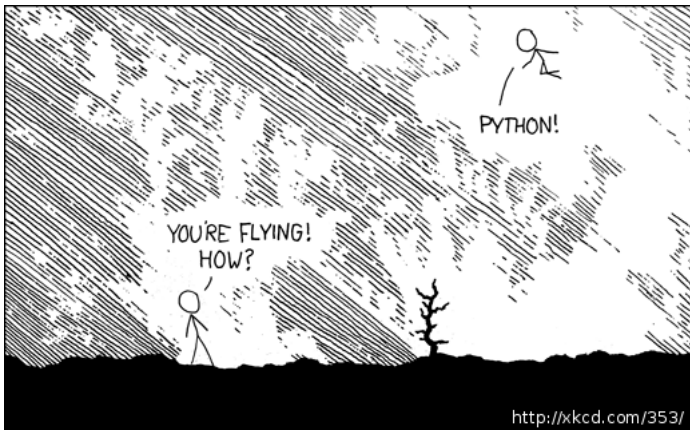


Bienvenido a Python

Un lenguaje especial



Lenguajes de programación

- Hay muchos lenguajes allá afuera
- C, C++, PHP, Lisp, Perl, Java, Ruby, SmallTalk...
- ¿Qué le pedimos a un lenguaje?
 - Fácil de usar
 - Poderoso
 - Multiplataforma
 - Libre



Especial

- Intérprete interactivo
- Librería amplia
- Tipos de datos poderosos
- Identación
- Todo es un objeto



Intérprete interactivo (o REPL)

```
juanjo@fenix:~$ python
Python 2.6.2 (release26-maint, Apr 19 2009, 01:56:41)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 2
3
>>> for i in [1,2,3]:
...     print i
...
1
2
3
```

Tipos de datos

- Básicos
 - Números (enteros, flotantes y más)
 - Cadenas de texto
- Compuestos (y poderosos)
 - Listas
 - Tuplas
 - Conjuntos
 - Diccionarios



Números

```
>>> 1 + 2
```

```
3
```

```
>>> 2 * 3
```

```
6
```

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

```
0
```

```
>>> 8 / 4          # una fácil
```

```
2
```

```
>>> 10 ** 2
```

```
100
```

```
>>> 100 - 5 * 4 - (40 * 2)
```

```
0
```

```
>>> 2.0 + 5
7.0
>>> 4.0 / 3
1.3333333333333333
>>> 4 / 3
1
>>> 4.0 / 3
1.3333333333333333
```

```
>>> int(8.9)
8
>>> float(9)
9.0
>>> abs(-5)
5
```

Complejos

Una parte real y una imaginaria

```
>>> 1j+3
(3+1j)
>>> 1j ** 2
(-1+0j)
>>> 1j * 5
5j
>>> 1j * 5.5
5.5j
>>> complex(1,3)
(1+3j)
>>> z = complex(1,3)
>>> z.real
1.0
>>> z.imag
3.0
```

Cadenas de texto

Una cadena de texto es una secuencia de caracteres

```
>>> "esto es una cadena de texto"
'esto es una cadena de texto'
>>> s = "esto es una cadena de texto"
>>> r = 'esto es una cadena de texto'
>>> s == r
True
>>> "una cadena 'dentro de otra'"
'una cadena 'dentro de otra''
>>> ''' para crear una cadena de
... texto multi línea'''
' para crear una cadena de\ntexto multi l\xc3\xadnea'
```

Formateo de cadenas:

```
>>> c = 'cadenas'  
>>> "Formateo de %s" % (c,)  
'Formateo de cadenas'
```

```
>>> "Formateo de %s: %d %f" % (c, 5, 5.0)  
'Formateo de cadenas: 5 5.000000'
```

```
>>> "Formateo de %s: %d %.2f" % (c, 5, 5.0)  
'Formateo de cadenas: 5 5.00'
```

Listas

```
>>> [1,2,3,4,5]
[1, 2, 3, 4, 5]
>>> [1, "dos", 3j, "cuatro"]
[1, 'dos', 3j, 'cuatro']
>>> a = [1, "dos", 3j, "cuatro"]
>>> a.pop()
'cuatro'
>>> a
[1, 'dos', 3j]
>>> a.pop(1)
'dos'
>>> a
[1, 3j]
```

```
>>> b = ["solo", "con", "cadenas"]
>>> a + b
[1, 3j, 'solo', 'con', 'cadenas']
```

```
>>> b * 2
['solo', 'con', 'cadenas', 'solo', 'con', 'cadenas']
>>> b.append("!!!")
>>> b
['solo', 'con', 'cadenas', '!!!']
```

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> c = range(10)
>>> c[5]
5
>>> c[-1]
9
>>> c[1:3]
[1, 2]
>>> c[2:8]
[2, 3, 4, 5, 6, 7]
>>> c[2:8:2]
[2, 4, 6]
```

```
>>> c[9:5:-1]
[9, 8, 7, 6]
>>> c[:5]
[0, 1, 2, 3, 4]
>>> c[5:]
[5, 6, 7, 8, 9]
>>> c[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> c[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
>>> c[0] = -1
>>> c[5:] = [3]
>>> c
[-1, 1, 2, 3, 4, 3]
```

Tuplas

Una tupla es muy similar a una lista. Con la diferencia de que es inmutable

```
>>> (9,)      # tener en cuenta
(9,)
>>> t = ('solo', 'con', 'cadenas', '!!!!')
>>> t[1]
'con'
>>> t[1] = 'mas'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Conjuntos / set

Un conjunto no posee elementos repetidos

```
>>> heladera = ['huevo', 'huevo', 'queso', 'leche', 'pera', 'pera', 'pera']
>>> alimentos = set(heladera)
>>> alimentos
set(['queso', 'leche', 'huevo', 'pera'])
>>> alimentos.pop()
'queso'
>>> alimentos.remove('leche')
>>> alimentos
set(['huevo', 'pera'])
>>> alimentos.add('empanada')
>>> alimentos
set(['empanada', 'huevo', 'pera'])
```

```
>>> set() < alimentos
True
>>> set() > alimentos
False
```

```
>>> frutas = set(['banana', 'naranja', 'pera'])
>>> frutas - alimentos
set(['banana', 'naranja'])
>>> alimentos - frutas
set(['huevo', 'empanada'])
>>> frutas & alimentos
set(['pera'])
>>> frutas | alimentos
set(['huevo', 'empanada', 'pera', 'banana', 'naranja'])
>>> frutas ^ alimentos
set(['huevo', 'empanada', 'banana', 'naranja'])
```

Diccionarios / dict

```
>>> edades = {'Mary': 22, 'Tomy': 7}
>>> edades['Tomy']
7
>>> edades.keys()
['Tomy', 'Mary']
>>> edades.values()
[7, 22]
>>> edades.items()
[('Tomy', 7), ('Mary', 22)]
>>> edades['Juanjo'] = 24
```

```
>>> edades['Martin']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Martin'
```

```
>>> edades.get('Martin')
>>> print edades.get('Martin')
None
>>> print edades.get('Martin', 0)
0
```

```
>>> edades.popitem()
('Tomy', 7)
>>> edades.pop('Juanjo')
24
>>> edades
{'Mary': 22}
>>> nuevo = dict([('Maira', 22)])
>>> edades.update(nuevo)
>>> edades
{'Mary': 22, 'Maira': 22}
```

Sentencias de control

Permiten definir el flujo de control de un programa.

- if/elif/else
- for
- while
- else, break y continue



if/elif/else

No tenemos switch/case

```
a = 10
if a < 5:
    print "Muy chico"
elif 5 <= a < 10:
    print "Aceptado"
else:
    print "Muy grande!!"
```

for

¿Se dieron cuenta de que el 90% de las veces que utilizamos `for` es para recorrer una secuencia?

```
heladera = ['huevo', 'huevo', 'queso', 'leche']  
for h in heladera:  
    print h[0]
```

```
r = range(10)  
r = range(2,10)  
r = range(2,10,2)  
  
for i in r:  
    print i**2
```

while

```
while True:  
    pass
```

```
while True:  
    entrada = raw_input()  
    if entrada == 'quit':  
        break
```

else, break y continue

- `break` sale del bucle y `else` solo se ejecuta si se consumió toda la lista (en un `for`) o si la condición se hizo falsa (en un `while`)

```
while heladera:
    comida = heladera.pop()
    if comida == 'Mayonesa':
        break
else:
    print "Heladera vacía"
```

- `continue` continua en la próxima ejecución del bucle

```
for x in range(100):
    if x % 11 == 0:
        continue
    suma += x
```

Excepciones

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>> try:
...     1 / 0
... except ZeroDivisionError:
...     print "No podemos dividir por 0"
...
No podemos dividir por 0
```

```
>>> raise TypeError("Se esperaba un entero")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Se esperaba un entero
```

Ejemplo completo:

```
n = raw_input()
try:
    a = 1 / int(n)
except:
    print "Ocurrió un error con %s" % (n,)
else:
    print "Resultado %d" % (a,)
finally:
    print "Tarea finalizada"
```

Funciones

- Retornan valores mediante la palabra reservada `return`
- Como todo en Python, las funciones son objetos

```
def area(a, b):  
    return a * b
```

- Argumentos por defecto al final

```
def saludar(veces, saludo='Hola', nombre='Juanjo'):  
    '''Saluda n veces.'''  
    saludo = "%s %s " % (saludo, nombre)  
    print saludo * veces
```

```
>>> saludar(2)
Hola Juanjo Hola Juanjo
>>> saludar(2, saludo='Hi')
Hi Juanjo Hi Juanjo
>>> saludar(2, 'Hi')
Hi Juanjo Hi Juanjo
>>> saludar(2, nombre='Tomy')
Hola Tomy Hola Tomy
>>> saludar(2, nombre='Tomy', saludo='Alo')
Alo Tomy Alo Tomy
```

```
>>> saludar(2, nombre='Tomy', 'Alo')
File "<stdin>", line 1
SyntaxError: non-keyword arg after keyword arg
```

Classes

```
class Persona(object):  
  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
  
    def mayorEdad(self):  
        return self.edad >= 21
```

```
>>> p = Persona('Juan', 24)  
>>> p.nombre  
'Juan'  
>>> p.mayorEdad()  
True
```

Módulos y paquetes

- Módulos: archivos `.py`

```
>>> import random
>>> dir(random)
...
>>> random._pi
3.1415926535897931
>>> random.choice([1,2,3,4])
```

```
>>> from random import choice
>>> choice([1,2,3,4])
```

```
>>> from random import *
```

- Paquetes: carpetas conteniendo un archivo `__init__.py`

Baterías incluidas

Servicios del sistema, fecha y hora, subprocessos, 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...

- `cd /usr/lib/python2.6/; ls`

Afuera de la librería estándar

- Aplicaciones web: Django
- Clientes y servidores de red: Twisted
- Juegos: PyGame, cocos2d
- GUI: GTK, Qt, Wx
- Imágenes: PIL
- Cálculo numérico: Numpy

Listos para empezar

- ¿Y ahora?
- PyAr: <http://python.org.ar>
- Tutorial: <http://python.org.ar/pyar/Tutorial>

Muchas gracias

- Comentarios, dudas, sugerencias: jjconti@gmail.com
- <http://www.juanjoconti.com.ar>
- <http://www.juanjoconti.com.ar/categoria/aprendiendo-python/>