Tema 1 Programación Funcional

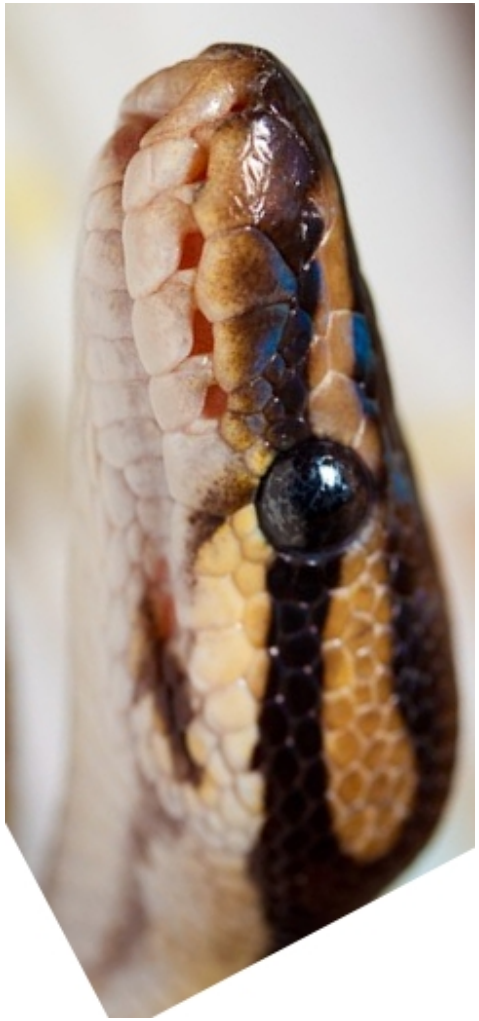
Curso de Python Avanzado

Juan Pedro Bolívar Puente

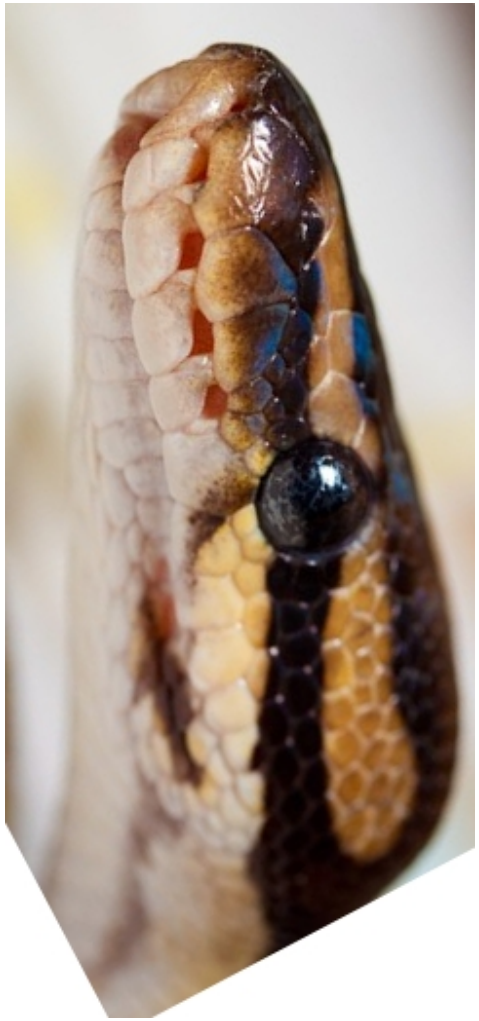
Instituto de Astrofísica de Andalucía

Mayo de 2011

Índice



Índice



Repaso ...

Sintáxis básica

```
def funcion (param, clave="default"):
    print "Ejecutando: funcion (" + \
          str (param) + ", clave=" \
          + str (clave) + ")"
    return None
```

```
funcion (1)
funcion (1, "mi clave")
```

Funciones variádicas

En la lista de parámetros ...

Operador * Captura en un nombre los parámetros restantes como una tupla.

Operador ** Captura en un nombre las claves restantes como un diccionario.

```
def funcion (*args, **keys):
    print "--␣Llamando␣a␣funcion␣con:␣--␣"
    print "Parametros:␣", args
    print "Claves:␣␣␣␣␣", keys
```

Funciones variádicas

En la lista de parámetros ...

