

NOSQL ВНУТРИ SQL

приземленные вопросы
практического применения



Профессиональная конференция
разработчиков высоконагруженных
систем



Дмитрий Долгов, Mindojо



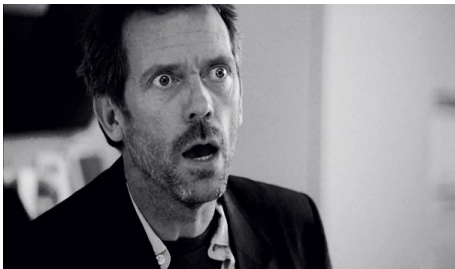
github.com/erthalion

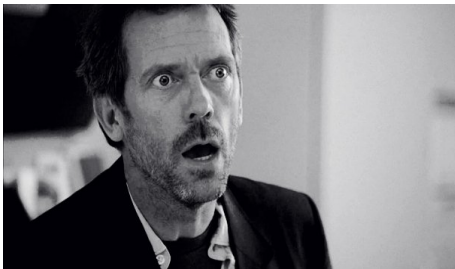


[@erthalion](https://twitter.com/erthalion)



[9erthalion6 at gmail dot com](mailto:9erthalion6@gmail.com)






Почему это важно, понимать функциональный подход к программированию?



Это позволяет по-новому взглянуть на ваш код, сделать его более модульным и надежным











- FP - это парадигма, не привязанная к языку
- Python - мультипарадигменный язык, позволяющий писать функционально
- Это позволяет увидеть преимущества FP в вашем коде уже завтра




→ Разделение данных и логики 



- Разделение данных и логики 
- Модульность, легкое написание тестов 




- Разделение данных и логики 
- Модульность, легкое написание тестов 
- Распараллеливание кода 




- Разделение данных и логики 
- Модульность, легкое написание тестов 
- Распараллеливание кода 

→ Сложно 

- Разделение данных и логики 
- Модульность, легкое написание тестов 
- Распараллеливание кода 

- Сложно 
- Страшно 

- Разделение данных и логики 
- Модульность, легкое написание тестов 
- Распараллеливание кода 

- Сложно 
- Страшно 
- Разработчики 

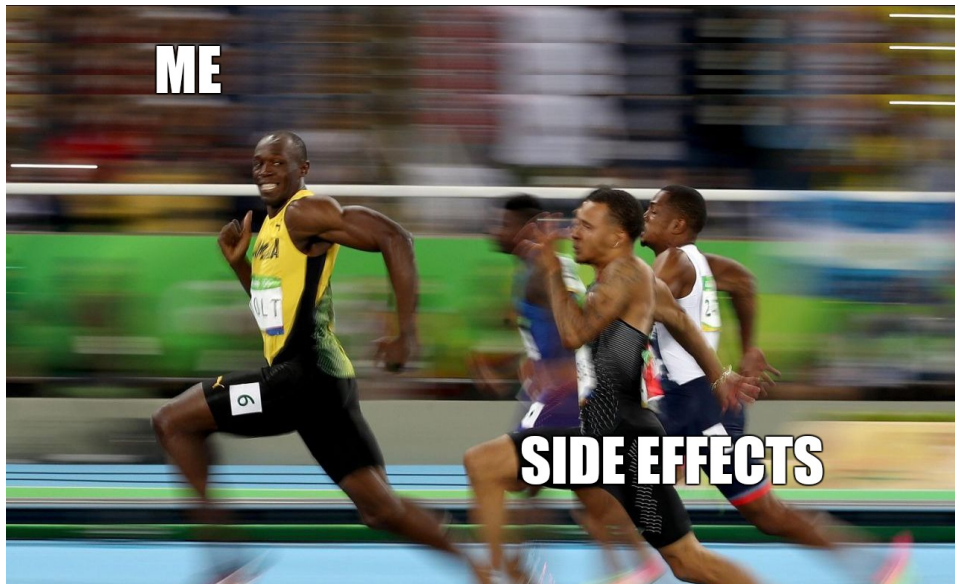
- postgres
- pandoc
- elm-compiler
- purescript
- aura (arch linux package manager)

