $n$  ступенек, если на каждом шаге можно подниматься только на 1 или на 2 ступеньки. Исследуйте наивное и оптимизированные решения на потребляемую память и вызовы (профилирование).
- Добавить логирование `debug/info/error` в ORM



**Напоминание отметить  
и оставить отзыв**



**СПАСИБО  
ЗА ВНИМАНИЕ**

