FP IN PYTHON

it's simpler than you thought

October 31, 2016

CONTACTS


Dmitry Dolgov, Mindojo

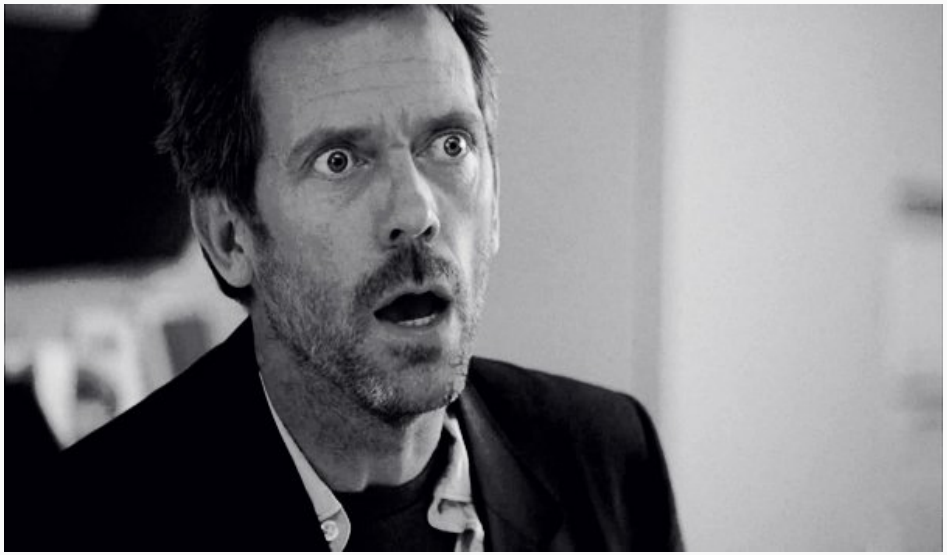

