

Una pequeña introducción a Python

Antonio Arauzo Azofra

DECSAI

(DEpartment of Computer Science and Artificial Intelligence)

Ernesto Revilla



(Software Integrado para el Control de la Empresa)



GCubo: Grupo de usuarios de GNU/Linux de Granada

E.T.S.Ing. Informática (Univ. Granada)

Marzo 2.004

¿Qué es eso de Python?

- pitón -> suena a serpiente
- Monty Python -> grupo humorístico británico
- Guido van Rossum (su creador) buscaba nombre corto, único y misterioso

Python: lenguaje de programación

- - Interpretado
 - Orientado a objetos
 - Interactivo
- - Gran potencia
 - Sintaxis muy clara
- Lenguaje de programación (¿otro más?)
- Viva la diversidad!
- ¿El mejor?:
 - Depende para qué
 - Presenta muchas ventajas en muchas circunstancias

Facilidad de aprendizaje

- Sintaxis simple -> evita muchos problemas.
- Interprete -> respuesta inmediata.
- Indentado de código -> buenas prácticas prog.
- Universidades (EEUU, y Universitat Jaume I) ya lo usan como primer lenguaje para enseñar prog.
- Incluso niños de entre 6 y 8 años.
- Alguien con experiencia prog. lo aprende en unas horas -> rápida expansión.

Python se lleva bien con todos

- Es multiplataforma:

- GNU/Linux
- Windows
- Mac
- PalmOS
- WindowsCE
- RiscOS
- VxWorks
- OS/2
- AS/400
- PlayStation
- ...

- Extensible:

- C
- C++
- Fortran
- Java (Jython)
- ...

- Incrustable:

- Se puede incorporar en otras aplicaciones. (Ej. Mod. Inteligencia en juegos)

Ventajas

- Lenguaje de alto nivel, que permite centrarnos en nuestro problema.
- Desarrollo más rápido.
- Código muy legible, facilidad de mantenimiento.
- Potente, permite expresar mucho.

Software Libre

- Python es Software Libre (Debian Free Software Guidelines)



- Interprete, librerías y documentación.

- VENTAJAS:

- Desarrollo abierto.
- No va a ser abandonado, seguirá disponible grat.
- Espíritu software libre -> hay muchísimos códigos fuentes de ejemplo disponibles.
 - Para aprender de ellos o usarlos.

Librerías

- Gran cantidad de librerías para hacer de todo!!

- Acceso a ficheros, manejo de cadenas
- Servicios web
- Retoque de imágenes
- Multimedia
- Edición tags MP3
- Aprendizaje automático
- Criptografía
- Interfaces gráficos
- XML
- Creación de PDFs
- ...

Calculo científico
Acceso a Bases de Datos
Estadística
Analizadores léxicos y
sintácticos
Desarrollo de juegos
...

Documentación y soporte

- <http://www.python.org>
- Tutorial en español:
<http://es.tldp.org/Tutoriales/Python/Tutorial-Python/>
- “Introducción a la Programación con Python” Andrés Marzal, Isabel Gracia (<http://marmota.act.uji.es/MTP/teoria.shtml>)
- Documentación en español: <http://www.hispapython.org/>

Lenguaje “inicial” y “final”

Hay lenguajes diseñados para ser fáciles de aprender.

Hay lenguajes enfocados al desarrollo profesional de aplicaciones.

Python es ambas cosas

Soporta diferentes paradigmas de programación
(funcional, orientada a objetos)

“Python amplifys your mind”

Eric Raymond

Author, The Cathedral and the Bazar

Soluciones profesionales ¿quien lo usa?



“Python is our secret weapon”

Paul Everitt, Co-Fundador de Digital Creations



"The Phantom Menace", "The Mummy Returns"



“Python has been an important part of Google since the beginning”

Peter Norvig, director dpto. calidad de busqueda de Google, Inc



Orange project
(Data mining)



Blender 3D



RealNetworks



NATIONAL AERONAUTICS
AND SPACE ADMINISTRATION

- Y muchos, muchos más....

... ¿más razones? ...

- Porque nos gusta y ...
- ... nos gusta enseñar lo que nos gusta y
- ... estamos un poco hartos del bombardeo comercial

- **Ciclo Python:**

- Introducción
- GUI
- Prog. web
- Python avanzado



GCubo

<http://www.gcubo.org>

No nos aburras, empecemos ya...

Degradando el intérprete a una calculadora...

```
$ python
>>> 1+1
2           # sabe sumar!
>>> 3**3
27          # hacha!
>>> sin(pi/2)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'pi' is not defined
```

Bueno, no tan listo!

