

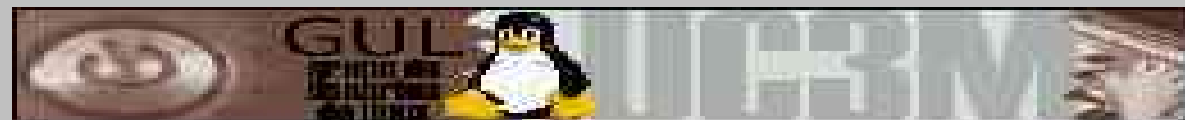
Python: El lenguaje de Moda

Grupo de Usuarios de Linux

Pablo Barrera González

<barrera@gsync.escet.urjc.es>

22 de Marzo de 2004



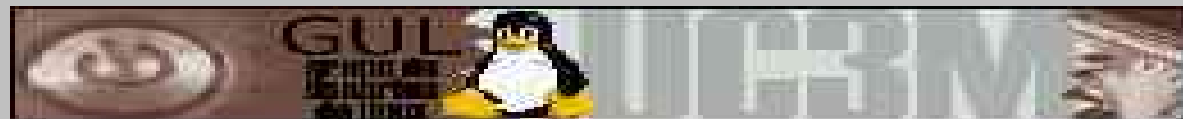
¿Por qué Python?



- Python es fácil de aprender
- Python es sencillo de usar
- Python es potente

¿Qué es Python?

- Es un lenguaje de programación
- Fue creado en las navidades de 1989
 - Su autor es Guido van Rossum
 - En origen era un lenguaje para la gestión de Amoeba
- Basado en ABC y Modula-3
- En febrero de 1991 pasa a USENET
- A partir de ahí el lenguaje no ha dejado de crecer
 - Actualmente tenemos la versión 2.3



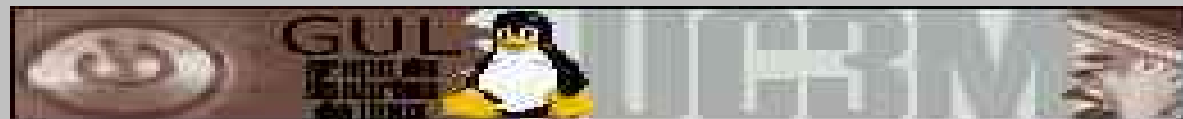
¿Por qué es especial? (I)

- Es libre (y gratis)
- Fácil de escribir
- Fácil de leer
- Fácil de mantener
- Propósito general



¿Por qué es especial? (II)

- Alto nivel
- Orientado a objetos
- Interpretado
- Introspectivo



¿Por qué es especial? (III)

- Extensible
- Completo
- Dinámico
- Robusto
- Múltiples plataformas
- Colaborativo



¿Por qué es especial? (y IV)

- Herencia múltiple
- Funciones sobre listas
- Funciones lambda
- ...



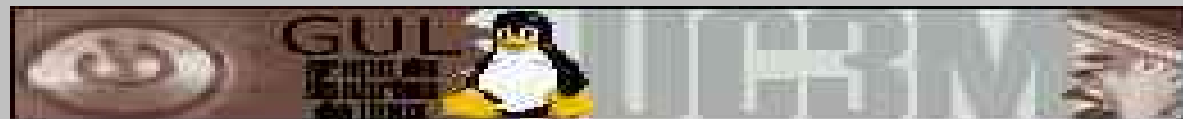
¿Quién lo usa?

- BEA Systems
- Walt Disney Company
- GE Aircraft Engines
- Google
- Hewlett-Packard
- IBM
- Industrial Light + Magic
- Lawrence Livermore National Laboratories
- Microsoft
- NASA
- National Center for Atmospheric Research
- The Nature Conservancy
- Red Hat
- U.S. National Weather Service
- U.S. Navy
- Verio
- Verizon
- WebMD
- Xerox
- Yahoo!
- Zope Corporation



Instalando Python

- <http://www.python.org>
- Instalado en muchas distribuciones GNU/Linux
- Incluido en Debian GNU/Linux
- Autoejecutable en Microsoft Windows



Interprete

```
pablo@aliazul:~$ python
Python 2.2.3+ (#1, Sep 30 2003, 01:19:08)
[GCC 3.3.2 20030908 (Debian prerelease)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```



Definición de variables

- No hace falta definir la variable
- Los tipos de datos son dinámicos
- Es sensible a las mayúsculas y minúsculas

