Python 101 para desarrolladores

Rubén Hortas Astariz

 http://es.linkedin.com/in/rubenhortas

 http://www.rubenhortas.blogspot.com.es/

 https://github.com/rubenhortas

 @rubenhortas

¿Qué es Python?

- Guido Van Rossum
- Finales de los '80 - principios de los '90
- v1.0 1994
- Monthy Python

WTF?

```
# Definimos esto y lo subimos al control de versiones
True = False

if __name__ == '__main__':

    # Esperamos por las risas
    if True == False:
        print 'WTF? AYFKM?'

    print

    if True is False:
        print "ARE YOU SERIOUS?"
```

WTF?

```
# Definimos esto y lo subimos al control de versiones
True = False

if __name__ == '__main__':

    # Esperamos por las risas
    if True == False:
        print 'WTF? AYFKM?'

    print
```

```
ruben@hackbook:~/workspace/python101paraDesarrolladores$ ./wtf.py
WTF? AYFKM?

ARE YOU SERIOUS?
ruben@hackbook:~/workspace/python101paraDesarrolladores$
```

¿Qué es Python?

- Guido Van Rossum
- Finales de los '80 - principios de los '90
- v1.0 1994
- Monthy Python
- Interpretado, tipado dinámico, multiplataforma, orientado a objetos
- Open Source
- Python Software Foundation License (GNU compatible)
- Python 2 vS Python 3

¿Por qué Python?

Pros

- Costes
- Open Source
- Sintaxis limpia
- Curva de aprendizaje suave
- Popularidad (Google, NASA, *hacking*...)
- Alta demanda
- Librerías

Contras

- Fragmentación (py2, py3)
- Algunas cosas sencillas se complican (Clases abstractas, *switch*, propiedades...)
- Arquitectura de los programas
- IDEs*

WTF?

- True = False
- Todo es público

Herencia múltiple

Alcance de las variables

Implementaciones

- CPython, Jython, IronPython, PyPy, Stackless Python, Active Python...

¿Rendimiento?

- py3 > py2
- vS Java?

Entorno de Desarrollo

Módulo interactivo

IDLE, bpython, iPython....

Añaden funcionalidades (Autocompletado, coloreado de sintaxis...)

IDEs

Vim, Sublime Text...

Eclipse + pyDev, Liclipse, Geany, PyCharm...

Desarollo de interfaces

wxPython, wxGlades, Boa Constructor, Visual Python...

https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Entorno de Desarrollo

Control del versiones

git, github, svn, cvs, mercurial...

Eclipse + PyDev + GitHub:

http://rubenhortas.blogspot.com.es/2014/09/integrar-pydev-y-github-en-eclipse.html

Creando repositorios locales en git:

http://rubenhortas.blogspot.com.es/2014/04/creando-repositorios-locales-en-git.html

Entorno virtual

python-virtualenv

Un vistazo rápido

PEP8

Guía de estilo para el código. https://www.python.org/dev/peps/pep-0008

Los bloques se dividen mediante tabulaciones (No mediante llaves {})

**Las líneas se dividen con ** (No hace falta cuando va entre () o [])

Cadenas: '*mi_cadena*' o "*mi_cadena*"

Comentarios

In line: *# comentario*

Documentación: *"""comentario"""* o '''*comentario*'''

Operadores

Lógicos

```
!    -> [is] not
||   -> or
&& -> and
```

Aritméticos

```
*  -> multipicación
** -> exponente
/   -> división
//  -> div. entera
% -> módulo (resto)
```

De bit

```
&    -> and
|    -> or
xor
not
<< -> desp. izq.
>> -> desp. der.
```

Un vistazo rápido

Tipado dinámico

```
def tipado_dinamico():
    # Inicializamos la variable
    x = None
    print '\'x\' vale: %s' % x

    # Ahora es un entero
    x = 1
    print '\'x\' vale: %s' % x

    # Ahora es un string
    x = "uno"
    print '\'x\' vale: %s' % x

    # Tipado dinámico, pero fuertemente tipado.
    # *NO* se puede hacer:
    # x = "uno" + 2
    # TypeError: cannot concatenate 'str' and 'int' objects
```

