Python ¿2 vs 3?

Joel Rivera

joel@pymty.org

Python Monterrey

www.pymty.org

19 de Febrero de 2015

Pure Python

La platica va orientada a **pure** Python, con esto quiero decir que no se involucra con el API en C de CPython que también cambió.

Como usuario de bibliotecas en python, solamente notas el efecto de que módulos que usan el API en C tienden a tardarse en ser portados a Python 3.

La razón

Como cualquier otro lenguaje que va evolucionando. Python llevaba arrastrando un conjunto de características *legacy* o en contra de la filosofía misma del lenguaje que se va formando conforme avanza.

Usualmente lo más susceptibles a pagar por esa deuda técnica son core developers, quienes simplemente les quedaba tolerar el no poder implementar mejoras en la robustés del lenguaje por problemas de compatibilidad.

Todo lo que se consideraba que no fuese retro-compatible se le suponía a un futuro lejano llamado **Python 3000** o **Py3k**

La idea se origino cercas del 2001.

Python 3.0 fue liberado el 3 de Diciembre del 2008.

Al día de hoy han pasado cerca de 6 años desde su salida.

El estado actual

Ultima versión estable **Python 3.4.2**

Soporte a Python 3 en bibliotecas populares:

- Python 3 Readiness: http://py3readiness.org/
- Python 3 Wall Of Superpowers: https://python3wos.appspot.com/

Casi todos los módulos más populares y con desarrollo activo ya fueron portados a Python 3.

Python 3.3 y superiores agregaron características en el lenguaje para facilitar la retro-compatibilidad con Python 2.7.

El problema

Rompieron la **regla de oro** justo cuando el lenguaje estaba más que nunca ganando popularidad (2006 - 2010). Por lo que a gran parte de la comunidad les pareció de locos.

Y a otros más no les parece razonable pagar el costo de re-escribir cosas por que no ganaban algo significativo (como no tener un GIL).

Mientras los core developers se regocijaban en su core limpio y organizado donde podían trabajar más a gusto.

Algunos de los cambios relevantes

- Nombres de módulos
- Clara distinción sobre bytes y strings.
- La estructura interna de las clases.
- Cambios sutiles en reflexión, excepciones y metaclases.
- Una reorganización en general de los módulos en la biblioteca estándar.

Y lo más importante…

print pasa de ser una **palabra reservada** a una **función.**

Al igual que `exec` pero a ese casi nadie le llora.

El verdadero problema

Reescribir código

Implementaciones más limitadas

- PyPy:: soporte experimental para Python 3.
- Jython:: Apenas está (en beta) con Python 2.7.
- IronPython:: Solo soporta Python 2.7.

Disponibilidad de bibliotecas

No hay mejor código que el que no se escribe.

¿Y por que habría de importarme?

Python 3 es mucho más uniforme.

Python 3 es El futuro del lenguaje.

Si apenas comienzas con el lenguaje conviene que te familiarices con idiomas modernos y saques provecho a lo más nuevo del lenguaje.

Python 2 se considera en mantenimiento. Es como Windows XP.

Estrategías de migración

- Soportar solamente Python 3.
- Ramas de desarrollo independientes para 2 y 3.
- Convertir el código con 2to3.
- Python 2 y 3 sin conversión.

Preparando el código.

Usa Python 2.7.

Usar // en vez de / para dividir enteros.

Al ordenar usa `key` en vez de `cmp`

New-style classes.

New style:

```
class Persona(object):
    def method(self):
        pass
```

Old style:

```
class Persona:
    def method(self):
        pass
```

En Python 3, no es necesario heredar de `object` por lo que es lo mismo:

```
class Persona:
    ....
```

que esto

```
class Persona(object):
    ....
```

Pero el segundo es new-style en Python 2 y Python 3.

Separar datos binarios y cadenas.

Gran parte de el como hacer esto es dependiendo de la lógica del programa, por ejemplo al procesar imágenes, no deberían de tener un encoding y por lo tanto deben de manipularse y comportarse como bytes:

```
>>> b'abcd'[0]
97
```

No como cadena:

```
>>> 'abcd'[0]
a
```

Literal byte string:

b'raw bytes'

Literal unicode string: (por compatibilidad se agrego en python 3.3)

u'¡San Nicolás!'

Methodos de comparación.

`__cmp__` es remplazado por los métodos de comparación más explícitos:

- `__lt__` <
- `__le__` <=
- `__gt__` >
- `__ge__` >=
- `__eq__` ==
- `__ne__` !=

Verificar dependencia a módulos que ya no existen.

Iteradores sobre diccionarios.

Reorganización de la biblioteca estándar

Lo siguiente es un extracto de **Porting to Python 3** por *Lennart Regebro*.

No incluye la reestructuración de `urllib`.

Parte 1 / 5

Python 2 name	Python 3 name	six name
anydbm	dbm	
BaseHTTPServer	http.server	BaseHTTPServer
__builtin__	builtins	builtins
CGIHTTPServer	http.server	CGIHTTPServer
ConfigParser	configparser	configparser
copy_reg	copyreg	copyreg
cPickle	pickle	cPickle
cStringIO.StringIO	io.StringIO	cStringIO

Parte 2 / 5

Python 2 name	Python 3 name	six name
Cookie	http.cookies	http_cookies
cookielib	http.cookiejar	http_cookiejar
dbhash	dbm.bsd	
dbm	dbm.ndbm	
dumbdb	dbm.dumb	
Dialog	tkinter.dialog	tkinter_dialog
DocXMLRPCServer	xmlrpc.server	
FileDialog	tkinter.FileDialog	tkinter_filedial
FixTk	tkinter._fix	

