# Functional Programming

While it may seem odd (or just annoying) to talk about the functional features before the object oriented ones, I think they are simpler to show before rather than after.

## Functions are first-class

Functions in Python can be referred to by name. They are run only when you *call* them with parenthesis:

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

def subtract(x,y): return x - y

def do_binary_op(op, x, y):
    return op(x, y)

z = do_binary_op(add, 5, 10) # z = 15
```

## Closures

```
def make_rotater(seq):
    def rotate():
        val = seq.pop(0)
        seq.append(val)
        return val
    return rotate

r = make_rotater([1,2,3])
assert [r(), r(), r(), r(), r()] == [1,2,3,1,2]
```

You can safely refer to objects from the scope a function was declared in, even after that scope ends.

Closure Limitation/Gotcha

An annoying implementation detail of Pythons closures is that *only* objects are properly closed over. Non-reference types (strings and numbers) only exist for the duration of the scope they were declared in.

## Decorators

Python has a special syntax that allows you to wrap or "decorate" functions (and classes) at compile time:

```
def off_by_one(original_function):
    def new_function(x, y):
        return original_function(x, y) + 1
    return new_function

@off_by_one
def add(x, y):
    return x + y
```

## Decorators: What just happened?

The `off_by_one` function takes a function as it's sole argument and returns a new function.

We *decorated* the `add` function with `off_by_one` by placing `@off_by_one` above the function definition. This is equivalent to writing the following after the definition:

```
add = off_by_one(add)
```

http://docs.python.org/glossary.html#term-decorator

## Lambda (anonymous) functions

Short functions can be declared inline using the `lambda` syntax:

```
is_odd = lambda x: x % 2
```

These are useful when used with functions that take a function as a parameter, like `sorted`:

```
numbers = [55, 22, 53, 16, 67, 363612, 64361, 12556]
# Sort the numbers by their least significant byte
lsB_ordered = sorted(numbers, key=lambda x: x & 0xFF)
```

http://docs.python.org/library/functions.html#sorted

## Generator Functions

Generator functions maintain state between calls. By using `yield` instead of return, the function will pick up on the next line the next time you call it.

```
def get_all_records(lookup, keys):
    for key in keys:
        yield datasource.get(key)

for record in get_all_records(lookup, keys):
    print record
```