

Разработчик Python

Best Practice по решению прикладных задач и освоению инструментов, применяемых программистом при разработке инфраструктурных решений, веб-приложений, систем контроля качества и аналитических систем

Длительность курса: 146 академических часов

1 Advanced basics

1 **Advanced basics.** **Протоколы.**

разобраться в особенностях представления естественных языков в Python, понять нюансы применения чисел с плавающей точкой, осознать систему протоколов языка, разобраться с концепцией итерирования в Python, получить представление о реализации и применимости итераторов

Домашние задания

1 ДЗ-1: Log Analyzer

Жил-был чудный веб-интерфейса и все у него было хорошо: в него ходили пользователи, что-то там кликали, переходили по ссылкам, получали результат. Но со временем некоторые

его странички стали тупить и долго грузиться. Менеджеры часто жалуются, мол "вот тут список долго грузился" или "интерфейсик тупит, поиск не работает". Но так трудно отделить те случаи, где проблемы на их стороне, а где действительно виноват веб-сервис. В логи интерфейса добавили время запроса (`\$request_time`` в ``nginx`` http://nginx.org/en/docs/http/nginx_http_log_module.html#log_format). Теперь можно распарсить логи и провести первичный анализ, выявив подозрительные URL'ы.

Про логи:

- * семпл лога: ``nginx-access-ui.log-20170630.gz``
- * шаблон названия логов интерфейса соответствует названию сэмпла (ну, только время меняется)
- * так вышло, что логи могут быть и plain и gzip
- * лог ротится раз в день
- * опять же, так вышло, что логи интерфейса лежат в папке с логами других сервисов

Про отчет:

- * `count` - сколько раз встречается URL, абсолютное значение
- * `count_perc` - сколько раз встречается URL, в процентах относительно общего числа запросов
- * `time_sum` - суммарный `\$request_time` для данного URL'a, абсолютное значение
- * `time_perc` - суммарный `\$request_time` для данного URL'a, в процентах относительно общего `\$request_time` всех запросов
- * `time_avg` - средний `\$request_time` для данного URL'a
- * `time_max` - максимальный `\$request_time` для данного URL'a
- * `time_med` - медиана `\$request_time` для данного URL'a

Задание: реализовать анализатор логов
`log_analyzer.py`.

Основная функциональность:

1. Скрипт обрабатывает при запуске последний (со самой свежей датой в имени, не по mtime файла!) лог в `LOG_DIR`, в результате работы должен получиться отчет как в `report-2017.06.30.html` (для корректной работы нужно будет найти и принести себе на диск `jquery.tablesorter.min.js`). То есть скрипт читает лог, парсит нужные поля, считает необходимую статистику по url'ам и рендерит шаблон `report.html` (в шаблоне нужно только подставить `\$table_json`). Ситуация, что логов на обработку нет возможна, это не должно являться ошибкой.
2. Если удачно обработал, то работу не переделывает при повторном запуске. Готовые отчеты лежат в `REPORT_DIR`. В отчет попадает `REPORT_SIZE` URL'ов с наибольшим суммарным временем обработки (`time_sum`).
3. Скрипту должно быть возможно указать считать конфиг из другого файла, передав его путь через `--config`. У пути конфига должно быть дефолтное значение. Если файл не существует или не парсится, нужно выходить с ошибкой.
4. В переменной `config` находится конфиг по умолчанию (и его не надо выносить в файл). В конфиге, считанном из файла, могут быть переопределены переменные дефолтного конфига (некоторые, все или никакие, т.е. файл может быть пустой) и они имеют более высокий приоритет по сравнению с дефолтным конфигом. Таким образом, результирующий конфиг получается слиянием конфига из файла и дефолтного, с приоритетом конфига из файла.
5. Использовать конфиг как глобальную переменную запрещено, т.е. обращаться в своем функционале к нему так, как будто он

глобальный - нельзя. Нужно передавать как аргумент.

6. Использовать сторонние библиотеки запрещено.

Мониторинг:

1. скрипт должен писать логи через библиотеку logging в формате ``[% (asctime)s] %(levelname).1s %(message)s`` с датой в виде ``%Y.%m.%d %H:%M:%S`` (logging.basicConfig позволит настроить это в одну строчку). Допускается только использование уровней ``info``, ``error`` и ``exception``. Путь до логфайла указывается в конфиге, если не указан, лог должен писаться в stdout (параметр filename в logging.basicConfig может принимать значение None как раз для этого).