```
>>> a = 1
>>> b = 1.0
>>> c = "1.0"
>>> d = 'hola'
>>> e = 5j
```

Buceando dentro de Python

- Lenguaje introspectivo
- `dir()` muestra los objetos que hay en memoria
- Los métodos también son objetos

```
>>> dir()
```

```
['__builtins__', '__doc__', '__name__', 'a', 'b', 'c', 'd', 'e']
```

```
>>> dir('__builtins__')
```

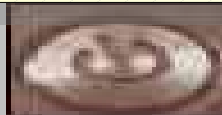
```
['__add__', '__class__', '__contains__', '__delattr__',  
 '__doc__', '__eq__', '__ge__', '__getattr__',  
 '__getitem__', '__getslice__', '__gt__', '__hash__',  
 '__init__', '__le__', '__len__', ...]
```



Buceando dentro de Python (II)

- Es muy útil cuando tienes mala memoria

```
>>> dir([ ])
['__add__', '__class__', '__contains__', '__delattr__',
 '__delitem__', '__delslice__', '__doc__', '__eq__',
 '__ge__', '__getattr__', '__getitem__', '__getslice__',
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
 '__le__', '__len__', '__lt__', '__mul__', '__ne__',
 '__new__', '__reduce__', '__repr__', '__rmul__',
 '__setattr__', '__setitem__', '__setslice__', '__str__',
 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
```



Un poquito de ayuda

```
help([])
```

```
Help on class list in module __builtin__:
```

```
class list(object)
```

```
| list() -> new list
```

```
| list(sequence) -> new list initialized from sequence's  
items
```

```
|  
| Methods defined here:
```

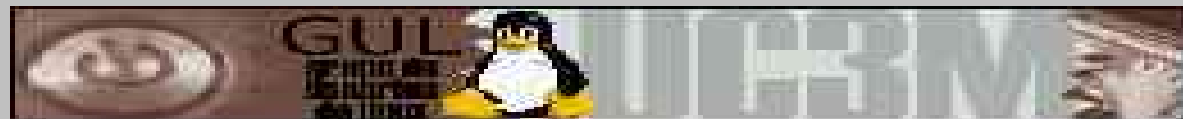
```
| __add__(...)
```

```
| x.__add__(y)  $\Leftrightarrow$  x+y
```



Tipos de datos básicos

- Enteros
- Coma flotante
- Números complejos
- Números de precisión arbitraria
- Cadenas de caracteres
- Tuplas
- Listas
- Diccionarios
- Son dinámicos



Listas

- Se identifican por []
- Lista vacía: []
- Elementos separados por comas
 - [1,2,3,4]
- Elementos heterogéneos
 - [1,(2,4), "avión",["gul","linux","python"]]
- Acceso a un elemento:
 - lista[posición]
- Listas dentro de listas
 - lista[índice1][índice2]...[índiceN]



Listas (II)

- Los índices pueden contar también desde el final:

0	1	2	3	4
-5	-4	-3	-2	-1

Listas (III)

- También se pueden seleccionar fragmentos:
 - lista[índice_inicial:índice_final]

```
>>> a = range(5)
```

```
>>> a
```

```
[0, 1, 2, 3, 4]
```

```
>>> a[1:-2]
```

```
[1, 2]
```

- Devuelve una lista

Tuplas

- Similares a las estructuras de C
- No hace falta definir las
- Se crean usando ()
- Sus elementos pueden ser heterogéneos
- Se accede a sus elementos igual que una lista



Ejemplos con Tuplas