github.com/erthalion


@erthalion


9erthalion6 at gmail dot com

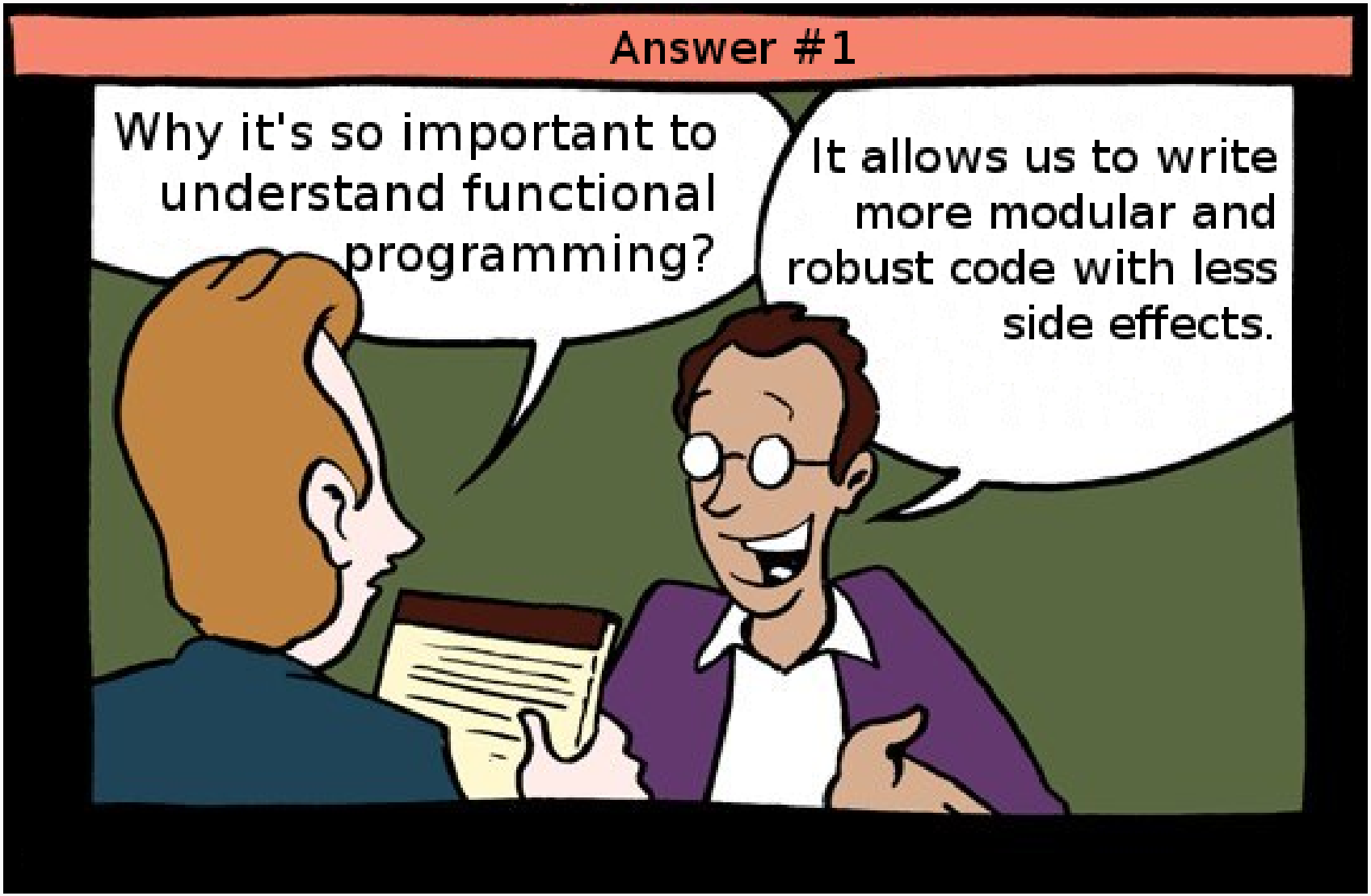
TARGET AUDIENCE?