Operador * Captura en un nombre los parámetros restantes como una tupla.

Operador ** Captura en un nombre las claves restantes como un diccionario.

```
funcion (1, 2, 3, 'manolete')
funcion (nombre    = 'Juan Pedro',
         apellidos = 'Bolivar Puente')
```

Funciones variádicas

En una llamada a función

Operador * Expande una secuencia como argumentos a la funcion.

Operador ** Expande un diccionario como argumentos clave a la funcion.

```
def sum3 (a, b, c):
    return a + b + c
print sum3 (* range (3))
```

Funciones variádicas

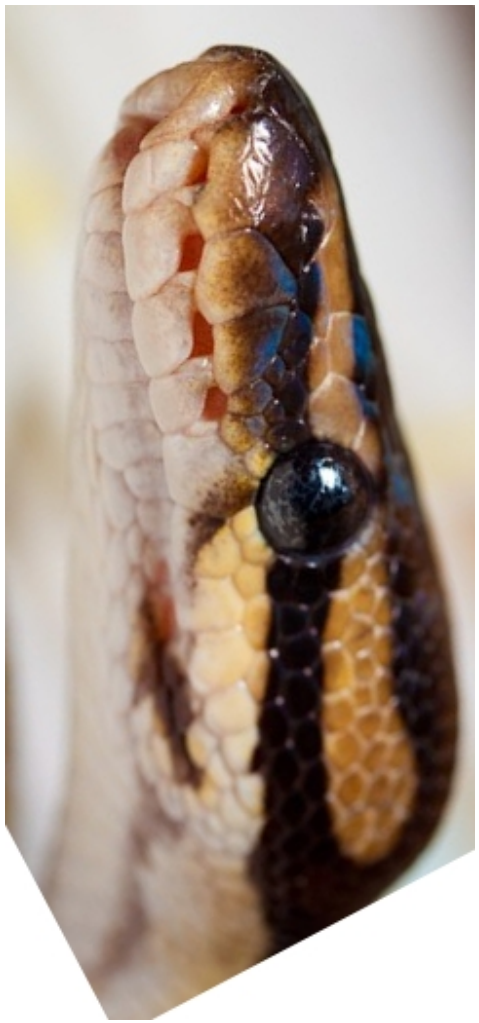
En una llamada a función

Operador * Expande una secuencia como argumentos a la funcion.

Operador ** Expande un diccionario como argumentos clave a la funcion.

```
def persona (nombre = 'Anonimo',
             apellidos = 'Bastardo',
             dni = 'Sin DNI'):
    print apellidos+",", nombre, "("+dni+")"
yo = { 'nombre'    : 'Jhon',
       'apellidos' : 'Doe' }
persona (**yo)
```

Índice



Programación Funcional

... érase una vez la conjetura de Hilbert ...

Modelo Máquina de Turing

- Paradigma Imperativo ⇒ Estructurado ⇒ Orientado a Objetos
- Fortran, Algol, C, Smalltalk, Java

Modelo Lambda Calculi

- Paradigma Funcional ⇒ Lógico ⇒ Declarativo
- Lisp, ML, Prolog, Erlang, Haskell

Figura: Alonzo Church

Programación multiparadigma

Python es multiparadigma

Opinión personal ...

Macrodiseño $\Rightarrow$ Orientación a Objetos

Microdiseño $\Rightarrow$ Programación Funcional

Programación imperativa

- Unidad sintáctica elemental: la sentencia
- Operación elemental: la asignación
- Modelo: variables ... "computar"

```
import random as r
x = 1
y = r.randint(0,10)
if y % 2:
  x = x + 3
else:
  x = x - 2
print x
```

Desventaja

Dificil razonar formalmente sobre el estado.