```
>>> a = (1,2,3)
>>> a
(1, 2, 3)
>>> b = (a,["gul","linux"])
>>> b
((1, 2, 3), ['gul', 'linux'])
>>> a[2]
3
>>> b[1]
['gul', 'linux']
```

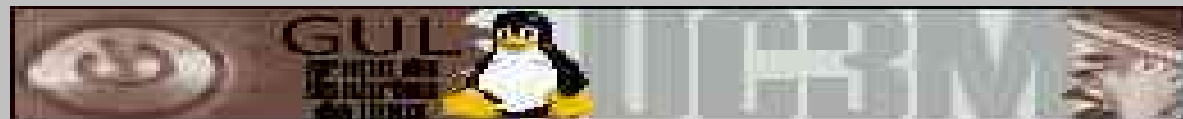
Diccionarios

- Como las tablas hash de Java
- Se identifican con { }
- Sus elementos están asociados a un clave
- Para acceder a un elemento:
diccionario[clave]
- Las claves deben ser únicas
- Los elementos complejos no pueden ser claves



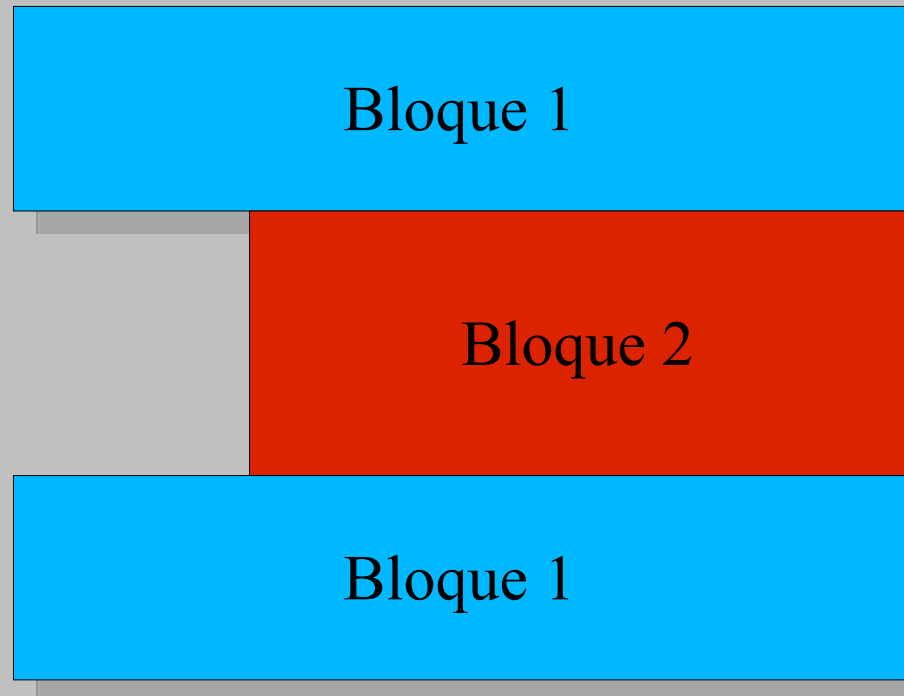
Diccionarios (II)

- Algunos métodos:
 - `has_key(x)`: Devuelve 1 si existe la clave
 - `items()`: Devuelve un lista con el contenido
 - `iteritems()`: Itera sobre la tupla (clave:elemento)
 - `iterkeys()`: Itera sobre las claves
 - `keys()`: Devuelve la lista de identificadores
- Más información `help(dict)`



Indentado

- El lenguaje es sensible al indentado
- Después de : hay un bloque



Condiciones

if **condición**:

 bloque si se cumple la condición

elif **condición2**:

 bloque si no se cumple la 1º condición y sí la 2º

else:

 bloque si no se cumple ninguna condición anterior

Bucles

while **condición**:

Lo que se hace en el bucle

- break sale del bucle
- continue pasa a la siguiente iteración



Iteraciones

for **variable** in **lista**:

Lo que está dentro de la iteración

- Se basa en las listas
- Función range



Rangos

- range devuelve una lista con un intervalo

```
>>> range(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> range(5,7)
```

```
[5, 6]
```



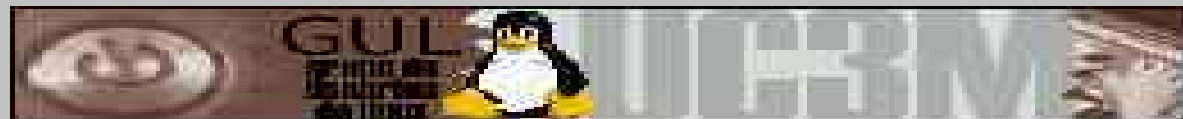
Definición de funciones

```
def función(argumento1, argumento2 ...):  
    'Documentación de la función'  
    Contenido de la función  
    [opcional: return salida]
```



Parámetros de las funciones

- Puede tener cero, uno, varios
- Puedo llamarla con menos parámetros
- Puede indicar los parámetros con los que llamo
- Puedo poner valores por defecto



Más claro con un ejemplo