Parte 3 / 5

Python 2 name	Python 3 name	six name
gdbm	dbm.gnu	
htmlentitydefs	html.entities	html_entities
HTMLParser	html.parser	html_parser
httplib	http.client	http_client
markupbase	_markupbase	
Queue	queue	queue
repr	reprlib	reprlib
robotparser	urllib.robotparser	urllib_robotpar
ScrolledText	tkinter.scolledtext	tkinter_scrolle
SimpleDialog	tkinter.simpledialog	tkinter_simpled

Parte 4 / 5

Python 2 name	Python 3 name	six name
SimpleHTTPServer	http.server	SimpleHTTPS
SimpleXMLRPCServer	xmlrpc.server	
StringIO.StringIO	io.StringIO	
SocketServer	socketserver	socketserve
test.test_support	test.support	tkinter
Tkinter	tkinter	tkinter
Tix	tkinter.tix	tkinter_tix
Tkconstants	tkinter.constants	tkinter_con
tkColorChooser	tkinter.colorchooser	tkinter_col

Parte 5 / 5

Python 2 name	Python 3 name	six name
tkCommonDialog	tkinter.commondialog	tkinter_commond
Tkdnd	tkinter.dnd	tkinter_dnd
tkFileDialog	tkinter.filedialog	tkinter_tkfiled
tkFont	tkinter.font	tkinter_font
tkMessageBox	tkinter.messagebox	tkinter_message
tkSimpleDialog	tkinter.simpledialog	tkinter_tksimpl
turtle	tkinter.turtle	
UserList	collections	
UserString	collections	
whichdb	dbm	
_winreg	winreg	winreg
xmlrpclib	xmlrpc.client	

Sobre Unicode

Errores comunes

Casi todo pythonista en algún momento de su vida le toco algo así:

```
python LoL_Coding.py
  File "LoL_Coding.py", line 3
SyntaxError: Non-ASCII character '\xc2' in file LoL_Coding.py on line 3, but no encoding
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 4: ordinal not in ra
```

En especial si lo que programas lee de archivos o red.

Erroces comunes cont.

La verdad es que es un problema que nos buscamos por no tener una clara distinción sobre cadenas de caracteres, bytes y encodings.

Que en general se puede resumir a:

bytes + encoding -> caracteres -> encoded string

Python 3 toma una postura mucho más acorde a lo que se espera en estos años de i18n y de paso te ahorras muchos:

```
-*- coding: utf-8 -*-
```

En todos tus archivos.

El lado obscuro

Dicho lo anterior existen problemas sobre la aproximación de unicode en Python, es particularmente molesta cuando se lidia con protocolos con encodings que no están claramente definidos o se pasa de bytes a cadenas de forma constante.

La critica principal es sobre un soporte a unicode enfocado hacia los principiantes.

Para más información:

- http://lucumr.pocoo.org/2014/1/5/unicode-in-2-and-3/
- http://lucumr.pocoo.org/2014/5/12/everything-about-unicode/

`form __future__ import *`

Como forma de facilitar el portar el código de vesiones anteriores se tienen al alcance las siguientes llamadas a `__future__`

Carácteristica	**Opt.**	**Mand.**	**Efecto**
`division`	2.2.0a2	3.0	PEP 238: Changing the Division Operator
`absolute_import`	2.5.0a1	3.0	PEP 328: Imports: Multi-Line and Absolute/Relative
`print_function`	2.6.0a2	3.0	PEP 3105: Make print a function
`unicode_literals`	2.6.0a2	3.0	PEP 3112: Bytes literals in Python 3000

Mi recomendación

Proyectos personales

- Verifica tus dependencias
- En caso de que todas soporten python 3 **usalo**.
- En caso contrario usa python 2.7 favoreciendo los módulos con soporte a 3.
- Escribe tú código de Python 2 usando idiomas modernos.

Distribuir código

- Si estar haciendo un módulo para compartir soporta las dos versiones.
- Empieza con >3.3 o >2.6 teniendo en mente la compatibilidad.
- Considera usar `six` para la capa de compatibilidad.

Cool Python 3 stuff

- Unicode por defecto en todas partes
- `asyncio`
- `enum`
- `ipaddress`
- `ensurepip`
- `pathlib`
- `statistics`
- `tracemalloc`
- `faulthandler`
- `nolocal`

Cool Python 3 stuff cont.

- `selectors`
- `concurrent.futures`
- `yield from`
- `@functools.singledispatch`
- `venv`
- `sys.setswitchinterval`
- keyword only arguments
- tuple unpacking
- anotaciones
- Mejor instalador en windows

Recursos y ¿Preguntas?

- Python 3 Readiness: http://py3readiness.org/
- Python 3 Wall Of Superpowers: https://python3wos.appspot.com/
- Porting to Python 3: http://python3porting.com/
- Why should OpenStack move to Python 3 right now?: http://techs.enovance.com/6521/openstack_python3
- Unicode in 2 and 3: http://lucumr.pocoo.org/2014/1/5/unicode-in-2-and-3/
- Everything about Unicode: http://lucumr.pocoo.org/2014/5/12/everything-about-unicode/
- six: https://pypi.python.org/pypi/six

Recursos y ¿Preguntas? cont.

- `__future__`: https://docs.python.org/2/library/__future__.html
- What's new in Python 3.0: https://docs.python.org/dev/whatsnew/3.0.html
- What's new in Python 3.1: https://docs.python.org/dev/whatsnew/3.1.html
- What's new in Python 3.2: https://docs.python.org/dev/whatsnew/3.2.html
- What's new in Python 3.3: https://docs.python.org/dev/whatsnew/3.3.html
- What's new in Python 3.4: https://docs.python.org/dev/whatsnew/3.4.html