2. все возможные "неожиданные" ошибки должны попадать в лог вместе с трейсбеком (посмотрите на logging.exception). Имеются в виду ошибки непредусмотренные логикой работы, приводящие к остановке обработки и выходу: баги, нажатие ctrl+C, кончилось место на диске и т.п.

3. должно быть предусмотрено оповещение о том, что большую часть анализируемого лога не удалось распарсить (например, потому что сменился формат логирования). Для этого нужно задаться относительным (в долях/процентах) порогом ошибок парсинга и при его превышении писать в лог, затем выходить.

Тестирование:

1. на скрипт должны быть написаны тесты с использованием библиотеки ``unittest`` (<https://pymotw.com/2/unittest/>). Имя скрипта с тестами должно начинаться со слова ``test``.

Тестируемые кейсы и структура тестов определяется самостоятельно (без фанатизма, в принципе достаточно функциональных тестов).

Цель задания: получить (прокачать) навык написания production-ready кода. То есть адекватного кода, который удобно расширять и поддерживать, протестированного и пригодного для мониторинга. Совпадение всех чисел с приведенным примером отчета целью не является (лишь бы похожи были =)

Критерии успеха: задание __обязательно__, критерием успеха является работающий согласно заданию код, для которого написаны тесты, проверено соответствие pep8, написана минимальная документация с примерами запуска (боевого и тестов), в README, например. Далее успешность определяется code review.

Распространенные проблемы:

- * не стоит делать свои кастомные классы ошибок, это иногда (!) имеет смысл для библиотек, но не для задач подобного рода.

- * ограничьтесь уровнями логирования DEBUG, INFO и ERROR:

<https://dave.cheney.net/2015/11/05/lets-talk-about-logging>

- * не выходите через sys.exit не из main. Это затрудняет тестирование и переиспользование кода.

- * чтобы отрендерить шаблон не надо итерироваться по всем его строкам и искать место замены, можно воспользоваться, например,

https://docs.python.org/2/library/string.html#string.Template.safe_substitute .

- * функцию, которая будет парсить лог желательно сделать генератором.

- * не забывайте про кодировки, когда читаете лог и пишете отчет.

- * из функции, которая будет искать последний лог удобно возвращать namedtuple с указанием пути до него, распаршенной через datetime даты

из имени файла и расширением, например.

* распаршенная дата из имени логфайла

пригодится, чтобы составить путь до отчета, это можно сделать "за один присест", не нужно проходить по всем файлам и что-то искать.

* протестируйте функцию поиска лога, она не должна возвращать .bz2 файлы и т.п. Этого можно добиться правильной регулярой.

* найти самый свежий лог можно за один проход по файлам, без использования glob, сортировки и т.п.

* нужный открыватель лога (open/gzip.open) перед парсингом можно выбрать через тернарный оператор.

* проверка на превышение процента ошибок при парсинге выполняется один раз, в конце чтения файла, а не на каждую строчку/ошибку.

2 **Advanced basics.**
"Граждане первого порядка"

разобраться с особенностями применения ФП в Python,
изучить изучить пространства имен и замыкания,
понять устройство декораторов и способы их использования.

3 **Internals.**
Виртуальная машина

разобраться с устройством виртуальной машины,
осознать процесс исполнения кода,
понять фундаментальные абстракции, которыми оперирует виртуальная машины

Домашние задания

1 ДЗ-2: CPython (опционально)

```
### Opcode
```

```
*Задание*: добавляем опкод, совмещающий в себе несколько других опкодов.
```

```
Взглянем на дизассемблер простой функции, которая вычисляет числа Фибоначчи.
```

```
...
```

```
[root@4e71999b346e cpython]$ python
```