Un vistazo rápido

Tipado dinámico

```
def tipado_dinamico():
    # Inicializamos la variable
    x = None
    print '\'x\' vale: %s' % x

    # Ahora es un entero
    x = 1
    print '\'x\' vale: %s' % x

    # Ahora es un string
    x = "uno"
    print '\'x\' vale: %s' % x

    # Tipado dinámico, pero fuertemente tipado.
    # *NO* se puede hacer:
    # x = "uno" + 2
```

```
ruben@hackbook:~/workspace/python101paraDesarrolladores$ ./tipadoDinamico.py
'x' vale: None
'x' vale: 1
'x' vale: uno
ruben@hackbook:~/workspace/python101paraDesarrolladores$
```

Un vistazo rápido

Tipos de datos

None

str, **unicode** (inmutables)

bool (numbers.Integral)

int, **float** -> Si hay *overflow*: **long** (inmutables)

Colecciones

list, **dict** (mutables)

tuple (inmutable)

Otros

set, **frozenset** (conjuntos)

complex , real

ellipsis

notImplemented

byteArray

buffer

xrange

...

Estructura de un programa

dir/

nombrePrograma.py

[**setup.py**]

paquete1/

__init__.py **normalmente vacío*

modulo1.py

modulo2.py

...

paquete2/

__init__.py

modulo1.py

...

#!/usr/bin/env/python
 # _*_ coding:utf-8 _*

shebang
hashbang
sharpgang
...
(gnu/linux)

charset

Módulos

PYTHONPATH

* en el directorio del programa -> accesibles para el programa

* en PYTHONPATH -> accesibles para todos los programas

Imports

import modulo ► modulo.funcion()

from modulo import función/clase ► funcion()

from modulo import función/clase as alias ► alias()

~~**from modulo import ***~~

* *en* ***python 3****:* '**.**' *y* '**..**'

Funciones

```
def hola_mundo():
    print 'hola mundo'

# En python una funcion puede devolver varios valores
def devuelve_valores():
    return 1, 3

# Funciones con número variable de parámetros
def parametros_variables(argumento1, argumento2, *varios):
    # Los parámetros *varios serán una tupla
    print 'Tupla de parámetros:', varios
    print 'Argumentos:'
    for argumento in varios:
        print argumento

# Funciones con número variable de parámetros *con nombre*
def kwargs(argumento1, argumento2, **varios):
    # Los parámetros **varios será un diccionario
    # y serán identificados por su nombre (o clave)
    print 'Diccionario de parámetros:', varios
    print 'Argumentos:'
    for item in varios.items():
        print item
```

los bucles *for* son como los *foreach*

Funciones

```
32  if __name__ == '__main__':
33      hola_mundo()
34
35      x,y = devuelve_valores()
36      print 'x vale %s, y vale %s' % (x,y)
37
38      parametros_variables(1, 2, 3)
39      parametros_variables(1, 2, 100, 200, 300)
40
41      kwargs(1, 2, tres=3, cuatro=4)
42
```

```
hola mundo
x vale 1, y vale 3
Tupla de parámetros: (3,)
Argumentos:
3
Tupla de parámetros: (100, 200, 300)
Argumentos:
100
200
300
Diccionario de parámetros: {'cuatro': 4, 'tres': 3}
Argumentos:
('cuatro', 4)
('tres', 3)
ruben@hackbook:~/workspace/python101paraDesarrolladores$
```