```
def diHolaMundo (mensaje="Hola Mundo!", numVeces=1):  
    'Mi funcion de Hola mundo pesado de ejemplo.'  
    for i in range(numVeces):  
        print mensaje
```



Usando el Hola Mundo

```
>>> diHolaMundo()  
Hola Mundo!  
>>> diHolaMundo("Hola a todos")  
Hola a todos  
>>> diHolaMundo("Hola a todos",3)  
Hola a todos  
Hola a todos  
Hola a todos
```



Usando el Hola Mundo (II)

```
>>> diHolaMundo(3)
```

```
3
```

```
>>> diHolaMundo(numVeces = 5)
```

```
Hola Mundo!
```

```
Hola Mundo!
```

```
Hola Mundo!
```

```
Hola Mundo!
```

```
Hola Mundo!
```



Creando nuestra propia ayuda

```
>>> help(diHolaMundo)
Help on function diHolaMundo in module __main__:

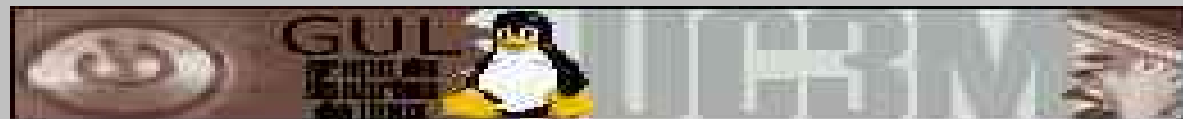
diHolaMundo(mensaje='Hola Mundo!', numVeces=1)
    Mi funcion de Hola mundo pesado de ejemplo.

>>> diHolaMundo.__doc__
'Mi funcion de Hola mundo pesado de ejemplo.'
```



Escribiendo programas

- En archivos .py
- Primera línea de un script de Unix
`#!/usr/bin/python`
- Ficheros .pyc son bibliotecas precompiladas
- Podemos llamarlo desde línea de comandos
 - `python programa.py`
 - `./programa.py`



Parámetros de entrada

- `sys.argv` es una lista con los parámetros
- Utilización:

```
import sys
nombre = sys.argv[0]
primer_param = sys.argv[1]
...
```

Consejos para los programas

- Es bueno dividir el código en funciones
- Podemos incluir código de prueba en un archivo
- Se ejecuta muy fácil con

```
if __name__ == '__main__':  
    testme()
```

Un poco sobre ficheros

- Crear un objeto fichero:
`f = open('nombre','modo')`
- Cerrar un fichero
`f.close()`
- Leer del fichero
`f.read()`, `f.readline()`, `f.readlines()`
- Guardar en un fichero
`f.write('texto')`, `f.writelines('texto')`



Trabajar con módulos

- Similar a las bibliotecas en C
- Agrupan archivos
- Se cargan con
`import modulo`
- Para llamar al contenido se antepone el nombre del módulo
`modulo.funcion()`
`modulo.variable`
- Similar a los espacios de nombres



Trabajar con módulos (II)

- Puede cargar todos los contenidos de un módulo al espacio de nombre actual

```
from modulo import *
```

- Similar al **using namespace** de C++
- También se pueden importar los ficheros del usuario

Algunos módulos básicos

- `sys`
 - Contiene funciones de sistema
 - `argv`, `exit`, `stderr`,...
- `OS`
 - Permite llamadas al sistema operativo
 - `popen`, `fork`, `chdir`, ...
- `os.path`
 - Trabaja con las rutas de los archivos
 - `isfile`, `exists`, `join`



Classes

```
class MiClase:  
    def setDato(self, dato):  
        self.Dato = dato  
    def display(self):  
        print self.Dato
```

Classes (II)

```
>>> x = MiClase()
>>> y = MiClase()
>>> x.setDato(4)
>>> y.setDato("hola")
>>> x.display()
4
>>> y.display()
hola
```

Herencia

```
class OtraClase (MiClase):  
    def display(self):  
        print 'El valor actual es ', self.Dato
```