Python 2.7.5 (default, Nov 6 2016, 00:28:07)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.

```
>>> def fib(n): return fib(n - 1) + fib(n - 2) if n > 1  
else n
```

```
...
```

```
>>> import dis
```

```
>>> dis.dis(fib)
```

```
1 0 LOAD_FAST 0 (n)  
3 LOAD_CONST 1 (1)  
6 COMPARE_OP 4 (>)  
9 POP_JUMP_IF_FALSE 40  
12 LOAD_GLOBAL 0 (fib)  
15 LOAD_FAST 0 (n)  
18 LOAD_CONST 1 (1)  
21 BINARY_SUBTRACT  
22 CALL_FUNCTION 1  
25 LOAD_GLOBAL 0 (fib)  
28 LOAD_FAST 0 (n)  
31 LOAD_CONST 2 (2)  
34 BINARY_SUBTRACT  
35 CALL_FUNCTION 1  
38 BINARY_ADD  
39 RETURN_VALUE  
>> 40 LOAD_FAST 0 (n)  
43 RETURN_VALUE  
...
```

LOAD_FAST и LOAD_CONST так часто идут
вместе

```
...
```

```
LOAD_FAST 0
```

```
LOAD_CONST 1
```

```
...
```

или вот

```
...
```

```
LOAD_FAST 0
```

```
LOAD_CONST 2
```

```
...
```

Давайте "склеим" их, сэкономим на размере
байткода, а может даже и по времени
исполнения (стоит проверить). Для этого

давайте сделаем свой opcode!

В итоге получится как-то так:

...

```
[root@4e71999b346e cpython]$ ./python
Python 2.7.13+ (default, Jul 14 2017, 16:25:35)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
```

```
>>> def fib(n): return fib(n - 1) + fib(n - 2) if n > 1
else n
```

...

```
[44740 refs]
```

```
>>> import dis
```

```
[46032 refs]
```

```
>>> dis.dis(fib)
```

```
1 0 LOAD_OTUS 1
```

```
3 COMPARE_OP 4 (>)
```

```
6 POP_JUMP_IF_FALSE 31
```

```
9 LOAD_GLOBAL 0 (fib)
```

```
12 LOAD_OTUS 1
```

```
15 BINARY_SUBTRACT
```

```
16 CALL_FUNCTION 1
```

```
19 LOAD_GLOBAL 0 (fib)
```

```
22 LOAD_OTUS 2
```

```
25 BINARY_SUBTRACT
```

```
26 CALL_FUNCTION 1
```

```
29 BINARY_ADD
```

```
30 RETURN_VALUE
```

```
>> 31 LOAD_FAST 0 (n)
```

```
34 RETURN_VALUE
```

```
[46059 refs]
```

```
>>>
```

...

Подсказка: придется поменять

Include/opcode.h, Lib/opcode.py,

Python/peephole.c, Python/ceval.c,

opcode_targets.h. В peephole.c нужно найти

место, где можно в случае если мы видим

LOAD_FAST с аргументом 0 со следующим за

ним LOAD_CONST, заменить последний на наш

opcode, а пространство до этого забить

NOP'ами.

Until

Задание: while и for недостаточно, давайте добавим until! Для этого нужно воспроизвести самостоятельно вот эту статью

<http://eli.thegreenplace.net/2010/06/30/python-internals-adding-a-new-statement-to-python/>.

Increment/Decrement

Задание: после выполнения других заданий в интерпретаторе не хватает, кажется, только инкремента и декремента (++/--). Делаем по материалам этой статьи

<https://hackernoon.com/modifying-the-python-language-in-7-minutes-b94b0a99ce14>

Ограничения:

* cpython 2.7

* centos 7

* рекомендую docker, см. code sample ниже

С чего начать

Пробовать что-то сделать проще и удобнее в докере, чтобы не сломать ничего на своей тачке. В нижеприведенном скрипте настраивается окружение и запускается сборка интерпретатора (в шапке даны инструкции по запуску докера). Процесс разработки такой: меняете код, запускаете make, проверяете.
...

```
# скачиваем image с 7кой: docker pull centos
```

```
# запускаем контейнер и заходим: docker run -ti --rm -v
```

```
/Users/s.stupnikov/Coding/docker/cpython:/tmp/bin  
centos /bin/bash
```

```
# контейнер при выходе убьется (--rm),  
монтируем к нему мапочку с этим скриптом (-v ...)  
в папку /tmp/bin внутри контейнера
```

```
#!/bin/bash
```

```
set -x
```

```
set -e
```

```
yum clean all
yum install -y\
git\
make\
gcc-c++\
vim\
ssh\

cd /opt
git clone https://github.com/python/cpython.git
cd cpython
git checkout 2.7
./configure --with-pydebug --prefix=/tmp/python
make -j2
...

