Python ♥ FP (?)

Functional Acrobatics in Python

```
>>> calculate('Answer to life, ' +
...                'the universe & ' +
...                'everything.')
```

```
>>> calculate('Answer to life, ' +
...           'the universe & ' +
...           'everything.')

42
```

```
>>> calculate = len

>>> calculate('Answer to life, ' +
...           'the universe & ' +
...           'everything.')

42
```

```
>>> name = 'Pyry Jahkola'

>>> nick = '@pyrtsa'

>>> work = 'knomi.com' # Startup

>>> vice = 'abusing Python as FP lang'
```

```
>>> import itertools as it
```

```
>>> import itertools as it

>>> useful_fns = [map, filter]
```

```
>>> import itertools as it

>>> useful_fns = [map, filter,
...               tuple, frozenset, list, dict]
```

```
>>> import itertools as it

>>> useful_fns = [map, filter,
...               tuple, frozenset, list, dict,
...               sorted, reversed]
```

```
>>> from functools import reduce
>>> import itertools as it

>>> useful_fns = [map, filter,
...               tuple, frozenset, list, dict,
...               sorted, reversed,
...               min, max, sum, reduce]
```

```
>>> from functools import reduce
```

Let's talk about reduce.

(I hope you knew map and filter already.)

```
>>> from functools import reduce
>>> import operator
>>> reduce(operator.add, [10,20,30])
60
```

```
>>> from functools import reduce
>>> import operator
>>> reduce(operator.add, [10,20,30])
60
>>> reduce(lambda a,b: a+b,[10,20,30])
60
```

```
>>> from functools import reduce
>>> import operator
>>> reduce(operator.add, [10,20,30])
60
>>> reduce(lambda a,b: a+b,[10,20,30])
60
>>> sum([10,20,30])
60
```

```
>>> from functools import reduce
>>> import operator
>>> reduce(operator.mul, [10,20,30])
6000
>>> reduce(lambda a,b: a*b,[10,20,30])
6000
```

```
def reductions(f, xs, *init):
    "Generate all steps of reduction."
    it = iter(xs)
    ac = init[0] if init else next(it)
    yield ac
    for x in it:
        ac = f(ac, x)
        yield ac
```

```
>>> def reductions(f, xs, *init): ...
>>> tup = lambda a, b: (a, b)

>>> list(reductions(tup, [10,20,30]))
[10, (10, 20), ((10, 20), 30)]
```

```
>>> def reductions(f, xs, *init): ...
>>> tup = lambda a, b: (a, b)

>>> list(reductions(tup, [10,20,30]))
[10, (10, 20), ((10, 20), 30)]

>>> list(reductions(tup, [10,20], ()))
[(), ((), 10), (((), 10), 20)]
```

```
nan = float('nan')

def mean(xs):
    s, n = reduce(lambda a, x:
                  (a[0] + x, a[1] + 1),
                  (x for x in xs if x==x),
                  (0, 0))
    return s / n if n else nan

>>> mean([1, nan, 2, 4, 4]) #=> 2.75
```

```
nan = float('nan')

def mean(xs):
    s, n = 0, 0
    for x in (x for x in xs if x==x):
        s += x
        n += 1
    return s / n if n else nan

>>> mean([1, nan, 2, 4, 4]) #=> 2.75
```

Ok, enough with reduce.

Simpler examples with iterators ahead…

```
>>> def do(xs):
...     for _ in xs: pass # "Klever"

>>> names = """Donald Daisy Pluto
... Huey Luey Dewey Scrooge""".split()

>>> do(print(i, n) for i, n
                   in enumerate(names))
# Prints names with indices.
```

```
>>> def first(xs):
...         for x in xs: return x

>>> names = """Donald Daisy Pluto
... Huey Luey Dewey Scrooge""".split()

>>> first(names) # "Donald"
>>> names[0]     # Sure, it's "Donald"
```

```
>>> def first(xs):
...         for x in xs: return x

>>> names = """Donald Daisy Pluto
... Huey Luey Dewey Scrooge""".split()

>>> first(names) # "Donald"
>>> first(n for n in names
                 if len(n) < 5) # "Huey"
```

```
>>> def first(xs):
...         for x in xs: return x

>>> names = open('comic_chars.txt')



>>> first(names)
>>> first(n for n in names
                 if len(n) < 5)
```

```
>>> def first(xs):
...         for x in xs: return x

>>> names = open('comic_chars.txt')



>>> first(names)
>>> first(n for n in names  # Oops!?
                 if len(n) < 5)
```

```
# Careful when sharing generators!


>>> first(names)
>>> first(n for n in names  # Oops!?
                 if len(n) < 5)
```

```
>>> from collections import deque

>>> def last(xs):
...     try:
...         return xs[-1]
...     except TypeError:
...         return deque(xs, 1)[0]
```

Pause.

(Breathe.)

```
def group_by(key, xs, op=list):
    """Group xs by key into a dict
    with op(values) as values."""
    gs = {}
    for x in xs:
        gs.setdefault(key(x),
                      []).append(x)
    return {k: op(gs[k]) for k in gs}
```

```
def progress(xs, n=None, file=stderr):
    """
    Iterate over `xs` (of estimated
    length `n`), printing progress
    display to `file`.
    """

    ...
```

https://gist.github.com/pyrtsa/7083770

Extras

```
def curry(f, n=None):
    """Enable currying on function f up to
    until n positional arguments."""
    if n is None:
        s = inspect.getfullargspec(f)
        n = len(s.args) - len(s.defaults or [])
    if not n: return f # nothing to curry
    def call(f, *a1, **k1):
        if len(a1) >= n: return f(*a1, **k1)
        return lambda *a2, **k2: \
            call(f, *(a1 + a2), **merge(k1, k2))
    return wraps(f)(call(f))
```

```
@curry
def at(n, xs, **kwargs):
    try: return xs[n]
    except (IndexError,KeyError) as e:
        if 'default' in kwargs:
            return kwargs['default']
        raise e

col0 = map(at(0), table)
```

```
def juxt(*fs):
    "Compose a new function returning
    as tuple the results of original
    ones"
    return lambda *a, **k: \
        tuple(f(*a, **k) for f in fs)

at_2n4 = juxt(at(2), at(4))
cols_2n4 = map(at_2n4, table)
```

Thanks.

This thing is in public domain now. Use it wisely. Or not at all. :) -- @pyrtsa