```
>>> z = OtraClase()  
>>> z.setDato("herencia")  
>>> z.display()  
El valor actual es herencia
```

Sobrecarga de operadores

- Existen métodos especiales dentro de las clases:
 - `__init__`: Constructor
 - `__del__`: Destructor:
 - `__add__`: Operador de suma
 - `__or__`: operador O lógico
 - `__getitem__`: Indexación
 - `__setitem__`: Asignación indexada
 - `__getslice__`: seleccionar una parte
 - `__repr__`: Para salida por pantalla
 - `__len__`: Longitud
 - `__cmp__`: Comparación

Ejemplo con la suma

```
class MiClase2(MiClase):  
    def __init__(self, num=0):  
        self.Dato = num  
    def __add__(self, other):  
        return MiClase2(self.Dato + other.Dato)
```

```
>>> x = MiClase2(3)  
>>> y = MiClase2(6)  
>>> (x + y).display()
```

Excepciones

- Para manejo de errores, notificación de eventos, ...

```
try:  
    <Código...>  
except:  
    Nombre_excepción:  
        <Código para la  
        excepción>  
else:  
    <Código a ejecutar si no  
    se produce ninguna>
```

```
try:  
    <Código>  
finally:  
    <Código que se ejecuta  
    siempre>
```



Excepciones (II)

- raise permite lanzar una excepción de forma manual
- Puedes crear excepciones propias como una cadena de caracteres

```
>>> ex1 = 'Problema indeterminado'
```

```
>>> raise ex1
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in ?
```

```
Problema indeterminado
```

¿Qué se queda en el tintero?

- Funciones lambda
- Programación funcional
- Comprensión de listas
- Funcionamiento interno
- Jython
- Profile
- Psyco
- Extensión con C/C++
- Empotramiento en C/C++
- Mucho más...



Referencias:

- <http://www.python.org>
- Usos:
 - <http://www.pythonology.org/success&story=esr>
- Tutoriales:
 - <http://www.python.org/doc/current/tut/tut.html>
 - <http://es.diveintopython.org/>
 - <http://www.freenetpages.co.uk/hp/alan.gauld/spanish/ç>
- Lista de correo en castellano
 - <http://listas.aditel.org/listinfo/python-es>



Si alguien ha sobrevivido:
¿tiene preguntas?



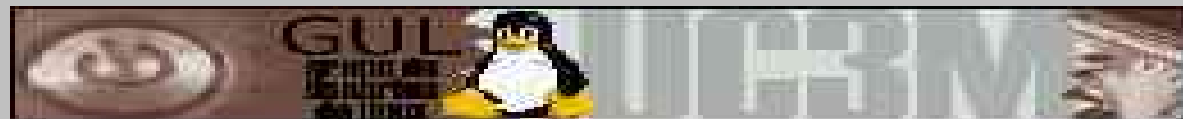
Expresiones lambda

- Permite crear funciones anónimas
lambda **operandos: expresión**

```
>>> def make_incrementor(n):  
    return lambda x: x + n  
>>> f = make_incrementor(42)  
>>> f(0)  
42  
>>> f(1)  
43
```

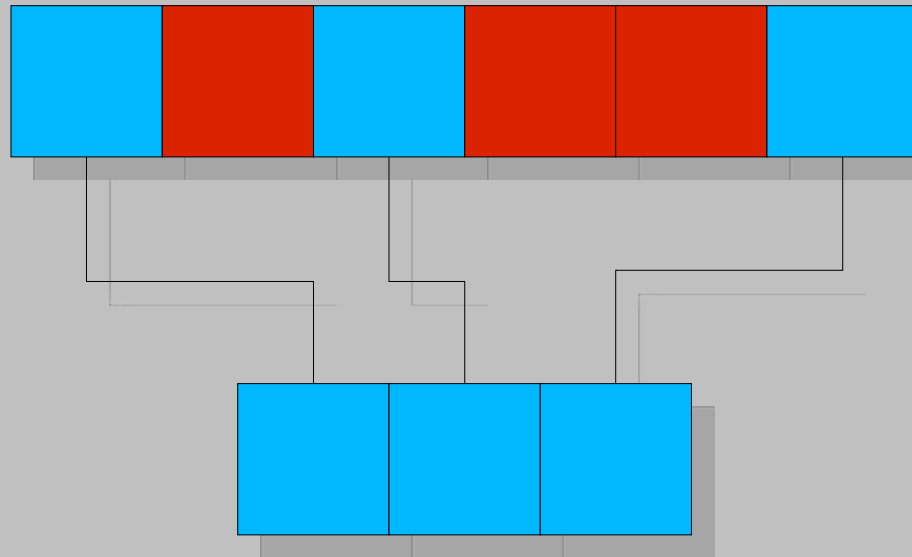
Más sobre programación funcional

- `filter(fun, seq)`
- `map (fun, seq)`
- `reduce (fun, seq)`



filter

- Aplica un filtro a una secuencia