КРАТКОЕ ВВЕДЕНИЕ В FR

- Неизменяемость данных (immutability)
- Чистые функции и side effects
- Функции высшего порядка
- Монады (?)
- Abstract Data Type (ADT)

**Data, data
never changes**









ВОЗМОЖНОСТИ PYTHON ДЛЯ FP

→ Неизменяемые типы данных:

`string`

`tuple/namedtuple`

`frozenset`

- Неизменяемые типы данных:
 - string
 - tuple/namedtuple
 - frozenset
- Функции высшего порядка

- Неизменяемые типы данных:
 - string
 - tuple/namedtuple
 - frozenset
- Функции высшего порядка
- List comprehension

- Неизменяемые типы данных:
 - string
 - tuple/namedtuple
 - frozenset
- Функции высшего порядка
- List comprehension
- Generators

- Неизменяемые типы данных:
 - string
 - tuple/namedtuple
 - frozenset
- Функции высшего порядка
- List comprehension
- Generators
- itertools

→ Неизменяемые типы данных:

string

tuple/namedtuple

frozenset

→ Функции высшего порядка

→ List comprehension

→ Generators

→ itertools

→ functools

→ Хвостовая рекурсия ✗

- Хвостовая рекурсия ✗
- Чистые функции ✗

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching ✗

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍

- Хвостовая рекурсия ✖
- Чистые функции ✖
- Pattern matching 🔍
- Automatic currying ✖

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍
- Automatic currying 🔍

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍
- Automatic currying 🔍
- Монады ✗

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍
- Automatic currying 🔍
- Монады 🔍

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍
- Automatic currying 🔍
- Монады 🔍
- ADT ✗

- Хвостовая рекурсия ✗
- Чистые функции ✗
- Pattern matching 🔍
- Automatic currying 🔍
- Монады 🔍
- ADT 🔍

- Использование «чистого» Python
- Собственные надстройки
- Сторонние библиотеки

ПРИМЕРЫ (PY2)


```
from collections import namedtuple
```

```
Record = namedtuple("Record", "id name value")  
r = Record(1, "first record", "record value")  
r.name = "second record"      # error
```

```
fset = frozenset([1, 2, 1, 3])  
fset.add(1)      # no such function
```

```
# list comprehension in python  
[v.attr for v in source if condition(v)]
```

```
# function chain in python  
list(reversed(list(islice(count(), 5))))
```

```
# slightly modified version in python  
fchain(list, reversed, list, islice, (count(), 5))
```

```
-- list comprehension in haskell  
[getAttr v | v ← source, condition v]
```

```
-- function chain in haskell  
reverse . take 5 $ [0..]
```

