Closures and Generators

OOSD

School of Information Technology
Otago Polytechnic
Dunedin, New Zealand

This works

```
x = 4
def f():
    return x

f()   # returns 4
```

Closures

"closures (also lexical closures or function closures) are a technique for implementing lexically scoped name binding in languages with first-class functions." (Wikipedia)
Basically, they are a way of taking the previous example and exploiting first class functions to do something useful.

Constant function generator

```
def constant_builder(cval):
    def f():
        return cval
    return f

four = constant_builder(4)
```

Kind of interesting, but not that useful...

A counter

```
 def counter_builder(start):
     count = [start]
     def f():
         val = count[0]
         count[0] += 1
         return val
     return f

counter = counter_builder(1)
```

Memoisation

```
def factorial(n):
    if n == 0:
        return 1
    else
        return n * factorial(n-1)
```

Suppose we use this to compute 50!, and then we later compute 51!. Wouldn't it be nice if we could have saved the earlier result?

Memoisation

```
 def fact_builder():
     memo = [1, 1]
     def f(in):
         try:
             return memo[n]
         except IndexError:
             result = n * f(n-1)
             memo[n] = result
             return result
     return f

factorial = fact_builder()
```

Generator functions

Often we need an arbitrarily long sequence of values, like the counter function we saw earlier. We saw that we can produce these with closures, but it's a common enough situation so that Python provides *generator functions* to handle this.

Generator example

```
def counter(n):
    count = 0
    while count < n:
        yield count
        count += 1
```

Exercises

1. Write a function that computes the nth *Fibonacci number*.
2. Use a closure to write a memoised version of your Fibonacci function. Use the `timeit` module to compare the speeds of your functions.
3. Take the function below and reimplement it as a generator function.

```
from math import sqrt
def primes(n):
    if n == 0:
        return []
    elif n == 1:
        return [1]
    else:
        p = primes(int(sqrt(n)))
        no_p = {j for i in p for j in range(i*2, n, i)}
        p = {x for x in range(2, n) if x not in no_p}
        return p
```