```

4 **Internals.
Управление
памятью,
"печально
известный" GIL**

разобраться с работой основных типов данных и следствиями такой реализации, понять процесс управления памятью в Python, осознать то как GIL влияет на производительность Python программ.

5 **ООП. Объектная
модель**

осознать устройство объектно модели Python, разобраться с разделением на новые и классические классы, понять тонкости множественного наследования, разобраться с нюансами реализации ООП в Python.

Домашние задания

1 ДЗ-3.0: Scoring API

***Задание*:** реализовать декларативный язык описания и систему валидации запросов к HTTP API сервиса скоринга. Шаблон уже есть в `api.py`, тесты в `test.py`, функционал подсчета сора в `scoring.py`. API необычно тем, что пользователи дергают методы POST запросами. Чтобы получить результат пользователь отправляет в

POST запросе валидный JSON определенного формата на локейшн /method.

Disclaimer: данное API ни в коей мере не является собой best practice реализации подобных вещей и намеренно сделано "странно" в некоторых местах.

Цель задания: применить знания по ООП на практике, получить навык разработки нетривиальных объектно-ориентированных программ. Это даст возможность быстрее и лучше понимать сторонний код (библиотеки или сервисы часто бывают написаны с применением ООП парадигмы или ее элементов), а также допускать меньше ошибок при проектировании сложных систем.

Критерии успеха: задание __обязательно__, критерием успеха является работающий согласно заданию код, для которого написаны тесты, проверено соответствие пер8, написана минимальная документация с примерами запуска (боевого и тестов), в README, например. Далее успешность определяется code review.

Структура запроса

...

```
{"account": "<имя компании партнера>", "login": "<имя пользователя>", "method": "<имя метода>", "token": "<аутентификационный токен>", "arguments": {<словарь с аргументами вызываемого метода>}}
```

...

* account - строка, опционально, может быть пустым

* login - строка, обязательно, может быть пустым

* method - строка, обязательно, может быть пустым

* token - строка, обязательно, может быть пустым

* arguments - словарь (объект в терминах json), обязательно, может быть пустым

Валидация

запрос валиден, если валидны все поля по отдельности

Структура ответа

ОК:

...

```
{"code": <числовой код>, "response": {<ответ вызываемого метода>}}
```

...

Ошибка:

...

```
{"code": <числовой код>, "error": {<сообщение об ошибке>}}
```

...

Аутентификация:

смотри check_auth в шаблоне. В случае если не пройдена, нужно возвращать

```
```{"code": 403, "error": "Forbidden"}```
```

#### ### Методы

##### #### online\_score.

Аргументы

\* phone - строка или число, длиной 11, начинается с 7, опционально, может быть пустым

\* email - строка, в которой есть @, опционально, может быть пустым

\* first\_name - строка, опционально, может быть пустым

\* last\_name - строка, опционально, может быть пустым

\* birthday - дата в формате DD.MM.YYYY, с которой прошло не больше 70 лет, опционально, может быть пустым

\* gender - число 0, 1 или 2, опционально, может быть пустым

## Валидация аргументов

аргументы валидны, если валидны все поля по отдельности и если присутствует хотя бы одна пара phone-email, first name-last name, gender-birthday с непустыми значениями.

## Контекст

в словарь контекста должна прописываться запись "has" - список полей, которые были не пустые для данного запроса

## Ответ

в ответ выдается число, полученное вызовом функции get\_score (см. scoring.py). Но если пользователь админ (см. check\_auth), то нужно всегда отдавать 42.

...

```
{"score": <число>}
```

...

или если запрос пришел от валидного пользователя admin

...

```
{"score": 42}
```

...

или если произошла ошибка валидации

...

```
{"code": 422, "error": "<сообщение о том какое поле(я) невалидно(ы)>"}
```

...

## Пример

...