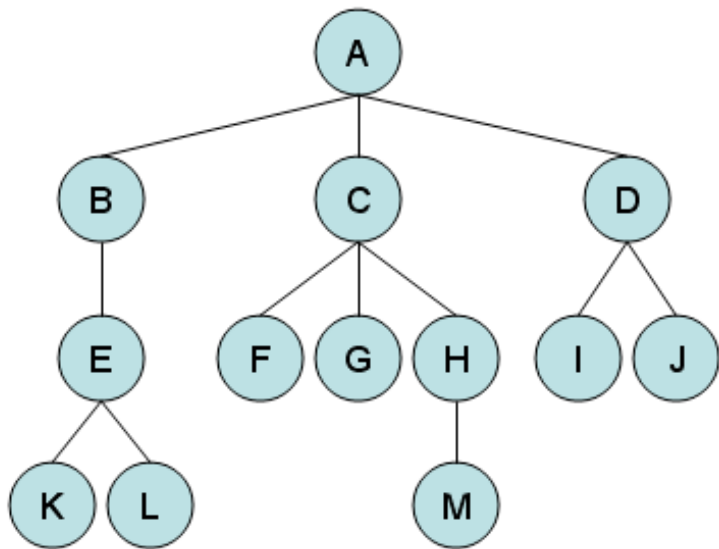
```
from itertools import cycle, ifilter

colors = cycle(["red", "green", "blue", "black"])
data = (
    {"id": i, "color": colors.next()}
    for i in range(10)
)
next(ifilter(
    lambda x: x["color"] == "black", data), None)
```

```
from maybe import Nothing, Just
```

```
def test_function(a, b):  
    """ a & b may be None  
    """  
  
    a2 = a * a  
    b2 = a2 * b  
  
    return (a2, b2)
```

```
test_function(1, 2)           # ok  
test_function(None, 2)       # exception  
test_function(Nothing, 2)    # ok
```



```
def all_childrens(node_id):  
    current_children_ids = Node.objects(  
        parent=node_id  
    ).values("id")  
  
    result = [node_id]  
    for child in current_children_ids:  
        result.extend(all_childrens(child))  
  
    return result
```

```
def all_childrens(node_id):  
    current_children_ids = Node.objects(  
        parent=node_id  
    ).values("id")  
    result = [node_id]  
  
    while current_children_ids:  
        result.extend(current_children_ids)  
        current_children_ids = Node.objects(  
            parent__in=current_children_ids  
        ).values("id")  
        current_children_ids = list(  
            current_children_ids)  
  
    return result
```

```
def all_childrens(node_ids):  
    for n in node_ids:  
        yield n.id  
  
        childrens = Node.objects(  
            parent__in=n.id  
        ).values("id")  
  
        for c in all_childrens(childrens):  
            yield c  
  
list(all_childrens((root_node,)))
```



```
# save source of data into class instance
class DataProcessor(object):
    def __init__(self, data_source):
        self.data_source = data_source

    def process_data(self, *args):
        # do some stuff

processor = DataProcessor(data_source)
processor.process_data()
```

```
# save source of data in partial
from functools import partial

process_with_source = partial(process_data,
                              data_source)

process_with_source()

# currying
process_data = curry(process_data)
initialized = process_data(data_source)(first_arg)
```

```
def get_data(self):  
    data = {}  
    if self.obj_id:  
        # do something with data[]  
    else:  
        if self.item_id:  
            # do something with data[]  
        else:  
            # do something with data[]  
    data["questions"] = process_questions()  
    data["answers"] = process_choices()  
    # do something  
    return data
```

```
def get_data(self, obj_id, item_id):
    def common_part():
        data["questions"] = process_questions()
        data["answers"] = process_choices()
    data = {}
    if obj_id:
        # do something with data[]
    if item_id:
        # do something with data[]
        common_part()
    if obj_id is None and item_id is None:
        # do something with data[]
        common_part()
    return data
```

```
obj = cache.objects[self.obj_id]
if obj.group_id:
    data['group_name'] = cache.groups[
        obj.group_id].title
if self.child_id:
    child = obj.child_by_id(self.child_id)
    if child:
        data["obj_name"] = child.prompt()
    else:
        logger.warning()
```

```
def noop(*args, **kwargs):  
    return  
  
obj = cache.objects[obj_id]  
group = cache.groups.get(obj.group_id)  
child = obj.child_by_id(child_id)  
data["group_name"] = getattr(  
    group, "title", None)  
data["object_name"] = getattr(  
    child, "prompt", noop())
```



БИБЛИОТЕКИ

→ PyFunctional

→ toolz

→ adt

→ Coconat

→ pyrsistent

→ funcy

→ effect

→ hask

→ fn.py

→ PyMonad

EntilZha/PyFunctional

pytoolz/toolz

lllllllll/adt

evhub/coconut

Suor/funcy

tobgu/pyrsistent

python-effect/effect

billpmurphy/hask

kachayev/fn.py

fnl/pymonad

- Функции на все случаи жизни
- Декоратор для каррирования
- Персистентные типы данных
- Синтаксис для функциональной композиции
- Декоратор для обхода хвостовой рекурсии
- Монады и ADT