```
>>> lista = ['azul','rojo','azul','rojo','rojo','azul']  
>>> filter(lambda x: x == 'azul', lista)  
['azul', 'azul', 'azul']
```

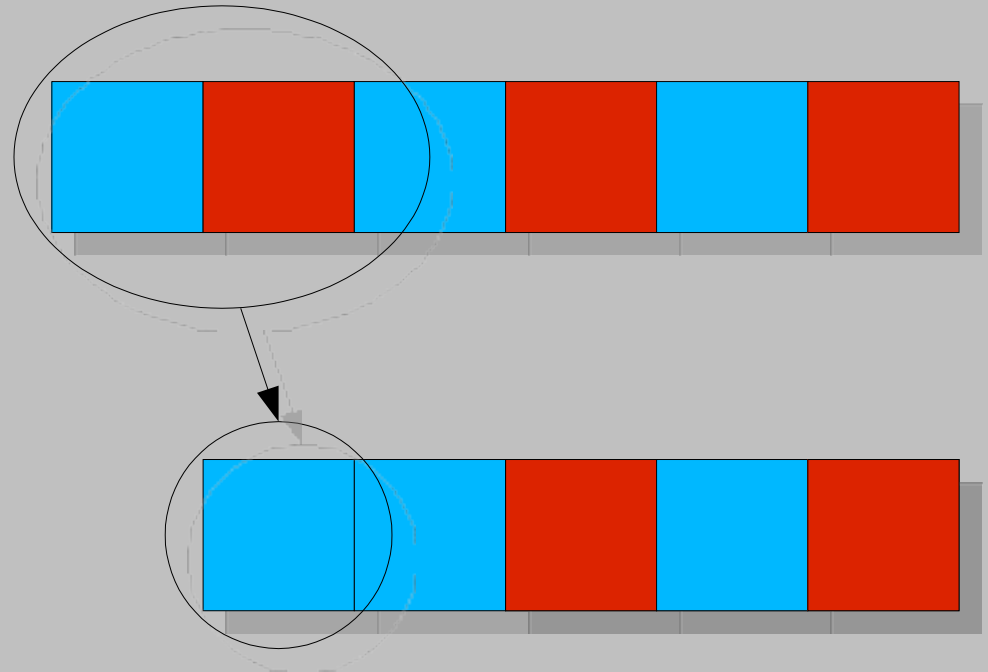
map

- Aplica una función a todas las posiciones

```
>>> a = map(ord,"pablo")
>>> a
[112, 97, 98, 108, 111]
>>> map(chr,a)
['p', 'a', 'b', 'l', 'o']
```

reduce

- Combina los elementos de una lista



```
>>> reduce(lambda x,y:x+y,map(chr,a))  
'pablo'
```

Más sobre listas

- List Comprehensions
- Permite crear un lista a partir de otra sin recurrir a filter, map y/o lambda

```
[x**3 for x in range(5)]
```

```
[x+y for x in vec1 for y in vec2]
```

```
[abs(x) for x in lista if x < 0]
```

Secretos de Python

- Cuando asignas una variable a otra
 - Puede copiarse el contenido
 - Pueden apuntar a la misma zona de memoria
- Depende del tipo de objeto
 - Simples: se copia
 - Complejos: Se enlaza a la misma zona de memoria



Secretos de Python (II)

```
>>> a = "hola"  
>>> b = a  
>>> a += " adios"  
>>> a  
'hola adios'  
>>> b  
'hola'
```

Secretos de Python (III)

```
>>> x = [1, 2]
>>> y = x
>>> x.append(3)
>>> y
[1, 2, 3]
>>> x
[1, 2, 3]
>>> x = [1]
>>> x
[1]
>>> y
[1, 2, 3]
```

Profile

- Antes de quejarte de la velocidad de Python, mira si tu código es bueno
- Módulo profile
- Uso:

