Functional Python

Matthew Rocklin
Sandia National Labs

November 7th, 2013

Setup

What do I need to do this tutorial?

```
Python 2.6+, 3.2+

pip install toolz

pip install toolz --upgrade

git clone git@github.com:mrocklin/pydata-toolz
    or
https://github.com/mrocklin/pydata-toolz/archive/master.zip
```

Disclaimer

Functional programming in Python is controversial

I lie

What is Functional Programming?

- Algebraic types
- Strong compilers
- Macros
- Monads
- `map, filter, reduce` - Standard functional library
- Functions as data, first class, higher order
- Laziness
- Purity - avoid state, side effects, and mutation

Why?

- Functional programmers report productivity increases
- Better at transitioning to parallel processing
- More testable / reliable

Why not?

- Inaccessible

What is Functional Programming?

- ~~Algebraic types~~
- ~~Strong compilers~~
- ~~Macros~~
- ~~Monads~~
- `map, filter, reduce` - Standard functional library
- Functions as data, first class, higher order
- Laziness
- Purity - avoid state, side effects, and mutation

Why?

- Functional programmers report productivity increases
- Better at transitioning to parallel processing
- More testable / reliable

Why not?

- Inaccessible

Outline

Themes:

- Pragmatic Principles
- Functional Standard Library (`toolz`)

Sections:

- Standard Interfaces
- Functions as data
  - `map`, `filter`, `reduce`, ...
  - Composition and orthogonality
- Partial Evaluation and Currying
- Laziness

Data:

- Start with trivial data, e.g. `[1, 2, 3, 4]`
- Textual data - "Tale of Two Cities"
- Genomics - yeast in repo, human on USB drive

Standard Interfaces

Robust systems base themselves on a standard interface

Software

- `numpy.ndarray` → Scientific Python stack
- `JSON` → interaction among web applications
- Files and streaming text → UNIX operating system
- R/Pandas DataFrame

Life

- Supra-national currencies
- Nuts and bolts
- Cell phone chargers
- LEGO bricks

Our standard interface today: Python's core data structures

```
tuple
list
dict
set
generator
function
```

We're not going to create custom objects

```
class MyCustomWidget(Widget):
    ....
```

Book Example

```
# Object oriented

class Book(object):
    def __init__(self, authors, title, isbn):
        self.authors = authors
        self.title = title
        self.isbn = isbn

book = Book(["Neil Gaiman"],
            "American Gods",
            "9780062113450")

}
```

```
# Just a dictionary

book = {
    "authors": ["Neil Gaiman"],
    "title": "American Gods",
    "isbn": "9780062113450"
}
```

Just use a dict!

Example from Shannon Behrens blogpost
http://jjinux.blogspot.com/2013/08/python-dicts-vs-classes.html

What do we get?

- Interoperation with other codebases

```
import json
json.dumps(book_dict)   # works
json.dumps(book_obj)    # raises TypeError
```

- Comprehension by other developers
  Q: *How does this Book object work?*
  A: *Read the docs*
  Q: *How does this dictionary work?*
  A: *It's a dictionary. . . .*
- Laziness/streaming for large datasets
- A powerful standard library. . . .

Composition of tools

Image courtesy of www.beisansystems.com

What do we get?

- Interoperation with other codebases

```
import json
json.dumps(book_dict)   # works
json.dumps(book_obj)    # raises TypeError
```

- Comprehension by other developers
  Q: *How does this Book object work?*
  A: *Read the docs*
  Q: *How does this dictionary work?*
  A: *It's a dictionary. . . .*
- Laziness/streaming for large datasets
- A powerful standard library. . . .

A Standard Functional Library

When we subscribe to a standard interface we usually get lots of free functionality

```
>>> import numpy as np
>>> X = np.arange(20)

>>> X
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,
       10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

>>> X[X>=10].cumsum()
array([ 10,  21,  33,  46,  60,  75,  91, 108, 126, 145])
```

A Standard Functional Library

Want: Concise set of general functions that can be combined to solve most problems

Have: `sum`, `min`, `max`, `sorted`, `set.union`, `...`

Feels incomplete....

More exist in functional languages. This problem has been solved and resolved, finally converging on a concise and powerful set. A standard functional toolbox.

PyToolz

Python's functional standard library is in core, `itertools`, `functools`

But it's lacking many common elements. Extended by `itertoolz` and `functoolz`. Merged into `toolz`.

This same API is implemented in many languages.

```
JavaScript - Underscore.js
Ruby - Enumerable
C# - LINQ
```

As well as most contemporary functional languages like Haskell, Scala, Clojure.