Programación funcional

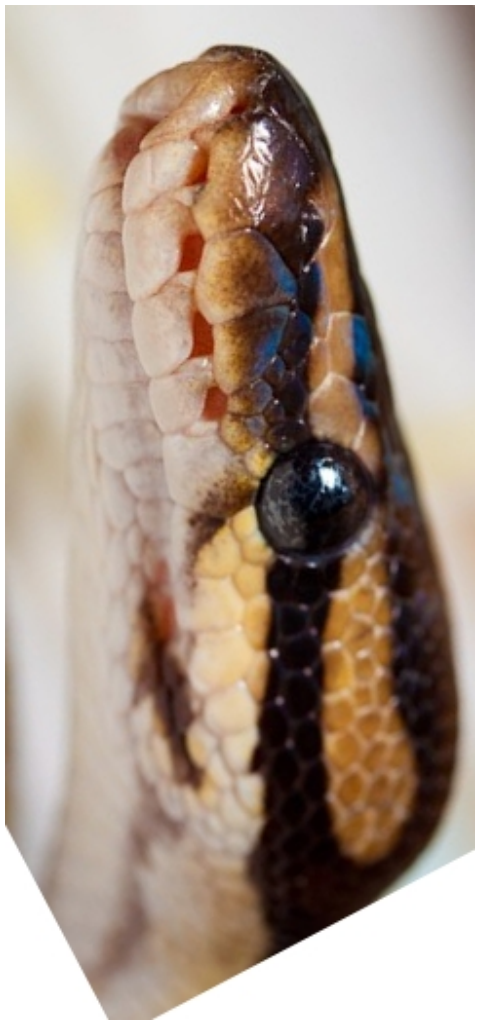
- Unidad sintáctica elemental:la expresión
- Operación elemental: la aplicación
- Modelo: valores ... "calcular"

```
y = random.randint (0, 10)
x = 1 + (3 if y % 2 else -2)
print x
```

Desventaja

Entrada y salida. Mutabilidad.

Índice



Funciones de primer orden

Programación funcional $\Rightarrow$ Funciones valores "normales"

Genericidad Las funciones pueden *pasarse como parámetro*.

Instanciación Las funciones pueden *devolver otras funciones*.

Abstracción procedural Cualquier sentencia puede *convertirse en función*.

Embebimiento Las funciones pueden *almacenerse en estructuras de datos*.

Genericidad

Las funciones pueden *pasarse como parámetro*.

Ejemplo

```
def ejecutarfun (funcion):
    res = funcion ()
    print "Resultado: " + str (res)

ejecutarfun (random.random)
```

Instanciación

Las funciones pueden *devolver otras funciones*.

Ejemplo

```
def devuelvefun (param):
    if param: return random.random
    else: return list

fun = devuelvefun (True)
print fun ()
fun = devuelvefun (False)
print fun ()
```

Abstracción procedural

Cualquier sentencia puede *convertirse en función*.

Ejemplo (Abstracción + Instanciación)

```
def make_sumador (k):
    def sumador (x):
        return x + k
    return sumador

mas_dos = make_sumador (2)
print mas_dos (1) # Imprime 3
print mas_dos (2) # Imprime 4
```

Abstracción procedural ... Lambdas

- **Lambda** = Función anónima (sin nombre).
- ¡Sólo expresiones!

Ejemplo (Lambdas...)

```
def make_sumador_lambda (k):
    return lambda x: x + k

mas_dos = make_sumador_lambda (2)
print mas_dos (1)
print mas_dos (2)
```

Abstracción procedural ... Lambdas

El **cierre** captura valores no nombres

I.e. no podemos modificar la variable "de fuera"

Ejemplo incorrecto (... y requiere Python 3)

```
def make_contador (x):
    def contador ():
        nonlocal x
        x += 1
    return contador
var = 2
cnt = make_contador (var)
cnt (); cnt (); print var # 2!!
```

Abstracción procedural ... Lambdas

Ejemplo "correcto"

```
def make_contador (x):
    def contador ():
        x [0] += 1
    return contador
var = [2]
cnt = make_contador (var)
cnt (); cnt (); print var # [4], Ok
```

Reflexión ...

¿Estado mutable + comportamiento? ⇒ ¡Objetos!

