

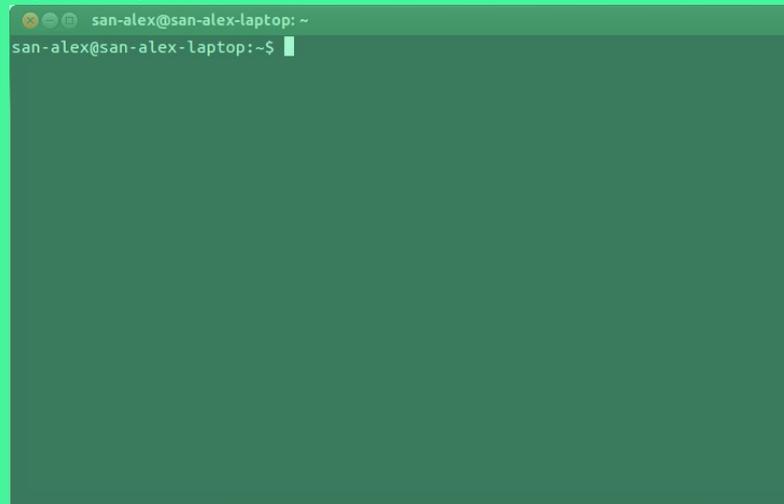
# Околопитона. Инструменты разработки

Александр Стёпкин

# Окружение

# Окружение

- virtualenv (virtualenvwrapper)
- pyenv



# Среда разработки

## «Кому и Vim — IDE» (Jason Statham)



# Код

## На что должен быть похож код за который не стыдно?

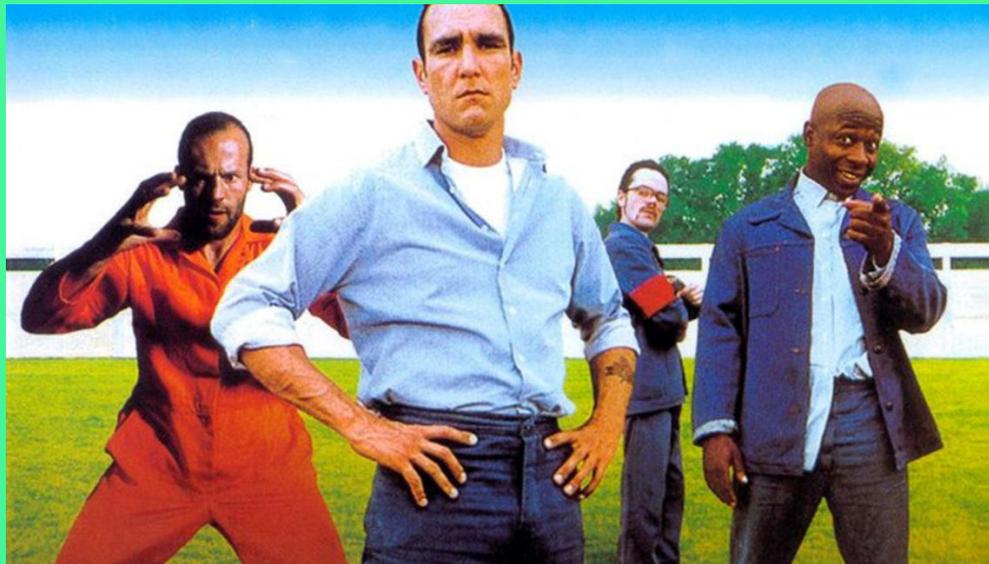
- Должен хорошо читаться (реп-Х)
- Должен писаться в хорошем стиле (различные хорошие практики)
- Должен быть оптимален



# Команда

## Кто делает нас сильнее

- Ревью
- Батлы
- Холивары



Все герои истории вымышлены. Основано на реальных событиях.

Предположим, у нас есть функция, 3 параметра которой имеют значение по умолчанию числовые значения или флаги, 2 параметра - это предопределённые массивы и один словарь. Как передать параметры и их инициализировать?

Реальный кейс:

```
def очень_старый_и_нужный_всем_apі_метод(  
    сначала_думали_что_такое_имя_параметра_гуд=None,  
    два_года_назад_уместным_стало_так=None,  
    год_назад_поменялась_нагрузка_на_параметр=None,  
    полгода_назад=None,  
    сейчас_он_называется_так=None,  
    ...  
):  
    ...
```

Вариант 1 в корне неверный и приводит к ошибкам исполнения. Кто знает почему?

```
def func(  
    param0, param1=112, param2=911, param3=True,  
    para_allow=[1,4,6], para_denied=[2,3,5],  
    para_kek=dict(a=1, b=2)  
):  
    ...
```

Вариант 1 в корне неверный и приводит к ошибкам исполнения. *Кто знает почему?*

```
def func(  
    param0, param1=112, param2=911, param3=True,  
    para_allow=[1,4,6], para_denied=[2,3,5],  
    para_kek=dict(a=1, b=2)  
):  
    ...
```

*спойлер: `__defaults__`*

Вариант 2.

```
def func(
    para0, para1=112, para2=911, para3=True,
    para_allow=None, para_denied=None,
    para_kek=None
):
    para_allow = para_allow or [1,4,6]
    para_denied = para_denied or [2,3,5]
    para_kek = para_kek or dict(a=1, b=2)
    ...
```