Calculadora, 2ª toma

```
>>> from math import sin, pi
>>> sin(pi/2)
1.0
```

Bueno, eso no ha estado mal, pero lo del import...
¿cómo usamos la memoria?

```
>>> a=2**16
>>> a
65536
>>> a*a
4294967296L
```

..., ¿y esto sirve para programar?

Vamos a probarlo con el famoso programa 'Hola mundo':

```
>>> print 'Hola mundo'  
Hola mundo
```

Ya está! ¿Cómo que ya está? Hola mundo en C:

```
#include <stdio.h>  
int main(void){  
    printf("Hola mundo\n");  
    return 0;  
}
```

además: gcc -o hola hola.c

Hola mundo en Java....????? Que alguien nos ayude...

Un pequeño programa

Escribimos una pequeña función con el intérprete interactivo:

```
>>> def factorial(n):  
...     if n<=0:  
...         return 1  
...     return n*factorial(n-1)  
...  
>>> factorial(3)  
6  
>>> factorial(10)  
3628800
```

No está nada mal. Pero, cuidado con los espacios y los tabuladores.

La indentación incorrecta produce error de sintaxis! Ese efecto, molesto, evita que cometamos errores semánticos (como en C, un else sin {} seguida por otra instrucción, etc).

Más sobre la indentación...

- Hace los programas más legibles
- Evita errores semánticos
- Indentación automática está soportado por muchos editores de programas, como IDLE, Emacs, PythonWin, Vim, etc.
- Período de adaptación: 2 días
- Sí hay problema con tabuladores y espacios: son diferentes. Por eso, los editores permiten configurarse que al pulsar el tabulador, se inserta un número predefinido de espacios. (En algunos editores puede hacerse visible estos caracteres.)

Programa no interactivo

Con la misma función factorial, escribimos un programa que imprime el factorial de un número introducido por el usuario y lo llamaremos factorial.py:

```
numero = int(raw_input('Introduce un numero: '))

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

print "Factorial de",numero,"es",factorial(numero)
```

Ejecución:

```
$ python factorial.py
Introduce un numero: 4
Factorial de 4 es 24
$
```

Variables

Antes de pelearnos con cosas más interesantes, vamos a ver cómo trata Python las variables.

¿Qué es una variable?

Una variable es un identificador ligado a un valor.

¿Qué es una asignación?

El hecho de ligar un identificador a un valor. Ejemplo:

```
a = 10
```

```
a = b = 20
```

Más sobre asignaciones y variables

Una asignación nunca crea un objeto!!! (Al menos no en Python.)

```
a=1
```

El valor 1 ya existía (internamente).

```
Lista= [ ]
```

Asigna a la variable Lista una lista vacía.

Probamos en Python:

```
>>> a
```

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

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

```
NameError: name 'a' is not defined
```

```
>>> a = 20
```

```
>>> a ** 2
```

```
400
```

Variables locales, variables globales

Primero el ejemplo:

```
>>> a=10
>>> def func():
...     a = a + 20
...
>>> func()
>>> a
10
```

Y eso por qué? Cuando se hacen asignaciones en funciones, siempre se usan variables locales, a menos que se use `global`.

```
>>> def func():
...     global a
...     a += 20
...
>>> func()
>>> a
30
```

Los espacios de nombres

En casos normales, Python usa 3 espacios (ámbitos) de nombres:

- local, es decir, de la misma función
- global, a nivel de módulo
- nombre internos (abs, execfile, etc.)

Si a la hora de buscar un identificador, Python no lo encuentra en el espacio local, lo busca en el global y finalmente en el de los nombres internos. Pero, las asignaciones se hacen siempre en el ámbito local.

Si trabajamos fuera de funciones, el ámbito local y el global es el mismo.

Nota: también hay espacios de nombres anidados que no se verán en esta introducción

Importar referencia desde otros módulos

- La instrucción **import** hace disponible un identificador (variable) en el módulo actual:

```
from math import sin
```

Ahora el identificador, originalmente parte del módulo math, está disponible en el módulo actual.

- Otras posibilidades con **import**:

```
import math
```

Ahora podemos acceder a las funciones a través del nombre del módulo:

```
print math.sin(math.pi / 2)
```

- O:

```
from math import sin as sen  
print sen(4)
```

Cosas útiles

Búsqueda y reemplazamiento de un texto en un archivo, strrepl.py:

```
1: import sys
2: if len(sys.argv)<3:
3:     print "Uso:",sys.argv[0], '"texto buscar"',\
4:         '"texto reempl"', "[<fichero1] [>fichero2]"
5: else:
6:     buscar, reemplazar = sys.argv[1:3]
7:     print sys.stdin.read().replace(buscar,reemplazar)
```

Interpretamos el programa:

1: importar del módulo sys para acceder tanto a los argumentos pasados al programa como al archivo 'stdin' (entrada estándar).