```
$ curl -X POST -H "Content-Type: application/json" -d '{"account": "horns&hoofs", "login": "h&f", "method": "online_score", "token": "55cc9ce545bcd144300fe9efc28e65d415b923ebb6be1e19d2750a2c03e80dd209a27954dca045e5bb12418e7d89b6d718a9e35af34e14e1d5bcd5a08f21fc95", "arguments": {"phone": "79175002040", "email": "stupnikov@otus.ru", "first_name": "Станислав", "last_name": "Ступников", "birthday": "01.01.1990", "gender": 1}}' http://127.0.0.1:8080/method/
```

```
...
...
{"code": 200, "response": {"score": 5.0}}
...
```

#### #### clients\_interests.

##### Аргументы

\* client\_ids - массив чисел, обязательно, не пустое

\* date - дата в формате DD.MM.YYYY, опционально, может быть пустым

##### Валидация аргументов

аргументы валидны, если валидны все поля по отдельности.

##### Контекст

в словарь контекста должна прописываться запись "nclients" - количество id'шников, переданных в запрос

##### Ответ

в ответ выдается словарь `<id клиента>:<список интересов>`. Список генерировать вызовом функции `get_interests` (см. `scoring.py`).

```
...
{"client_id1": ["interest1", "interest2" ...], "client2":
[...] ...}
...
```

или если произошла ошибка валидации

```
...
{"code": 422, "error": "<сообщение о том какое
поле(я) невалидно(ы)>"}
...
```

##### Пример

```
...
$ curl -X POST -H "Content-Type: application/json" -
d '{"account": "horns&hoofs", "login": "admin",
"method": "clients_interests", "token":
"d3573aff1555cd67dccf21b95fe8c4dc8732f33fd4e3
2461b7fe6a71d83c947688515e36774c00fb630b039
```