TARGET AUDIENCE?

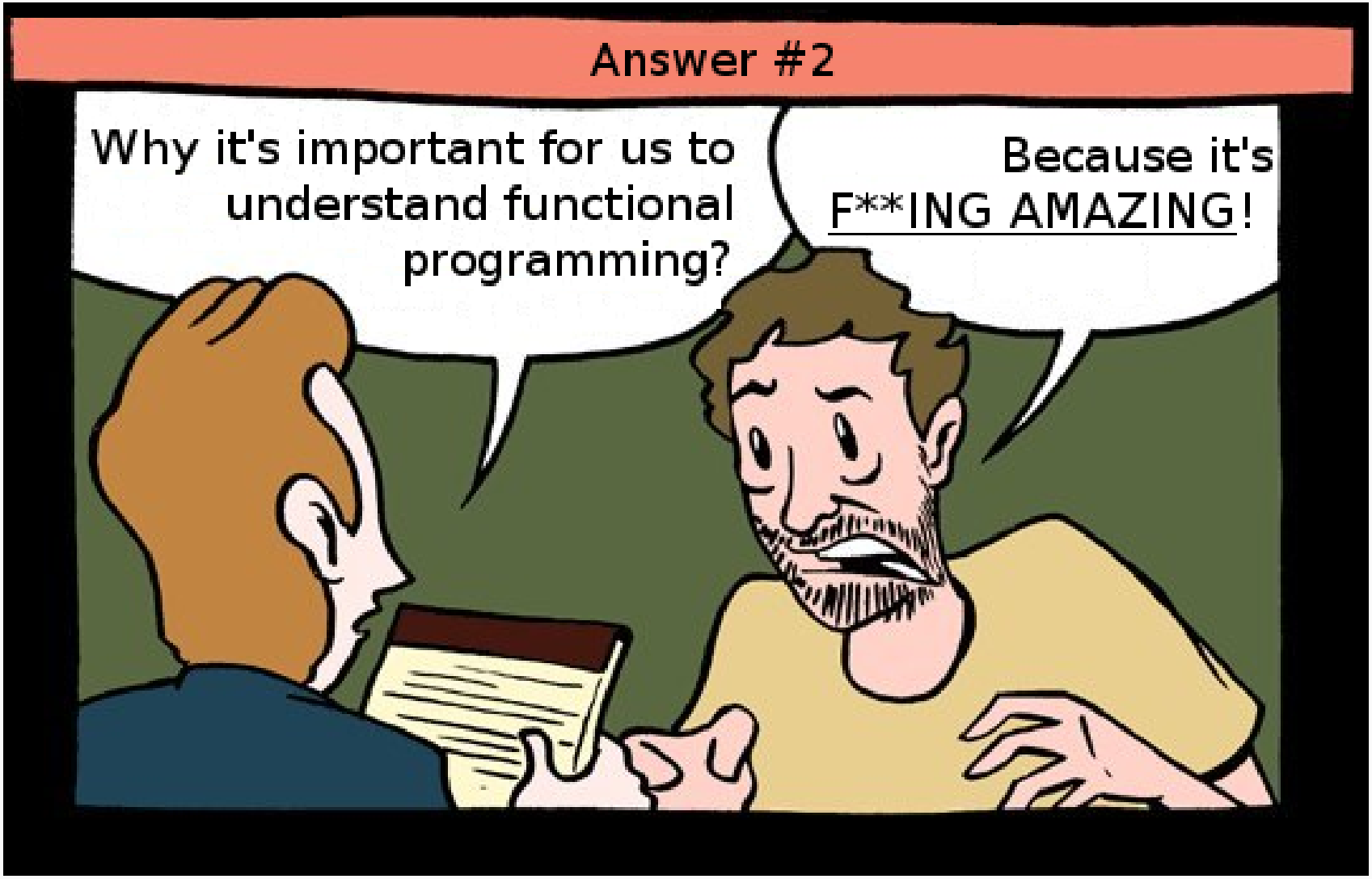


TARGET AUDIENCE?

WHY?


Answer #1
Why it's so important to understand functional programming?
It allows us to write more modular and robust code with less side effects.

WHY?


Answer #2
Why it's important for us to understand functional programming?
Because it's F**ING AMAZING!

WHY?

- ➔ FP is a paradigm, not a language feature
- ➔ Python is a multi-paradigm language, that allows to write functional code
- ➔ It’s possible to use advantages of FP today

PROS AND CONS

- ➔ Logic is separated from data ☑

PROS AND CONS

- ➔ Logic is separated from data ☑
- ➔ Modularity, testability ☑

PROS AND CONS

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

PROS AND CONS

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

- ➔ Difficult ✖

PROS AND CONS

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

- ➔ Difficult ✖
- ➔ Scary ✖

PROS AND CONS

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

- ➔ Difficult ✖
- ➔ Scary ✖
- ➔ Developers? ✖

WHO?