```
import profile  
profile.run('funcion()')
```

- Devuelve el tiempo que ha tardado y cuantas veces ha sido llamada cada función

Últimos consejos

- Existen aplicaciones que ayudan en la programación
- IDLE (usa TKinter)
- PythonWin (for Windows)
- Boa
- Emacs mode



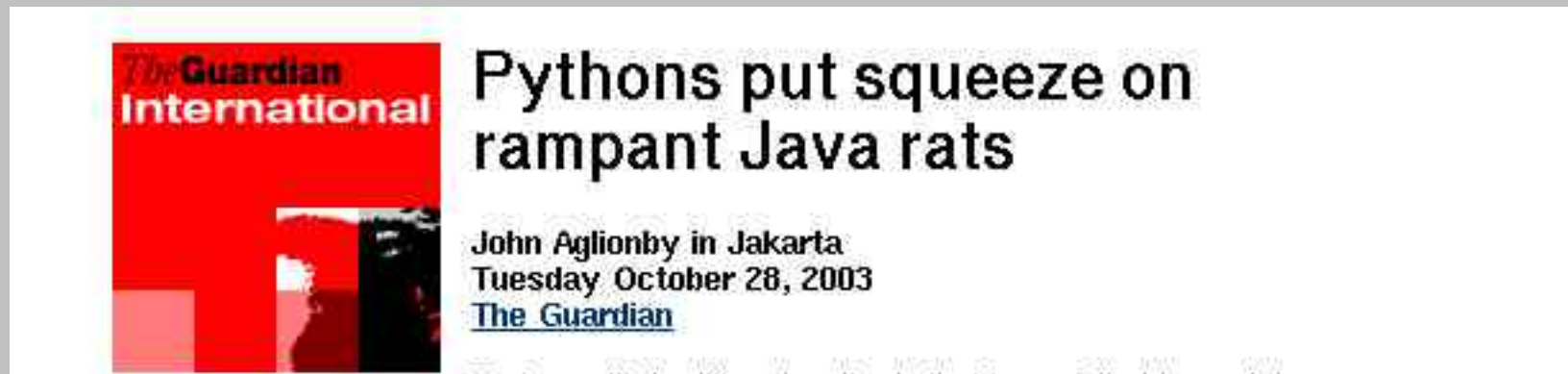
SWIG

- Permite crear interfaces para múltiples lenguajes de una biblioteca en C o C++
- Entre esos lenguajes está Python.
- Una forma muy cómoda de extender Python
- Entre otros lenguajes están: Java, C#, Ocaml, Perl, Php, Ruby...



Jython

- Python para Java
- El interprete se ejecuta sobre una máquina virtual de Java
- Lenguaje de Script para máquinas de Virtuales de Java



Ejemplos con Listas

```
>>> x = ['a', 'b', 'b', 'a', 'a', 'c']
>>> print x.count('a'), x.count('c'), x.count('gul')
3 1 0
>>> x.insert(2, 'gul')
>>> x.append('fin')
>>> x
['a', 'b', 'gul', 'b', 'a', 'a', 'c', 'fin']
>>> x.index('gul')
2
```

Pilas

- Pilas

```
>>> pila = [1,9,3]
```

```
>>> pila.append(8)
```

```
>>> pila.append(2)
```

```
>>> pila
```

```
[1, 9, 3, 8, 2]
```

```
>>> pila.pop()
```

```
2
```

```
>>> pila
```

```
[1, 9, 3, 8]
```

```
>>> pila.pop()
```

```
8
```

```
>>> pila.pop()
```

```
3
```

```
>>> pila
```

```
[1, 9]
```

Colas

```
>>> cola = [1, 2, 3]
>>> cola.append(4)
>>> cola
[1, 2, 3, 4]
>>> cola.pop(0)
1
>>> cola.append(5)
```

```
>>> cola
[2, 3, 4, 5]
>>> cola.pop(0)
2
>>> cola
[3, 4, 5]
```



Unas cuantas líneas con diccionarios

```
>>> tel = {'pablo': 8289, 'mario': 2454}
>>> tel['julio'] = 3459
>>> del tel['mario']
>>> tel
{'julio': 3459, 'pablo': 8289}
>>> tel['pablo']
8289
>>> tel.keys()
['julio', 'pablo']
>>> tel.has_key('julio')
1
```