```
fe2223c991f045f13f24091386050205c324687a0",
"arguments": {"client_ids": [1,2,3,4], "date":
"20.07.2017"}}' http://127.0.0.1:8080/method/
...
...
{"code": 200, "response": {"1": ["books", "hi-tech"],
"2": ["pets", "tv"], "3": ["travel", "music"], "4":
["cinema", "geek"]}}
```

#### #### Мониторинг

1. скрипт должен писать логи через библиотеку logging в формате ``[% (asctime)s] %(levelname).1s %(message)s`` с датой в виде ``%Y.%m.%d %H:%M:%S``. Допускается только использование уровней ``info``, ``error`` и ``exception``. Путь до логфайла указывается в конфиге, если не указан, лог должен писаться в stdout

#### #### Тестирование

1. Тестировать приложение мы будем после следующего занятия. Но уже сейчас предлагается писать код, что называется, `with tests in mind`.

## 6 ООР. Белая и черная "магия"

---

понять дескрипторы и их протокол,  
разобраться с использованием "магических методов",  
понять область применимости абстрактных базовых классов,  
осознать особенности эксплуатации метаклассов.

---

- 7 **Testing. Дизайн тестов** осознать необходимость тестирования и его место в жизненном цикле ПО, разобраться с конструированием кейсов тестирования, понять различия между видами тестирования.

Домашние задания

1 ДЗ-3.1: API Testing

Дописываем тесты  
API

---

- 8 **Testing. Пирамида тестирования** разобраться с устройством пирамиды тестирования, понять область применения инструментов тестирования (моков, фикстур и т.д.), разобраться с видами автоматизации тестирования.
- 

- 9 **Automatization. Сетевое взаимодействие** понять принципы сетевого взаимодействия через сокеты, разобраться с особенностями сетевых протоколов, усвоить нюансы написания программ общающихся по сети.

Домашние задания

1 ДЗ-4: Web Server

Создаем свой сервер на "ванильном" Python, частично реализующий протокол HTTP (будет корректно отдавать страницу wikipedia).  
Проводим нагрузочное тестирование.

---



10 **Automatization.**  
**Общение с БД**  
**и демонизация**

осознать нюансы общения с серверной БД и  
основные паттерны,  
разобраться с процессом демонизации программ,  
разобраться с дистрибуцией Python программ.

## 1 Dynamic Web

- понять принципы функционирования динамического веба
- разобрать с WSGI и его особенностями
- получить представление о различных WSGI контейнерах

Домашние задания

- 1 ДЗ-5: uWSGI Daemon/ Django Tutorial (опционально)

Пишем стандартного "промышленного" демона, который будет отвечать по HTTP, ходить в базу, писать логи и собираться в пакет. Те, кто не знаком с Django, проходят tutorial.

---

## 2 Django. Intro

- разобраться с классической структурой веб-сервисов
  - усвоить лучшие практики развертывания Django проектов
  - понять как конфигурируются и эксплуатируются Django проекты
- 

## 3 Django. ORM и "зло"

- получить представление о лучших практиках использования моделей
- понять, как работает ORM и из чего он состоит
- разобраться написанием запросов через ORM
- научиться анализировать запросы

Домашние задания

- 1 ДЗ-6.0: Django project

Создание web-приложения аналога Stack Overflow. Определяемся со структурой и схемой данных.

---

- 4 **Database layer**
- понять как навигироваться в комплексном пространстве мира распределенных систем и баз данных
  - познакомиться с основными понятиями, связанными с эксплуатацией и оптимизацией хранилища данных
- 
- 5 **Django. Views**
- разобраться с устройством логики представления в Django
  - понять, каким образом выбирать подход к описанию этой логики
- Домашние задания
- 1 ДЗ-6.1: Django project
- Начинаем рисовать красивые странички
- 
- 6 **Django. Forms**
- разобраться с логикой обработки форм и принципом их функционирования
  - понять, как устроен template engine и как это влияет на его производительность
- 
- 7 **REST API. Know-how**
- понять, что такое REST
  - разобраться с лучшими практиками реализации этого подхода
- Домашние задания
- 1 ДЗ-6.2: Django REST API (опционально)
- Добавляем к разрабатываемому приложению API.
- 
- 8 **Web performance**
- понять пути масштабирования веб-проектов
  - разобраться с производительностью фронтенда

# 3 Data engineering

- 1 **NumPy. Часть 1** IPython, базовое использование, интроспекция, поиск, история, макросы, магические методы, взаимодействие с ОС, разработка. NumPy. ndarray, индексирование, маски, векторизация. Universal functions.

Домашние задания

- 1 ДЗ-7: LogRegression

Дописываем логистический регрессор, используем его для классификации отзывов о еде из Amazon.

---

- 2 **NumPy. Часть 2** Reshaping, broadcasting, structured и record массивы, хранение и загрузка данных. Внутренности ndarray. Меммар, HDF5.
- 

- 3 **Pandas** Pandas. Series. Dataframe, иерархические индексы, missing data, агрегация.

Домашние задания

- 1 ДЗ-8: Open Data Analysis (опционально)

Выбираем один из открытых dataset'ов и анализируем его с помощью pandas в IPython notebook.

---

- 4 **Matplotlib** Построение графиков, гистограммы, subplots, аннотации, стили. Data visualization, принципы правильных визуализаций.

## 1 Concurrency. Часть 1

Concurrency, parallelism. IO/CPU bound задачи.  
Multithreading, sharing, communication.

Домашние задания

### 1 ДЗ-9: MemcLoad

Реализуем конкурентную заливку данных в memcache'ы

---

## 2 Concurrency. Часть 2

Multiprocessing, IPC, shared memory, Manager.  
Distributed computing

---

## 3 C extensions

Написание расширений на C, C API.

Домашние задания

### 1 ДЗ-10: Protobuf (de)serializer (опционально)

Пишем свое расширение, которое будет писать файлы с protobuf сериализованным содержимым. Понадобятся знания C.

---

## 4 ffi. Cython. PyPy

## 5 Async. Часть 1

Generators, coroutines, yield from. Event loop. Future.

Домашние задания

### 1 ДЗ-11: YCrawler

Пишем асинхронный краулер для новостного сайта [news.ycombinator.com](http://news.ycombinator.com)

---

6	<b>Async. Часть 2</b>	async/await. Error handling	
<hr/>			
7	<b>Golang. Часть 1</b>	Производительность Python. Golang. Toolchain, структура проекта, менеджмент зависимостей, тур по языку.	
		Домашние задания	
		1 ДЗ-12: MemcLoad v2	
		Создаем простого демона на Go, проводим сравнение с аналогичным на Python.	
<hr/>			
8	<b>Golang. Часть 2</b>	Внутренности: горутины, сборщик мусора, оптимизации.	
<hr/>			
9	<b>Profiling</b>	Особенности архитектуры, характеристики железа. Антипаттерны профилирования. Методология. cProfile, line_profiler, memory_profiler. Инструменты Linux, perf.	
<hr/>			
10	<b>Python 3</b>	Обзор изменений, новые фишки. Миграция проектов с 2 на 3 версию.	

# 5 Проектная работа

## 1 Вводное занятие

Домашние задания

1 Проект

---

## 2 Консультация

---

## 3 Защита