Introducción a Python

Matías Iturburu

Córdoba 2012

Section 1

print “Hello Python”

Mirando el lenguaje a lo paracaidista

- Interpretado:
- Sintaxis simple
- Multiparadigma
- Baterias incluidas

Interpretado

Scripts compilados a demanda a bytecode.

Soporta extensiones en C y C++

La ejecución de programas grandes es costosa en memoria (pero no tanto)

Sintaxis simple

Amplia variedad de estructuras de datos:

- Strings “”
- Enteros, Flotantes: 3 3.4
- Muchas colecciones: (‘a’, ‘b’) [1,2,3,4,5] {‘key’:value}

Variables sin declaración de tipos, el interprete mantiene la referencia.

```
>>> a = 3
3
>>> a + 2
5
>>> a = "A"
A
```

Con algunas restricciones de tipos:

```
>>> a + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Sin ornamentaciones, los bloques se definen con indentaciones:

```
if 2<3:
    print "Cuando llueve llueve"
else:
    print "Esto no sucede nunca"
```

Un bucle `for` de alto nivel para cualquier tipo `iterable`

```
for i in "This will print each char":
    print i

d = {'a':'b', 'c':'d', 'f':'g'}

for k,v in d
    print v
```

Multiparadigma

Procedural

- Ejecución secuencial de arriba a abajo.
- Ideal para reemplazar scripts bash o perl

```
#!/usr/bin/env python

def func(name):
    return "Hello %s " % name

if __name__ == "__main__":
    import sys
    script, name = sys.argv
    print func(name)
```

Algunos elementos de progamación funcional:

Comprehension de listas, sets* y diccionarios[1]:

```
[x for x in range(0, 10) if x<5 or x>7]

(x for x in range(x, 10))

{i : chr(65+i) for i in range(4)}

map(lambda x: x<3 and x+2 or x+9,
    [x for x in range(0, 5)]
)
```

[1] disponible a partir de python 2.7

Programación orientada a objetos:

- Todo es un `object`. Argumentos, funciones y primitivas.
- Herencia intuitiva, soporte para herencia múltiple.
- No hay métodos privados, pero el scope funciona como esperás.

```
>> class A:
...     name='A'

>>> A
<class __main__.A at 0x1b7ba78>

>>> A.name
'A'

>>> class B:
...     name='B'

>>> class C(A, B):
...     pass

>>> C.name
'A'
```

pass es noop