Map, Filter, Reduce

Matthew Rocklin

November 7th, 2013

Functions as data

Functions are first class objects.

```
def f(x):
    return ...

g = f                       # We treat functions like variables

result = map(f, 3)          # We use functions as arguments to other functions

f = makeFunc(...)           # We use functions that make other functions
```

Functions that consume or produce functions are called *higher order*

Functions as data

Iterate over a list of functions just like we iterate over data

```
>>> data = [5, 2, 9, 4, 3]
>>> functions = [len, sum, min, max]
>>> for f in functions:
>>>     print f.__name__, f(data)
len 5
sum 23
min 2
max 9
```

Map

Download the HTML text from a list of web addresses

```
from urllib import urlopen

urls = [’http://www.google.com’,
        ’http://www.wikipedia.com’,
        ’http://www.apple.com’]



result = []
for item in urls:
    result.append(urlopen(item))
return result
```

Map

Compute the Fibonacci numbers on a list of integers

```python
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return e

integers = [1, 2, 3, 4, 5]

result = []
for item in integers:
    result.append(fib(item))
return result
```

Map

Pull out the common structure into a higher order function

```
def map(function, sequence):
    result = []
    for item in sequence:
        result.append(function(item))
    return result

html_texts = map(urlopen, urls)
fib_integers = map(fib, integers)
```

The `map` function is higher order.

List Comprehensions

This pattern is so important that we elevate it to *syntax*

```
html_texts = map(urlopen, urls)
fib_integers = map(fib, integers)

html_texts = [urlopen(url) for url in urls]
fib_integers = [fib(i) for i in integers]
```

Filter

Think about other common programming patterns.

Transform these patterns into higher order functions

```
def iseven(n):                          # A predicate
    return n % 2 == 0

>>> filter(iseven, [1, 2, 3, 4, 5, 6, 7, 8]
[2, 4, 6, 8]

>>> from sympy import isprime
>>> filter(isprime, [1, 2, 3, 4, 5, 6, 7, 8])
[2, 3, 5, 7]

>>> [i for i in [1, 2, 3, 4, 5, 6, 7, 8] if isprime(i)]
[2, 3, 5, 7]

def filter(predicate, sequence):
    result = []
    for item in sequence:
        if predicate(item):
            result.append(item)
    return result
```

Reduce

```python
def add(x, y):
    return x + y
```

```python
def sum(data):
    result = 0
    for x in data:
        result = add(result, x)
    return result

sum([5, 2, 3])
```

Reduce

```
def lesser(x, y):
    if x < y:
        return x
    else:
        return y

def min(data):
    result = 999999999999
    for x in data:
        result = lesser(result, x)
    return result

min([5, 2, 3])
```

Reduce

```
# Sum
result = sum(data)
result = reduce(add, data, 0)

# Min
result = min(data)
result = reduce(lesser, data, 9999999999)

binary operator -> list operator

add             -> sum
mul             -> product
lesser          -> min
greater         -> max
...
```