Closures**¶**

Functions are first class citizens in Python; can be

* passed as args to other funcs
* returned from other funcs
* stored in data structs
* ...

In [ ]:

def callf(func, arg):  
 return func(arg)

In [ ]:

def hello(name):  
 print 'hello'+name  
   
callf(hello, 'world')

In [ ]:

Or defined nested

In [ ]:

def f(n):  
 x=1  
 def g():  
 print x,n  
 return g

In [ ]:

h = f(10)  
h

As a matter of fact, functions are objects with attributes of their own

In [ ]:

h.

In [ ]:

h.func\_closure

In [ ]:

h.func\_closure[0].cell\_contents

Internal functions *remember* their defining context!

That is a Closure !**¶**

In [ ]:

Application of closures**¶**

Lazy evaluation

In [ ]:

from urllib import urlopen

In [ ]:

def page(url):  
 def get():  
 return urlopen(url).read()  
 return get

In [ ]:

python = page('http://python.org')  
jython = page('http://jython.org')  
python

In [ ]:

pyorg = python()

In [ ]:

In [ ]:

Or a way of preserving state between calls

In [ ]:

def countdown(n):  
 def next():  
 #nonlocal n  
 n -= 1  
 return n  
 return next

In [ ]:

next = countdown(10)  
next()  
next()  
next()

NonLocal**¶**

* Makes a 'mutable' closure for a function
* Closures are immutable(aka read-only) by default

Defined at this PEP: <https://www.python.org/dev/peps/pep-3104/>

**NOT** backported to 2.x!

Solution for 2.x?

* Give the context a mutable data struct: list, dict, ..
* Make required changes inside the struct (usually a dict)

In [ ]:

def countdown(n):  
 d = {'n':n}  
 def next():  
 d['n'] -= 1  
 return d['n']  
 return next

In [ ]:

next = countdown(10)  
next()  
next()  
next()

Other applications:

* for avoiding very small/function-only classes
* for implementing generators, as we are very soon seeing..

In [ ]: