

# Introducción a



Taller de Python – UNLu

Marcelo Fernández

2007

Presentación Original: Facundo Batista

Arte gráfico: Diana Batista



# Indice

1. ¿Qué es Python?
2. Corriendo e interpretando
3. Tipos de datos
4. Controles de flujo
5. Encapsulando código
6. Tres detalles

# ¿Qué es Python?

- a) Algunas características
- b) Propiedades del lenguaje
- c) Biblioteca estándar (con las pilas puestas)
- d) Comunidad de Desarrolladores: PyAr

# Algunas características

- Gratis **Y** Libre
  - × Y Open Source, todo por el mismo precio: **cero**
- Maduro (+14 años)
  - × Diseño elegante y **robusto**
  - × Pero **evoluciona** (Python 3k en 2008)
- **Fácil** de aprender
  - × Se lee como pseudo-código
  - × **Sintaxis** sencilla, lenguaje muy **ortogonal** (dada su filosofía)
- Extremadamente **portable**
  - × Unix, Windows, Mac, BeOS, Win/CE
  - × DOS, OS/2, Amiga, VMS, Cray...

# Propiedades del lenguaje

- Compila a bytecode interpretado
  - × La compilación es **implícita y automática**
  - × Tipado **dinámico**, pero **fuerte**
- **Multi-paradigma**
  - × Todo son objetos
  - × Pero puede usarse de manera procedural
- Módulos, clases, funciones, generadores
- Viene con las **baterías incluidas**
  - × Extensa biblioteca estándar
  - × Clave en la **productividad** de Python

# Más propiedades

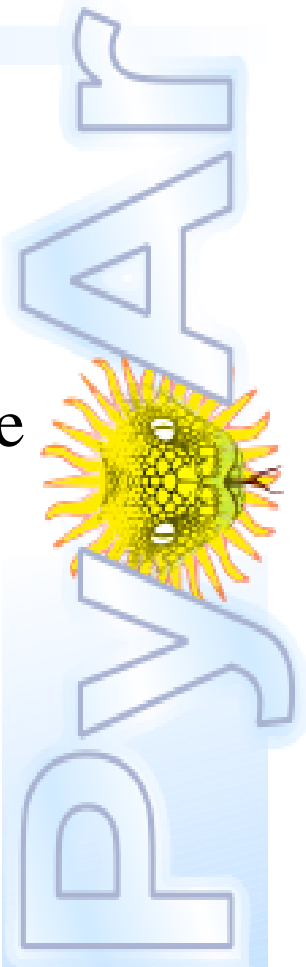
- Manejo moderno de **errores**
  - × Por **excepciones**
  - × Muy útil **detalle de error**
- Tipos de datos de **alto nivel**
  - × **Enteros sin límites**, strings, flotantes, complejos
  - × Listas, **diccionarios**, conjuntos
- Intérprete interactivo
  - × Clave en el **bajo conteo de bugs**
  - × **Acelera** sorprendentemente el **tiempo de desarrollo**
  - × Permite **explorar, probar** e incluso ver la **documentación**

# Las baterías incluidas

- La Biblioteca Estándar ayuda con...
  - × Servicios del sistema, fecha y hora, subprocesses, sockets, internacionalización y localización, base de datos, threads, formatos zip, bzip2, gzip, tar, expresiones regulares, XML (DOM y SAX), Unicode, SGML, HTML, XHTML, XML-RPC (cliente y servidor), email, manejo asíncrono de sockets, clientes HTTP, FTP, SMTP, NNTP, POP3, IMAP4, servidores HTTP, SMTP, herramientas MIME, interfaz con el garbage collector, serializador y deserializador de objetos, debugger, profiler, random, curses, logging, compilador, decompilador, CSV, análisis lexicográfico, interfaz gráfica incorporada, matemática real y compleja, criptografía (MD5 y SHA), introspección, unit testing, doc testing, etc., etc...

# Python Argentina

- ¿Quiénes somos?
  - × Grupo de **entusiastas** de Python
  - × Referencia para la aplicación y **difusión** del lenguaje
- ¿Cómo **participar**?
  - × Suscribiéndose a la **Lista de Correo** (somos +250)
  - × **Asistiendo** a las **reuniones** y eventos
  - × Más info en la página: [www.python.com.ar](http://www.python.com.ar)
- **PyAr es federal**
  - × Se ~~pueden~~ deben organizar **reuniones en otras provincias**
  - × No hay que pedir permiso, **sólo coordinarlas**



# Corriendo e interpretando

- Menos charla y más acción
  - × Python es interpretado
  - × No hace falta compilar
  - × Ciclo corto de pruebas
  - × Y encima tenemos el **Intérprete Interactivo**
- **Go! Go! Go!**
  - × Acá es donde vamos **a la realidad**, :)
  - × ¡Juro que antes andaba!
  - × **Go!**

# Tipos de datos

- Haciendo números, y más números
- Cadenas, y como accederlas
- Listas, listas, y muchas listas
- Conjuntos
- Diccionarios, ¡diccionarios!

# Haciendo números

## Enteros

```
>>> 2+2
```

```
4
```

```
>>> (50 - 5*6) / 4
```

```
5
```

```
>>> 7 / 3
```

```
2
```

```
>>> 7 % 3
```

```
1
```

```
>>> 23098742098472039 * 120894739  
2792516397223089453702821
```

## Floats

```
>>> 3 * 3.75 / 1.5
```

```
7.5
```

```
>>> 7 / 2.3
```

```
3.0434782608695654
```

# Más números

## Complejos

```
>>> 2 + 3j
(2+3j)
>>> (2+3j * 17) ** (2+5j)
(-0.91258832667469336-0.82498333629811516j)
```

## Recortando los decimales

```
>>> int(12.3)
12
>>> round(2.7526)
3.0
>>> round(2.7526, 2)
2.75
```

## Convirtiendo

```
>>> str(5)
'5'
>>> float(37.25)
37.25
```

# Cadenas

## Comillas, apóstrofes, triples

```
>>> 'Una cadena es una secuencia de caracteres'
'Una cadena es una secuencia de caracteres'
>>> "Ella dijo: 'si'"
"Ella dijo: 'si'"
>>> """Una linea
... y la otra"""
'Una linea\ny la otra'
```

## Algunas operaciones

```
>>> "Hola" + " mundo"
'Hola mundo'
>>> "Eco " * 4
'Eco Eco Eco Eco '
>>> "    Hola mundo    ".strip()
'Hola mundo'
>>> len("Hola mundo")
10
```

# Accediendo a las cadenas

## Por posición

```
>>> saludo = 'Hola mundo'
>>> saludo[0]
'H'
>>> saludo[3]
'a'
>>> saludo[-2]
'd'
```

## Rebanando

```
>>> saludo[2:5]
'la '
>>> saludo[2:8]
'la mun'
>>> saludo[:4]
'Hola'
>>> saludo[-2:]
'do'
```

# Listas

**Corchetes**, varios tipos de elementos

```
>>> a = ['harina', 100, 'huevos', 'manteca']
>>> a
['harina', 100, 'huevos', 'manteca']
```

Accedemos como cualquier **secuencia**

```
>>> a[0]
'harina'
>>> a[-2:]
['huevos', 'manteca']
```

Concatenamos, reemplazamos

```
>>> a + ['oro', 9]
['harina', 100, 'huevos', 'manteca', 'oro', 9]
>>> a[0] = "sal"
>>> a
['sal', 100, 'huevos', 'manteca']
```

# Y dale con las listas

Pueden tener **otras listas**, ¡y a sí mismas!

```
>>> a
['sal', 100, 'huevos', 'manteca']
>>> a[1] = ["Hola", 7]
>>> a
['sal', ['Hola', 7], 'huevos', 'manteca']
>>> a[2] = a
>>> a
['sal', ['Hola', 7], [...], 'manteca']
```

**Borramos** elementos

```
>>> del a[-1]
>>> a
['sal', ['Hola', 7], [...]]
>>> del a[1:]
>>> a
['sal']
```

# Listas, listas, listas

Tenemos otros **métodos**

```
>>> b = [2, "H", 3.5, "H"]
>>> b
[2, 'H', 3.5, 'H']
>>> b.append("Go!")
>>> b
[2, 'H', 3.5, 'H', 'Go!']
>>> b.pop()
'Go!'
>>> b.index(3.5)
2
>>> b.count("H")
2
>>> b.sort()
>>> b
[2, 3.5, 'H', 'H']
```

# Conjuntos

## Definimos con set()

```
>>> juego = set("typus pocus")
>>> juego
set([' ', 'c', 'o', 'p', 's', 'u', 't', 'y'])
>>> hechizo = set(["h", "o", "c", "u", "s", " "])
>>> hechizo.update(set("pocus"))
>>> hechizo
set([' ', 'c', 'h', 'o', 'p', 's', 'u'])
```

## Operamos

```
>>> hechizo - juego
set(['h'])
>>> hechizo & juego
set([' ', 'c', 'o', 'p', 's', 'u'])
>>> hechizo.remove("h")
>>> hechizo.add("Merlin")
>>> hechizo
set([' ', 'c', 'Merlin', 'o', 'p', 's', 'u'])
```

# Diccionarios

## Definimos con llaves

```
>>> dias = {"enero": 31, "junio": 30, "julio": 30}
>>> dias
{'julio': 30, 'enero': 31, 'junio': 30}
>>> dias["enero"]
31
>>> dias["agosto"] = 31
>>> dias["julio"] = 31
>>> dias
{'julio': 31, 'enero': 31, 'junio': 30, 'agosto': 31}
>>> cualquiercosa = {34: [2,3], (2, 3): {3: 4}}
```

## Borrando

```
>>> del dias["julio"]
>>> dias
{'enero': 31, 'junio': 30, 'agosto': 31}
```

# Más diccionarios

## Viendo qué hay

```
>>> "marzo" in dias
False
>>> dias.keys()
['enero', 'junio', 'agosto']
>>> dias.values()
[31, 30, 31]
```

## Otros métodos

```
>>> dias.get("agosto", "No tenemos ese mes")
31
>>> dias.get("mayo", "No tenemos ese mes")
'No tenemos ese mes'
>>> dias.pop("agosto")
31
>>> dias
{'enero': 31, 'junio': 30}
```

# Controles de flujo

- ¿Que pasaría **si...**
- ...**por cada** uno de esos hacemos algo...
- ...**mientras** esperamos otra cosa?
- Eso sí, ¡hasta que se rompa algo!

# Si tal cosa o la otra

## Estructura del `if`

```
a = ...  
if a == 0:  
    print "Ojo con el valor de b"  
    b = 0  
elif a > 100 or a < 0:  
    print "Error en el valor de a"  
    b = 0  
else:  
    b = c / a  
print b
```

## Eso que hay **después** del `if`:

- `or`, `and`, `not`
- `<` `>` `==` `!=` `in` `is`
- **Todo** evalúa a Falso o Verdadero

# Por cada uno

## Estructura del `for`

```
>>> bichos = ["pulgas", "piojos", "cucarachas"]
>>> for bich in bichos:
...     print "Mata-" + bich
...
Mata-pulgas
Mata-piojos
Mata-cucarachas
```

## Si queremos la `secuencia de números`

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> for i in range(5):
...     print i**2
...
0
1
16
```

# Mientras tanto...

## Estructura del `while`

```
>>> a = 0
>>> while a<1000:
...     print a**5
...     a += 3
0
243
7776
...
980159361278976
995009990004999
```

Al igual que el `for`, tiene:

- `continue`: Vuelve a empezar al principio del loop
- `break`: Corta el loop y sale
- `else`: Lo ejecuta si no cortamos con el `break`

# Excepciones

Tenemos algo que **funciona...**

```
>>> def divide(a, b):  
...     return a / b  
...  
>>> divide(14, 2)  
7
```

...pero **no siempre...**

```
>>> divide(14, 0)
```

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

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

```
  File "<stdin>", line 2, in divide
```

```
ZeroDivisionError: integer division or modulo by zero
```

# Manejando lo excepcional

Podemos **capturar** las excepciones

```
>>> try:
...     print divide(14, 0)
... except:
...     print "epa"
...
epa
```

**Además:** try/else, try/finally, ¡y sus **combinaciones!**

Y podemos **generarlas**

```
>>> def algo(x):
...     if x > 1000:
...         raise ValueError("El x es feo!")
...     # procesamos x
...     # ...
```

# Encapsulando código

- **Funciones** y más funciones
- Clases, o como tratar de **modelar la realidad**
- **Módulos** y paquetes

# Funciones

## Estructura básica

```
>>> def alcuadrado(n):  
...     "Devuelve el cuadrado de un número."  
...     res = n ** 2  
...     return res  
...  
>>> alcuadrado(3)  
9
```

## Las funciones son **objetos**

```
>>> alcuadrado  
<function alcuadrado at 0xb7c30b54>  
>>> f = alcuadrado  
>>> f(5)  
25  
>>> help(alcuadrado)  
alcuadrado(n)  
    Devuelve el cuadrado de un número.
```

# Más funciones

Tengo mucha **flexibilidad** con los **argumentos**

```
>>> def func(a, b=0, c=7):  
...     return a, b, c  
...
```

```
>>> func(1)
```

```
(1, 0, 7)
```

```
>>> func(1, 3)
```

```
(1, 3, 7)
```

```
>>> func(1, 3, 9)
```

```
(1, 3, 9)
```

```
>>> func(1, c=9)
```

```
(1, 0, 9)
```

```
>>> func(b=2, a=-3)
```

```
(-3, 2, 7)
```

# Clases

## Armando una clase

```
>>> class MiClase:
...     x = 3
...     def f(self):
...         return 'Hola mundo'
...
>>> c = MiClase()
>>> c.x
3
>>> c.f()
'Hola mundo'
```

## Heredando

```
>>> class MiClase(ClasePadre):
>>> class MiClase(ClasePadre, ClaseTio):
```

# Otra clase sobre clases

```
>>> class Posicion:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...     def distancia(self):
...         dist = math.sqrt(self.x**2 + self.y**2)
...         return dist
...
>>>
>>> p1 = Posicion(3, 4)
>>> p1.x
3
>>> p1.dist()
5.0
>>> p2 = Posicion(7, 9)
>>> p2.y
9
>>> p1.y
4
```

# El módulo más paquete

- Módulos

- × Funciones, o clases, o **lo que sea** en un archivo
- × Es un **.py normal**, sólo que lo importamos y usamos
- × Fácil, rápido, **funciona**

Tengo un pos.py, con la clase de la filmina anterior:

```
>>> import pos
>>> p = pos.Posicion(2, 3)
>>> p.x
2
```

- Paquetes

- × Cuando tenemos **muchos módulos** juntos
- × Usamos **directorios**, e incluso subdirectorios

# Tres detalles

- List comprehensions
- Generadores
- Espacios de nombres

# Entendiendo de listas

## List comprehensions

```
>>> vec = [3, 7, 12, 0, 3, -13, 45]
>>> [x**2 for x in vec]
[9, 49, 144, 0, 9, 169, 2025]
>>> [x**2 for x in vec if x <= 7]
[9, 49, 0, 9, 169]
```

## Son extremadamente útiles

```
>>> sum([x**2 for x in range(1000)])
332833500
>>> len([x for x in range(1000) if (x**2)%2 == 0])
500
```

# Generadores

Ejemplo: Función que nos devuelve una cantidad de algos

```
>>> def fibonacci(limite):
...     valores = []
...     a, b = 0, 1
...     while b < limite:
...         valores.append(b)
...         a, b = b, a+b
...     return valores
>>> fibonacci
<function fibonacci at 0xb7c30b54>
>>> fibonacci(10)
[1, 1, 2, 3, 5, 8]

>>> t = 0
>>> for i in fibonacci(10): t += i
>>> t
20
>>> for i in fibonacci(999999999999999999999999): # ouch!
```

# Seguimos generando

Somos vagos, vamos devolviendo **valor por valor**

```
>>> def fibonacci(limite):
...     a, b = 0, 1
...     while b < limite:
...         yield b
...         a, b = b, a+b
...
>>> fibonacci
<function fibonacci at 0xb7c30bfc>
>>> fibonacci(10)
<generator object at 0xb7c294ac>

>>> t = 0
>>> for i in fibonacci(9999999999999999999999):
...     t += i
>>> t
1779979416004714188
```

# Una gran idea

- Hay varios **espacios de nombres**
  - × Básicos: **local** y **global**
  - × Los tienen las funciones, clases, módulos

¡El **mismo ejemplo** que antes!

```
>>> import pos
>>> p = pos.Posicion(2, 3)
>>> p.x
2
```

- Más **útiles** de lo que parecen
  - × **Simplifican** la **estructura** a mentalizar
  - × Prolijidad, **legibilidad**, traceabilidad
  - × Pruébenlos, los van a extrañar cuando no los tengan

¿Preguntas?  
¿Sugerencias?

Espero que lo hayan  
disfrutado tanto  
como yo, :)

¡Muchas gracias!

Facundo Batista

facundo@taniquetil.com.ar

www.taniquetil.com.ar