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

INTRODUCTION IN FP

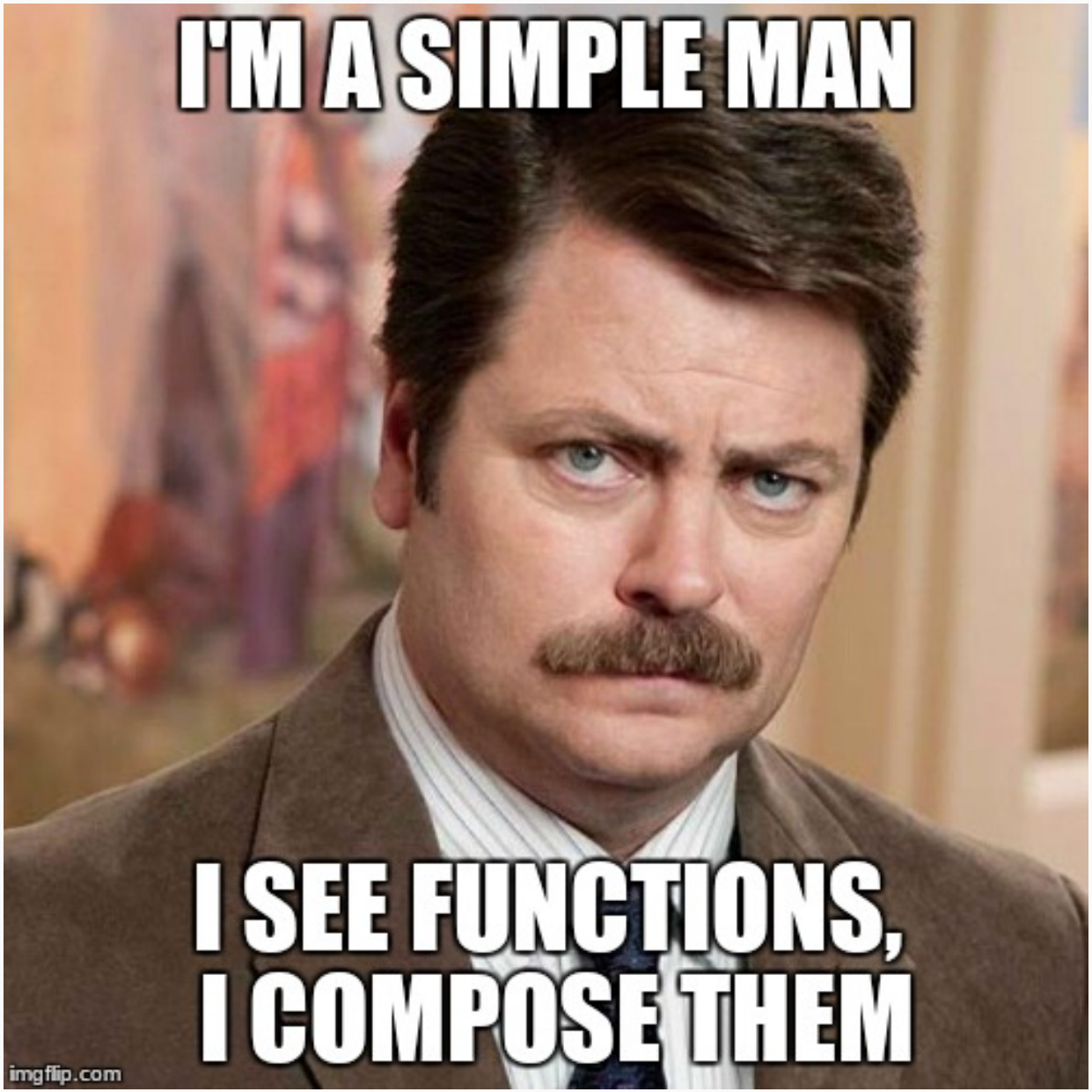
TERMS AND DEFINITIONS

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

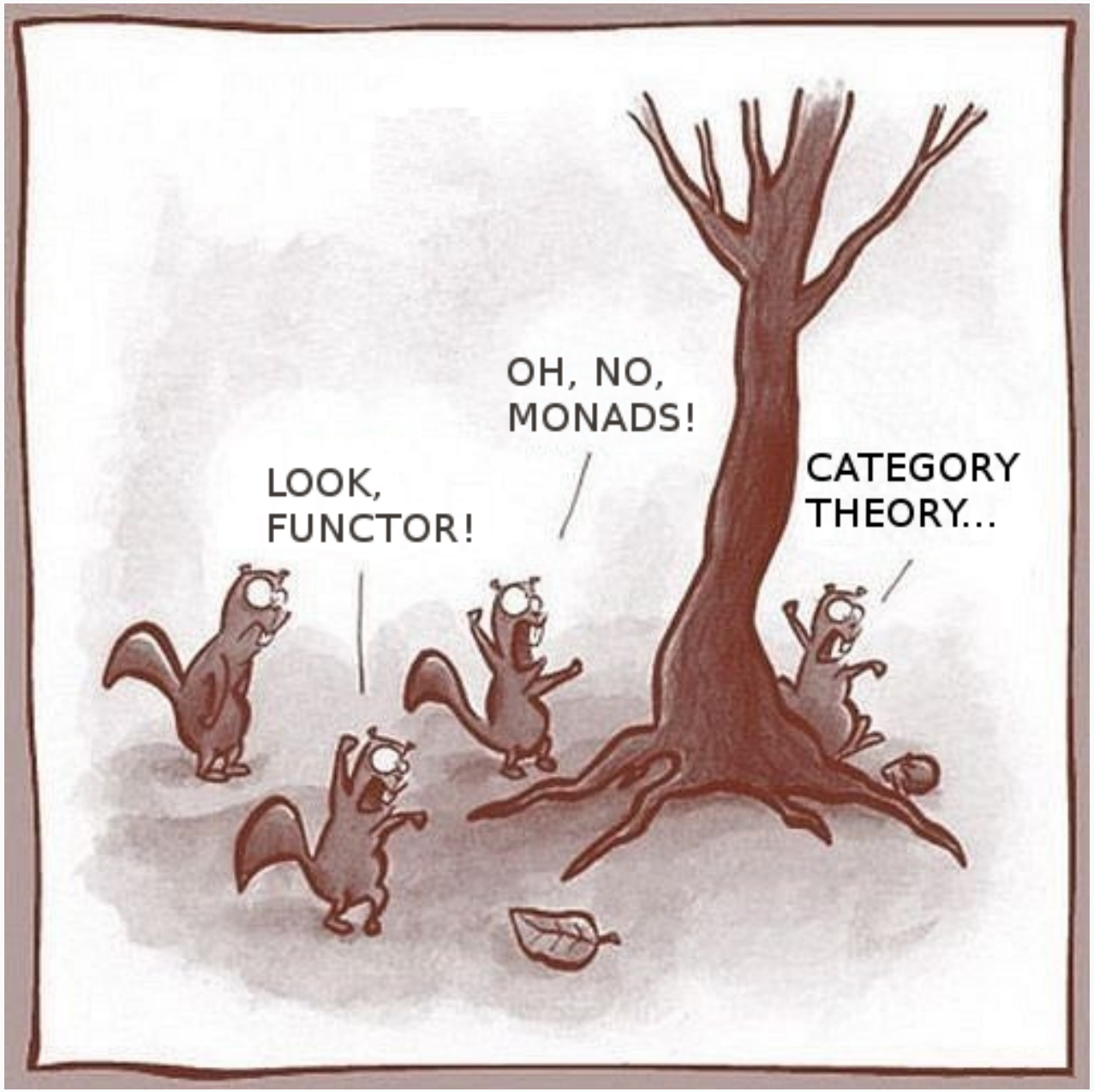
Data, data never changes

PURE FUNCTIONS AND SIDE EFFECTS

HIGHER-ORDER FUNCTIONS AND FUNCTION COMPOSITION



MONADS, ADT



FP SUPPORT IN PYTHON

WHAT'S AVAILABLE

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

WHAT'S AVAILABLE

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

WHAT'S AVAILABLE

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

WHAT'S AVAILABLE

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

WHAT'S AVAILABLE

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

WHAT'S AVAILABLE

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

WHAT'S MISSING

➔ Tail recursion optimization ✖

WHAT'S MISSING

- ➔ Tail recursion optimization ✗
- ➔ Pure functions ✗

What's missing

- ➔ Tail recursion optimization ✗
- ➔ Pure functions ✗
- ➔ Pattern matching ✗

WHAT'S MISSING

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

WHAT'S MISSING

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

What's missing

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

What's missing

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

WHAT'S MISSING

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

WHAT'S MISSING

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

WHAT'S MISSING

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

STRATEGIES

- ➔ Pure Python
- ➔ Utility functions
- ➔ Third party libraries

EXAMPLES (PY2)

