

# FP IN PYTHON/JS

---

September 23, 2016

Версия #1

Почему это важно, понимать функциональный подход к программированию?

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




Версия #2



Почему это важно, понимать функциональный подход к программированию?




Потому что это просто ОХ\*\*ЕННО!






- FP is a paradigm, not a language feature
- Python is a multi-paradigm language, that allows to right in functional style
- It's possible to use advantages of FP today

→ Logic is separated  
from data 




- Logic is separated from data 
- Modularity, testability 

- Logic is separated from data 
- Modularity, testability 
- Parallelization 







- Logic is separated from data 
- Modularity, testability 
- Parallelization 

→ Difficult 



- Logic is separated from data 
- Modularity, testability 
- Parallelization 

- Difficult 
- Scary 

- Logic is separated from data 
- Modularity, testability 
- Parallelization 
- Difficult 
- Scary 
- Developers? 

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

# INTRODUCTION IN FP

---

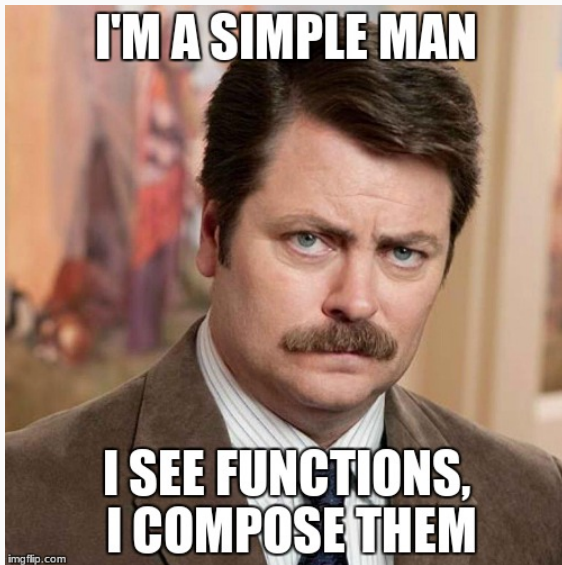
- Immutability
- Pure functions and side effects
- Higher-order functions
- Monads (?)
- Abstract Data Type (ADT)

**Data, data  
never changes**



# PURE FUNCTIONS AND SIDE EFFECTS









# FP SUPPORT IN PYTHON

---

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset
- Higher-order functions

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset
- Higher-order functions
- List comprehension

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset
- Higher-order functions
- List comprehension
- Generators

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset
- Higher-order functions
- List comprehension
- Generators
- itertools

- Immutable data types:
  - string
  - tuple/namedtuple
  - frozenset
- Higher-order functions
- List comprehension
- Generators
- itertools
- functools



→ Tail recursion optimization ✖

- Tail recursion optimization ✖
- Pure functions ✖

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching ✖

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying ✖

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads ✖

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍



- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍
- ADT ✖

- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍
- ADT 🔍

# FP SUPPORT IN JS/COFFEESCRIPT

---

→ Immutable data types:  
Object.freeze

- Immutable data types:  
    `Object.freeze`
- Higher-order functions

- Immutable data types:  
    `Object.freeze`
- Higher-order functions
- List comprehension

- Immutable data types:  
    `Object.freeze`
- Higher-order functions
- List comprehension
- `lodash/underscore`

- Most immutable data types ✖
- Tail recursion optimization ✖



- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching ✖

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying ✖

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads ✖

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍

- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍
- ADT ✖



- Most immutable data types ✖
- Tail recursion optimization ✖
- Pure functions ✖
- Pattern matching 🔍
- Automatic currying 🔍
- Monads 🔍
- ADT 🔍

- Pure Python/JS
- Utility functions
- Third party libraries

# EXAMPLES (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
```

```
record = {"id": 1, "name": "first record"}  
record.id = 2    # ok
```

```
Object.freeze record  
record.id = 3    # nothing was changed
```

# 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 coffee
(v.attr for v in source when condition(v))

# function chain with lodash
_.chain()[..10].slice(5).reverse().value()
```

```
-- 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)
```



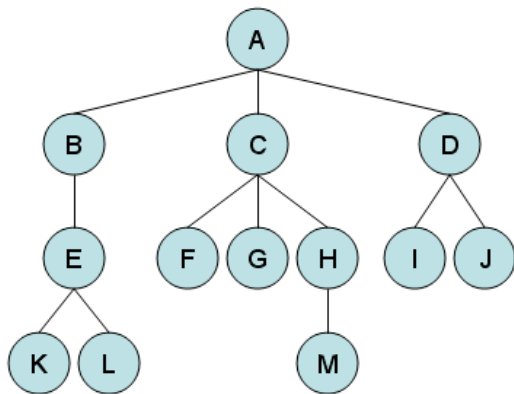
```
var myIterable = {}  
  
myIterable[Symbol.iterator] = function* () {  
    for(i=0; i < 5; i++) {  
        yield i;  
    }  
};  
  
[...myIterable] // [0, 1, 2, 3, 4]
```

```
function* fibs() {  
  let a = 0;  
  let b = 1;  
  while (true) {  
    yield a;  
    [a, b] = [b, a + b];  
  }  
}  
  
const isEven      = n  $\Rightarrow$  n % 2 == 0;  
const lessThanTen = n  $\Rightarrow$  n < 10;  
  
wu(fibs())  
  .filter(isEven)  
  .takeWhile(lessThanTen)  
  .forEach(console.log.bind(console));
```

```
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)
```

```
process_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())
```

```
setUserDisplayData: (userData) →  
  if userData  
    # set username  
    if userData.is_free_trial  
      # show upgrade link if free trial plan  
    if userData.plan_remaining_days?  
      # show remaining days of free trial  
      if userData.plan_remaining_days ≤ 0  
        # set one upgradeText  
      else  
        # set another upgradeText  
  
    $('[data-upgrade-days]').text(upgradeText)  
    $('[data-upgrade-days]').show()
```

```
setUserDisplayData: (userData) →  
  commonPart = (text) →  
    $('[data-upgrade-days]').text(text)  
    $('[data-upgrade-days]').show()  
  if userData  
    # set username  
  else  
    return  
  if userData.is_free_trial  
    # show upgrade link if free trial plan  
  if not userData.plan_remaining_days?  
    return  
  if userData.plan_remaining_days ≤ 0  
    # set one upgradeText  
    commonPart(upgradeText)  
  else  
    # set another upgradeText  
    commonPart(upgradeText)
```





# LIBRARIES

---

→ PyFunctional	<a href="#">EntilZha/PyFunctional</a>
→ toolz	<a href="#">pytoolz/toolz</a>
→ adt	<a href="#">lllllllll/adt</a>
→ Coconat	<a href="#">evhub/coconut</a>
→ pyrsistent	<a href="#">Suor/funcy</a>
→ funcy	<a href="#">tobgu/pyrsistent</a>
→ effect	<a href="#">python-effect/effect</a>
→ hask	<a href="#">billpmurphy/hask</a>
→ fn.py	<a href="#">kachayev/fn.py</a>
→ PyMonad	<a href="#">fnl/pymonad</a>

- A lot of functions
- Decorator @curry
- Persistent data types
- Nice syntax for function composition
- Decorator to bypass tail recursion optimization
- Monads and ADT

QUESTIONS?