Alternativa Generadores (tema 3)

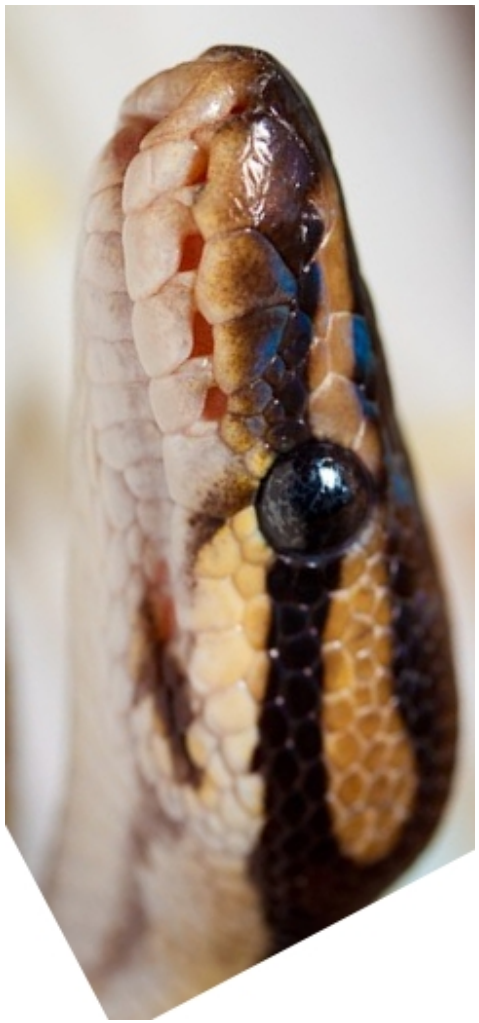
Embebimiento

Cualquier función puede *guardarse en cualquier parte*.

Ejemplo (Embebimiento)

```
from operator import add, sub, div #, mul
funcs = { '+'   : add,
          '-'   : sub,
          '*'   : lambda a, b: a * b,
          '/'   : div }
print funcs ['+'] (1, 2)
print funcs ['-'] (1, 2)
print funcs ['*'] (1, 2)
print funcs ['/'] (1, 2)
```

Índice



Funciones de alto orden

Los tres mosqueteros de la programación funcional

Filosofía

Trabajar sobre listas $\Rightarrow$ Abstraer la iteración

`map (func, lista)`

Devuelve una lista aplicando `func` a cada elemento

`reduce (func, lista, (primero))`

Devuelve un valor aplicando la operación binaria `func`

`filter (pred, lista)`

Devuelve una lista filtrando con el predicado `pred`

La función map (func, lista)

```
def map (func, lista):
    res = []
    for x in lista:
        res.append (func (x))
    return res

def map (func, lista):
    return [ func (x) for x in lista ]
```