2: Si no se ha especificado suficientes argumentos

3,4: imprimir una pequeña ayuda

5: de lo contrario

6: sys.argv[1:3] devuelve una tupla con los argumentos 1 y 2, asignar respectivamente a buscar y reemplazar

7: leer desde entrada estándar todo, reemplazar texto e imprimir resultado.

Tipos y estructuras de datos 1: cadenas

Cadenas de caracteres son secuencias y se pueden indexar:

```
>>> a="Hola mundo!"
```

```
>>> a[0] # el primer carácter
```

```
H
```

```
>>> a[:4] # los cuatro primeros
```

```
Hola
```

```
>>> a[-1] # el último carácter
```

```
!
```

```
>>> a[-2:] # los últimos dos caracteres
```

```
o!
```

```
>>> a[:-1] # todo menos el último carácter
```

```
Hola mundo
```

No se puede asignar a cadenas (cadenas son inmutables), pero se pueden crear nuevas.

```
>>> a[1]="a"
```

```
...
```

```
TypeError: object doesn't support item assignment
```

Tipos y estructuras de datos: listas

Las listas también son secuencias:

```
>>> a=[1,2,3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> a[0]
1
>>> a[-1]
4
>>> a[1:3]
[2, 3]
>>> x, y = a[1:3]
>>> print "x es", x, ", y es", y
x es 2, y es 3
>>> a.remove(3); a
[1, 2, 4]
>>> a[2:2]=[3,3.5, 3.75]; a
[1, 2, 3, 3.5, 3.75, 4]
```

Tipos y estructuras de datos: listas (2)

Más operaciones sobre listas (secuencias mutables):

```
>>> a=[2,3]
>>> a.insert(0,1); a
[1, 2, 3]
>>> len(a)
3
>>> a.reverse(); a
[3, 2, 1]
>>> a.sort(); a
[1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
>>> a.pop(0)
1
>>> a
[2]
```

Tipos y estructuras de datos: listas (3)

Las listas pueden contener elementos arbitrarios, es decir:

- Pueden contener datos heterogéneos, mezclar números, cadenas y otros tipos complejos:

```
lista=[1, "esto", 2, "es", 3, "python"]
```

- pueden ser anidadas:

```
milista=["java", lista]
```

- pueden ser recursivas:

```
lista.append(lista)
```

- listas declarativas:

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

Tipos y estructuras de datos: diccionarios

Un diccionario es un conjunto de elementos clave, valor.

```
>>> d={ }
>>> cadena="Hola, esto es una prueba."
>>> for letra in cadena:
...     try:
...         d[letra] += 1
...     except KeyError:
...         d[letra] = 1
...
>>> print "frecuencias:", d
>>> d['z']=10
>>> del d['z']
>>> d['z']
```

Traceback (most recent call last):

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

KeyError: z

Otros tipos de datos y estructuras

- None, 'nada', es como nil en Pascal o NULL en SQL
- Números enteros (int), números largos (long), números reales (float), números complejos (complex)
- cadenas unicode, string sólo soporta 128 caracteres: u'Python es fantástico'; unicode('¡qué maravilla!','latin1')
- Tuplas: como lista, pero no mutable, ej: (1,'hola',3). Listas y tuplas permiten packing y unpacking:

```
res1, res2 = 2**30, 2**31
```

Sentencias de control 1

Ya hemos visto algunas sentencias en ejemplos anteriores. Revisemos las más importantes:

```
if condicion:
    sentencias
elif condicion:
    sentencias
else:
    sentencias
```

While: Secuencia fibonacci:

```
>>> def fib(n):
...     a,b=0,1
...     while b <= n:
...         print b,
...         a, b = b, a+b
...
>>> fib(100)
1 1 2 3 5 8 13 21 34 55 89
```

Sentencias de control 2

Un ejemplo de `for`:

```
for palabra in ['hola', 'esto', 'es', 'python']:  
    print palabra, len(palabra)
```

Bucle con números (n veces):

```
for i in range(10):  
    print i, i*i
```

Podemos usar `break` y `continue` para seguir o terminar el bucle:

```
for i in range(100):  
    if i % 2 == 0: continue  
    c=str(i*i)  
    print i,c  
    if len(c)>3 and c[0]==c[-1]: break  
else:  
    print "No encuentro ningun cuadro con long.>3 con  
    "\  
        "el mismo primer y último dígito."
```