## Вариант 3.

```
def func(para0, **kwargs):  
    para1 = kwargs.get('para1', 112)  
    para2 = kwargs.get('para2', 911)  
    para3 = kwargs.get('para3', True)  
    para_allow = kwargs.get('para_allow', [1,4,6])  
    para_denied = kwargs.get('para_denied', [2,3,5])  
    para_kek = kwargs.get('para_kek', dict(a=1, b=2))  
    ...
```

## Доводы против:

- опыт интеграций с библиотеками, где передай в kwargs параметры, но какие неизвестно
- оборачивание в kwargs не упрощает, а скрывает переменные, docstring читать не каждый хочет
- современный IDE подсвечивает требуемую переменную

## Доводы за:

- меняются требования к методу, переименовываются параметры после осознания, а старые клиенты не должны начать испытывать трудности
- стандартизация описательной части

# Инструменты

## Профилирование

**Профилирование** — сбор характеристик работы программы с целью их дальнейшей оптимизации.

**Оптимизация** — это модификация системы для улучшения её эффективности.

Из пушки по воробьям: **cProfile**

Рогатка: **timeit**

```
import time
...
start = time.time()
value = calc_value(num)
print("Time: %.03f s" % (time.time() - start))
```

```
#=====
```

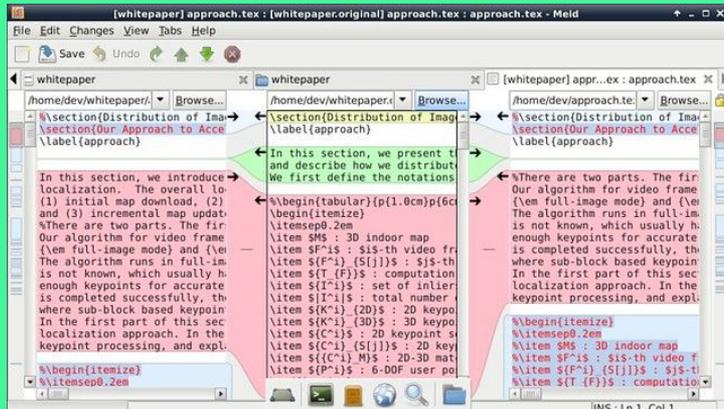
```
import timeit
...
timeit.timeit("from __main__ import calc_value, num; calc_value(num)", number=10000)
```

## Разрешение инцидентов



## Хорошие парни

- Guake
- Meld
- Makefile (make kek)
- Gitk (--follow)



```
1 #whatstodo:
2   echo "то я закончил?"
3   @echo -n "projecting files to git log --pretty=format:%C(yellow)%C(white)%C(green) --after=yesterday --author=stepkin --all
4
5 common:
6   echo "не забудь активировать окружение!"
7   echo "для того, чтобы проверить наличие makefile на сервере --cache-clear, --no-networks, установите python-cache-wopep pip"
8
9 test: common
10  py.test --cache-clear --maxfail=10000 --duration=0 ./target
11
12 pep8: common
13  py.test --pep8 --cache-clear --maxfail=10 ./target
14
15 pylint: common
16  pylint ./target
17
18 lint: common pep8 pylint
19
20 #scripting test
21 RESULT=$(imgo-find-conflicts skyrim_whoite)
22 echo $RESULT
23 [ $RESULT = "no conflicts detected." ]
24
25 clear-ppc:
26   find ./name -type f -exec rm -f {} \;
27   find ./name -type d -exec rm -rf {} \;
28
29 api-test: common
30  py.test --cache-clear --maxfail=10 --pep8 -v ./api/testsrc/target
31
32 api-test-safe: common
33  SAFE=1 py.test --cache-clear --maxfail=10 ./api/testsrc/target
34
35 check:
36   @echo "-----TODOS-----"
37   find ./type f -name "*.py" -exec grep -in "TODOS" {} +
38   @echo "-----CODE-----"
39   find ./type f -name "*.py" -exec grep -in "pdb.set_trace" {} +
40   find ./type f -name "*.py" -exec grep -in "os.system" {} +
41   find ./type f -name "*.py" -exec grep -in "os.system" {} +
42   find ./type f -name "*.py" -exec grep -in "os.system" {} +
43   @echo "-----"
44   # проверить наличие makefile запускаем make с ключом -l, либо парем командой сканера --
45
46
47 startide:
48   docker-compose -f docker-compose.yml up --detach
49
50 #final VERSION
51 VERSION=$(shell git checkout --quiet $(BRANCH) && qd-q-parsechange | shov-field Version && git checkout --quiet --)
52
53 sky-build-pkg:
54   BRANCH=master $(MAKE) custom-build-pkg
55
56 dev-build-pkg:
57   BRANCH=development $(MAKE) custom-build-pkg
58
59 SALT=$(shell date +%Y%M)
60 custom-build-pkg:
61   # в эту строку совершенно любой ветки, можно запустить make:
62   # @branch=$(shell git checkout --quiet $(BRANCH) && qd-q-parsechange | shov-field Version && git checkout --quiet --)
63   @echo "Building for version=$(VERSION) salt=$(SALT) branch=$(BRANCH) make build-custom"
```

## Вместо заключения. Хороший погромист



## Материалы



**Спасибо!**