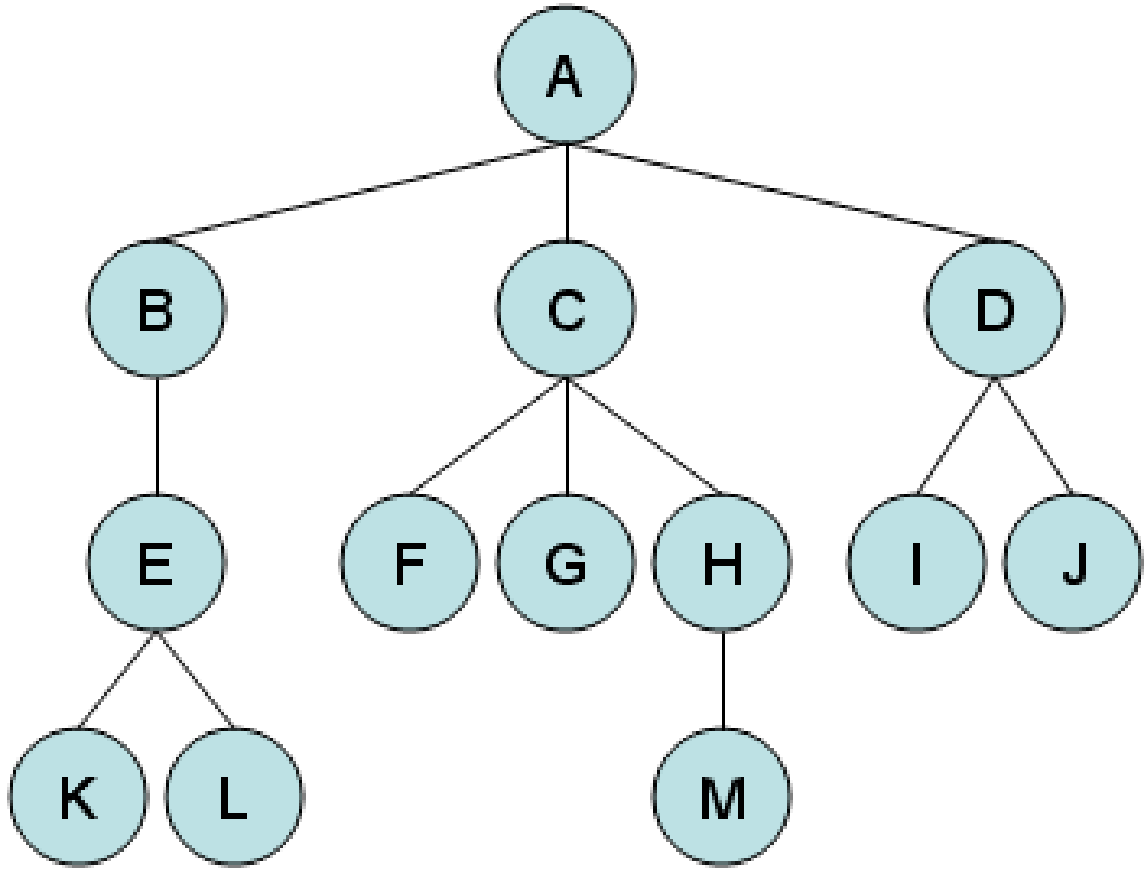
```
from collections import namedtuple

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
```

```python
# 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))
```

```haskell
-- 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
```


A
B
C
D
E
F
G
H
I
J
K
L
M

```
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)()
```

LIBRARIES

LIST OF FP LIBRARIES FOR PYTHON

➔ PyFunctional	EntilZha/PyFunctional
➔ toolz	pytoolz/toolz
➔ adt	lllllllll/adt
➔ Coconat	evhub/coconut
➔ pyrsistent	Suor/funcy
➔ funcy	tobgu/pyrsistent
➔ effect	python-effect/effect
➔ hask	billpmurphy/hask
➔ fn.py	kachayev/fn.py
➔ PyMonad	fnl/pymonad

WHY?

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

Question?