Clases

```
class SuperHeroe():
    nombre = None
    batmovil_disponible = True
    batcoptero_disponible = False

    def __init__(self):
        self.nombre = 'Batman'
        print 'Mi nombre es %s' % self.nombre
        self.compi = 'Robin'

    def compi(self):
        print 'Mi compañero es %s' %compi

    def conducir(self):
        if self.batcoptero_disponible:
            print "Al batcóptero!"
        else:
            if self.batmovil_disponible:
                print "Al batmóvil!"
            else:
                print "A batpatas!"

if __name__ == '__main__':
    mi_batman = SuperHeroe()
    mi_batman.conducir()
```

no hereda de nada

atributos

atributos y métodos especiales

siempre como 1er parámetro

los atributos
se pueden definir en
cualquier parte

Name Mangling

```
    def __identidad_secreta(self):
        print 'Soy Bruce Wayne :('

if __name__ == '__main__':
    mi_batman = SuperHeroe()
    mi_batman.desvelar_compi()
    mi_batman.conducir()

    mi_batman._SuperHeroe__identidad_secreta()
```

privado?

objeto — un guión — clase — dos guiones

útil para *debug* y *tests*

```
Mi nombre es Batman
Mi compañero es Robin
Al batmóvil!
Soy Bruce Wayne :(
ruben@hackbook:~/workspace/python101paraDesarrolladores
```

Manejo de excepciones

```
try:
    # hacer algo

except [tipo]:
    # manejar la excepción

[ except otro_tipo]:
    # manejar la excepción

[else]
    # hacer algo

[finally]
    # hacer algo
```

0..n tipos (NameError, ValueError, IOError, OverflowError...)

Si no se ha producido ninguna excepción

Notas rápidas

Threads y subprocesos

Threads:

- Comparten recursos entre sí (p.e. memoria global).
- Controlados por el *Global Interprete Lock* (**GIL**)➙ Sólo 1 proc. a la vez
- Librerías: *Thread* y *Threading*

Subprocesos:

- Lanzar nuevos procesos y conectarnos a su salida (Código de salida, entrada, salida, error).
- Librerías: subprocess, subprocess32,

* *IronPython* y *Jython* carecen de **GIL**.

Expresiones regulares

- Módulo ***re***
- *Compilarlas para aumentar el rendimiento:*
  re.compile("^[\w \(\)]*", re.UNICODE)

Notas rápidas

Tests de Unidad:

PyUnit

- Estándar para *Python*.
- Una versión de *JUnit*
- Muy fácil de usar
- Muy flexible

Documentación automática:

- **PyDoc**
- **EpyDoc**

html, xml, pdf...

Python 2 a Python 3:

- **2to3** https://docs.python.org/2/library/2to3.html

Notas rápidas

Distribuir las aplicaciones:

Distutils → **setup.py**

Setuptools →
- Extiende a *Distutils*
- Introduce el formato ***.egg***
- permite instalar paquetes PyPi

***egg** →
- multiplataforma
- permiten manejar dependencias
- permiten instalar distintas versiones del mismo paquete
- se pueden instalar con ***easy_install***

***exe** → py2exe

Bibliografía

***Online*:**

- Documentación oficial: https://www.python.org/doc/
- Python wiki: https://wiki.python.org/moin/

Libros y documentos: (mi recomendación)

- Python para todos (Raúl González Duque) http://mundogeek.net/
- Learn Python The Hard Way (Zed Shaw) http://learnpythonthehardway.org/
- Dive Into Python (Mark Pilgrim) http://www.diveintopython.net/
- Dive Into Python 3 (Mark Pilgrim) http://www.diveintopython3.net/
- Gray Hat Python (Justin Seitz)
- High Performance Python tutorial v0.2 (from EuroPython 2011) (Ian Osvald)

Cuando todo falle... **stackoverflow**! ;)

Python 101 para desarrolladores

Rubén Hortas Astariz

 http://es.linkedin.com/in/rubenhortas

 http://www.rubenhortas.blogspot.com.es/

 https://github.com/rubenhortas

 @rubenhortas