No hacer nada: `pass`

```
for i in range(10000): pass
```

Definición de funciones

```
def fact2(numero):  
    res=1  
    while numero>=1:  
        res *= numero  
        numero -= 1  
    return res
```

```
>>> fact2(10)
```

```
3628800
```

```
>>> fact2
```

```
<function fact2 at 0x837b02c>
```

```
>>> f=fact2
```

```
>>> f(11)
```

```
39916800
```

```
def preguntar(que, defecto="S"):  
    res=raw_input(que+" [" +defecto+"]"):  
    return res or defecto  
>>> preguntar("¿Python es interesante?")
```

Un poquito más sobre funciones

Argumentos nombrados (keyword arguments):

```
def argtest (arg1, arg2, arg3):  
    print "arg1:",arg1," , arg2:",arg2," , arg3:", arg3  
>>> argtest(1,2,3)  
>>> argtest(arg3=10, arg2=20, arg1=50)
```

Listas de argumentos:

```
def sumar(*args):  
    res=0  
    for arg in args: res += arg  
    return res
```

```
def verargs(*args, **kwargs):  
    print "args:",args  
    print "kwargs:", kwargs  
>>> verargs('a','b',1,2,pepe=3, juan=4)
```

Clases 1

En Python es muy fácil escribir clases, pero es más importante comprender bien los conceptos, para escribir código correcto.

```
class A:
    "Esta es mi clase A"
    variableClase=5
    def __init__(self):
        self.variableInstancia=7
    def info(self):
        print "Variables: clase %s, instancia %s" % \
            (self.variableClase, self.variableInstancia)
    def cambiaVariableClase(self):
        print self.variableClase=8
>>> a=A(); a.hola="Hola"; a.info()
Variables: clase 5, instancia 7
>>> a.cambiaVariableClase(); a.info()
Variables: clase 8, instancia 7
>>> b=A(); b.info()
Variables: clase 5, instancia 7
```

Clases 2: Conceptos

Los 'objetos' clase permiten dos tipos de operaciones:

- Instanciación
- Referencia a atributos. Hay dos tipos de atributos
 - Atributos de datos
 - Métodos (funciones asociadas a clases o instancias)

```
>>> class A:
...     def cuadrado(self, i):
...         return i*i
>>> A.mivariable=10    # referencia
>>> a=A()              # Asignación
>>> A.__dict__        # ver lo que tiene la clase
```

Los objetos 'instancia' sólo admiten la referencia de atributos:

```
>>> a.mivariable=50
>>> a.cuadrado(10)
100
>>> A.cuadrado(a,10) # es lo mismo q llamada
anterior!
>>> def saluda(self, nombre): print "Hola %s" %
nombre
>>> A.saluda=saluda d# esto es posible!!
```

Clases 2: Herencia

Python soporta la herencia múltiple. Las clases que abstraen una funcionalidad común y que mezclamos con nuestras clases los llamamos Mixin.

```
class PrintInfoMixin:  
    def printInfo(self):  
        print self.__dict__
```

```
class A:  
    def cuadrado(self, i):  
        return i*i
```

```
class B(A, PrintInfoMixin):  
    pass
```

```
>>> b=B()
```

```
>>> b.printInfo()
```

Clases 2: Herencia 2

Heredando, los métodos pueden:

- reemplazar una funcionalidad
- ampliar una funcionalidad

```
class A:
    def func(self, *args, **kwargs):
        print "A.func"
    def func2(self):
        print A.func2
```

```
class B(A):
    def func(self, *args, **kwargs):
        print "hacer cosas antes"
        A.func(self, *args, **kwargs)
        print "hacer cosas después"
        self.func2()
```

¿Qué más?

- Instalar Python en casa. En vez de escribir .sh, .bat o lo que sea, escribirlo en Python!
- Si no todavía no tienes mucha experiencia o conocimiento de programación, leer el libro de Andrés Marzal. Es muy bueno. (información en www.hispapython.org)
- Leer el tutorial de Python, es muy bueno. Además aclara conceptos aplicables también a otros lenguajes.
- Apúntate a la lista python-es. (<http://listas.aditel.org/listinfo/python-es>)
- ¡¡¡Apúntate a Gcubo!!! (www.gcubo.org) o entra en el chat en [#gcubo](http://irc.freenode.org)
- Si quieres hacer cosas Web complejas, mira Zope (www.zope.org)
- Ayúdanos si te sobra el tiempo.