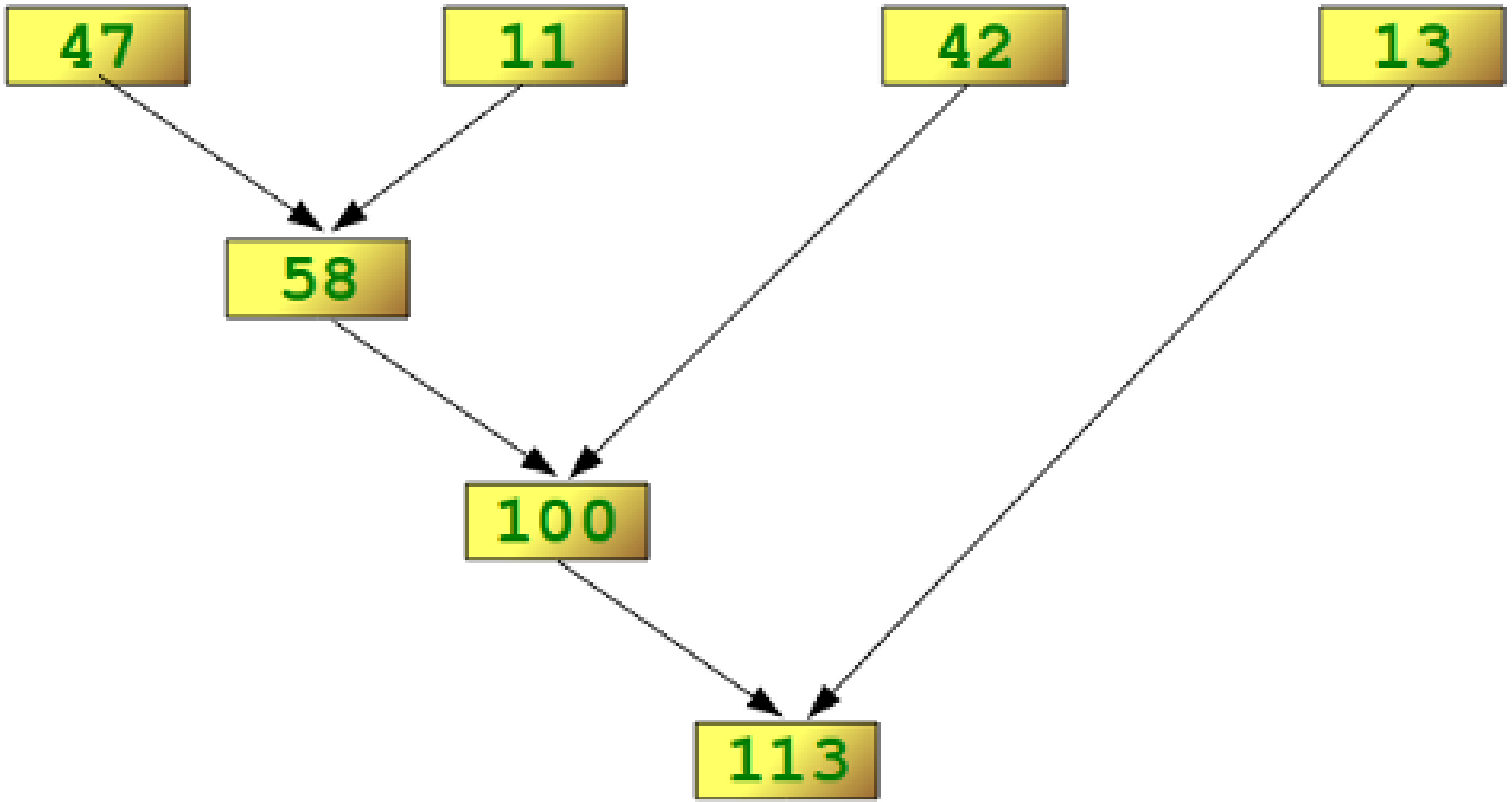
```
print map (lambda x: x * 2, range (10))
```

La función reduce (func, lista)

```
def reduce (func, lista, inic = None):
    if inic is None:
        inic, lista = lista [0], lista
    for x in lista:
        inic = func (inic, x)
    return inic
```

```
print reduce (lambda x, y: x + y, range (10))
```

La función *reduce* ...



La función filter (pred, lista)

```
def filter (pred, lista):
    res = []
    for x in lista:
        if pred (x):
            res.append (x)
    return res

def filter (pred, lista):
    return [ x for x in lista      \
                 if pred (x) ]
```

```
print filter (lambda x: x%2 == 0, range (10))
```

map + filter vs comprehensión

¡No hay que fliparse!

```
map (lambda x: x*2, filter (
          lambda x: x%2==0, range(10)))
```

Mejor...

```
[ x*2 for x in range(10) if x%2==0 ]
```

```
range (0, 20, 4)
```

Currificación

Currificación = Fijar parámetros de una función.

A mano

```
from operator import add
addtwo = lambda x: add (2, x) print
print addtwo (3) # 5
```

Con functools.partial

```
from functools import partial
addtwo = partial (add, 2)
print addtwo (3) # 5
```

Otras utilidades

Emparejar los elementos de dos listas ...

```
zip ('hola', range (3))
```

Verdadero si todos son verdaderos ...

```
all ([True, [], True])
```

Verdadero si alguno es verdadero ...

```
any ([True, [], True])
```

Recursos adicionales

Dive Into Python
Mark Pilgrim
http://diveintopython.org/functional_programming/index.html

Functional Programming HOWTO
Andrew M. Kuchling
http://www.amk.ca/python/writing/functional

Recursos adicionales

Concepts, Techniques, and Models of Computer Programming
Peter Van Roy, Seif Haridi
MIT Press 2005

`python-functional`
Collin Winter
`http://oakwinter.com/code/functional/documentation/`

`python-goopy`
Google
http://goog-goopy.sourceforge.net/

¿Preguntas?

Muchas gracias por su atención.