# Functional Programming in Python

Ram Vedam
(@unxn3rd)

# Agenda

- Overview

- FP Features

- Code

- Questions

"I have never considered Python to be heavily influenced by functional languages, no matter what people say or think. I was much more familiar with imperative languages such as C and Algol 68 and although I had made functions first-class objects, I didn't view Python as a functional programming language."

— Guido van Rossum

*Python BDFL*

*Source: http://python-history.blogspot.com/2009/04/origins-of-pythons-functional-features.html*

# History of FP in Python

- Python 2

    - first-class functions introduced

    - higher-order functions such as map, filter, reduce, and apply introduced

        - apply later was deprecated in 2.3 in favor for extended call syntax fn(*args, **kwargs)

    - Anonymous function support

    - support for using closures (via inner functions)

    - Initial support for comprehensions (mainly list comprehensions)

- Python 3

    - reduce reduced to functools namespace

    - apply function removed

    - comprehensions were improved

    - new comprehensions were introduced (generator, dictionary, set)

        - back ported to Python 2.7

# FP Features

- First-class Functions

- Higher-Order Functions

- Callable Objects

- Closures

- Iterator Protocol

- Generators

- Comprehensions

- Decorators

# Recursion

- Considered not "Pythonic" in many situations

    - Examples: recursion is used to loop through a collection of some sort ("iteration by another name")

- Generally discouraged except in a relatively small number of cases.

- Disadvantages in Python:

    - No TCO

    - very limited stack depth (~1000)

# First-Class Functions

- Created at runtime

- Assigned to a variable or element in a data structure

- passed as an argument to a function

- returned as function result

# First-Class Functions

- Python 2 and 3 both support first-class functions

- Support includes:

  - anonymous functions (lambdas)

  - user-defined functions

  - callable objects

# Higher Order Functions

- Definition: A function that takes a function as an argument or returns a function as a result is called a **_higher-order function_**.

- Built-in higher order functions: sorted, map, filter, reduce, apply (removed in Python 3)

# Higher Order Functions

|          | Python 2            | Python 3                    |
|----------|---------------------|-----------------------------|
| sorted   | returns a list      | returns a list              |
| map      | returns an iterable | returns an iterable         |
| filter   | returns a list      | returns an iterable         |
| reduce   | part of stdlib      | moved to functools          |
| apply    | deprecated          | removed                     |

# Callables

- can determine if something is callable by callable() method

- 7 types

  1. User-defined functions

  2. built-in functions

  3. built-in methods

  4. methods

  5. Classes

  6. Class instances (if __call__ is implemented for Class)

  7. generator functions

*Source: Fluent Python*

# Closures

"a closure is a function with an extended scope that encompasses *nonglobal* variables referenced in the body of the function but not defined there"

-  Luciano Ramalho

Fluent Python

# Closures

- nonglobal variables

  - usually variables defined in the scope that the actual function is defined (module scope, program scope, global scope, etc.)

# Iterator Protocol

- Methods to be implemented

  - __next__

  - __iter__

- Any object that implements the iterator protocol is an *iterable* object.

# Iterator Protocol

- Advantages

  - Custom Objects can be used in Comprehensions (later)

  - Python standard library higher-order functions that act on standard Python collections can also act on custom objects now

  - Allows one to write very lazy data structures in Python.

  - Mixed with OO and subclassing from Sequence, allows one to create Streams that act very close to their Haskell Equivalent

# Generators

- A special kind of iterable object

- fully implements iterator interface

- generator typically yield results using yield keyword

- generator expressions vs generator functions

# Comprehensions

- List comprehensions have been around the longest

- Recent versions of Python (2.7 and later) include:

  - generator comprehensions

  - set comprehensions

  - dict comprehensions

# Comprehensions

- Advantages

  - Leads to a more declarative style of iterating and transforming collections of data

- Disadvantages

  - Nested comprehensions can lead to unreadable and unmaintainable code

    - after 2 or 3 levels, it would be better and preferable to extract functions and call them instead

# Decorators

- callable that takes a function as an argument

- may perform processing with the decorated function and returns the function itself

- replaces function with another function or callable object

- executed at import-time

# Packages

- operator

  - Provides methods for standard intrinsic operators

  - provides methods for generating functions to access members of tuples and instance members

- functools module

  - Methods for partial application

  - convenience functions for decorators

  - back ported to Python 2.7

# Code

# Is that it?

- Only scratched the surface

- Awesome Third-Party Library system:

  - multipledispatch

  - more_itertools

  - hypothesis

  - pyrsistent

# Summary

- Use of Functional Programming in Python:

  - leads to cleaner code

  - code that concentrates on the "what" instead of the "how"

  - lazier code that is computed on demand

  - Code that still follows the "Python Way"