Python 3.3: ¡A migrar!

Jesús Cea Avión
Twitter: @jcea
jcea@jcea.es
http://www.jcea.es/

12 de diciembre de 2012

Origen y motivación:

Objetivo: Para un programador nuevo, Python 3 es un lenguaje más “limpio” y regular.

- Limpieza de inconsistencias y redundancias.
  - “next” en vez de “__next__”.
- Sintaxis más clara.
  - “except Exception as e”.
- Eliminar errores de diseño:
  - “print” como función.
- Abrazar UNICODE de forma nativa.
- Unificación de clases (ahora todas son “new style”).

No hay beneficio sin coste:

- Incompatible con Python 2.x.
  - División de la comunidad.
  - Problema de “bootstrapping”: Librerías y “frameworks”.
- Unicode nativo supone un cambio conceptual.
  - Todos pagan el coste, pocos perciben el beneficio.
  - Posiblemente la mayor causa de incompatibilidad.
  - Los “pickles” necesitan conversión.
- Reestructuración de módulos en la “stdlib”.
  - urllib.
- Necesidad de definir nuevos APIs:
  - WSGI.
  - Email.

Ayudas a la migración (I):

- Herramientas de migración:
  - Mantener una única versión del código.
  - 2to3 y 3to2.
  - Integrado con herramientas de distribución, como “pip”.

- Coexistencia con Python 2.x, llamándolo “python3”.

- Presión social, becas y “bounties”.

- Algunas mejoras de sintaxis en Python 2.6/2.7:
  - “print” como función.
  - “except Exception as e”.
  - “bytes”.
  - Ya no se va a hacer más backporting a Python 2.x.

Ayudas a la migración (II):

- Mantenimiento de Python 3.x y 2.7 en paralelo
  - Python 2.7 mantenido hasta 2015.
    - Solo “bugfixes”, no nuevas prestaciones o sintaxis.
    - Soporte extendido comparado con Python 2.x anteriores, donde en Python 2.x solo se daba soporte de seguridad tras publicarse Python 2.x+1.

- Desarrollo concentrado en Python 3, para incentivar el cambio y no dividir esfuerzos.
  - No va a haber un Python 2.8.
  - Python 3 tiene mejoras importantes en funcionalidades, sintaxis y rendimiento.

- Plan de migración de CINCO años.
  - Estamos en el año TRES, y la migración va “bien”.
  - Efecto red: proyectos “clave”.
  - Optimismo.

Migración de proyectos actuales:

- Programa:
  - Si es posible, Python 3 exclusivamente.
  - Si es necesario, un único código y conversión automática con “2to3” o “3to2”, o librerías puente.
    - Se puede automatizar hacia arriba o hacia abajo.
    - Target Python 2.7.
    - Mejoras en Python 3.3: prefijo “u” e importlib implementado en Python.
    - Vigilar las dependencias externas de módulos.
    - **¡Bajo ningún concepto mantener dos árboles de código!.**

- Librería:
  - Hoy por hoy, soportar solo Python 3 no suele ser realista.
    - Conversión automática.
    - Pensar cuidadosamente cuál es la versión que se desarrolla: 2.x o 3.x.
    - Se pueden publicar ambas versiones o utilizar conversión en tiempo de instalación.
    - Verificar los tests en ambas versiones.
    - La distinción bytes/unicode
    - **¡Bajo ningún concepto mantener dos árboles de código!.**

Ejemplos concretos (I):

- Módulo en C (pybsddb):

```
    #if (PY_VERSION_HEX < 0x03000000)
    DL_EXPORT(void) init_bsddb(void)
    #else
    PyMODINIT_FUNC  PyInit__bsddb(void)    /* Note the two underscores */
    #endif
    {
[…]
        /* Create the module and add the functions */
    #if (PY_VERSION_HEX < 0x03000000)
        m = Py_InitModule(_bsddbModuleName, bsddb_methods);
    #else
        m=PyModule_Create(&bsddbmodule);
    #endif
        if (m == NULL) {
    #if (PY_VERSION_HEX < 0x03000000)
            return;
    #else
            return NULL;
    #endif
        }
```

Ejemplos concretos (II):

- Módulo en C (pybsddb):

```
#if (DBVER >= 44)
static PyObject*
DBTxn_get_name(DBTxnObject* self)
{
    int err;
    const char *name;

    MYDB_BEGIN_ALLOW_THREADS;
    err = self->txn->get_name(self->txn, &name);
    MYDB_END_ALLOW_THREADS;

    RETURN_IF_ERR();
#if (PY_VERSION_HEX < 0x03000000)
    if (!name) {
        return PyString_FromString("");
    }
    return PyString_FromString(name);
#else
    if (!name) {
        return PyUnicode_FromString("");
    }
    return PyUnicode_FromString(name);
#endif
}
#endif
```

Ejemplos concretos (III):

- Módulo en Python 2.4 (pybsddb):
  - ¡El único (casi) cambio es la batería de test!.

```
    if sys.version_info[0] >= 3 :
        charset = "iso8859-1"  # Full 8 bit
[…]
        class cursor_py3k(object) :
            def __init__(self, db, *args, **kwargs) :
                self._dbcursor = db.cursor(*args, **kwargs)

            def __getattr__(self, v) :
                return getattr(self._dbcursor, v)

            def _fix(self, v) :
                if v is None : return None
                key, value = v
                if isinstance(key, bytes) :
                   key = key.decode(charset)
                return (key, value.decode(charset))

            def next(self) :
                v = getattr(self._dbcursor, "next")()
                return self._fix(v)
[…]
```

CONTINUA...

Ejemplos concretos (IV):

- Módulo en Python 2.4 (pybsddb):
  - ¡El único (casi) cambio es la batería de test!.

VIENE DE LA PÁGINA ANTERIOR...

```
        bsddb._db.DBEnv_orig = bsddb._db.DBEnv
        bsddb._db.DB_orig = bsddb._db.DB
[…]
        def do_proxy_db_py3k(flag) :
            flag2 = do_proxy_db_py3k.flag
            do_proxy_db_py3k.flag = flag
            if flag :
                bsddb.DBEnv = bsddb.db.DBEnv = bsddb._db.DBEnv = DBEnv_py3k
                bsddb.DB = bsddb.db.DB = bsddb._db.DB = DB_py3k
                bsddb._db.DBSequence = DBSequence_py3k
            else :
                bsddb.DBEnv = bsddb.db.DBEnv = bsddb._db.DBEnv =
bsddb._db.DBEnv_orig
                bsddb.DB = bsddb.db.DB = bsddb._db.DB = bsddb._db.DB_orig
                bsddb._db.DBSequence = bsddb._db.DBSequence_orig
            return flag2

        do_proxy_db_py3k.flag = False
        do_proxy_db_py3k(True)
```

Ejemplos concretos (V):

- Módulo en Python 2.4 (pybsddb):
  - Vale, hay más cambios en el código en sí :-).

```
    class _DBWithCursor(_iter_mixin):
    """
    A simple wrapper around DB that makes it look like the bsddbobject in
    the old module.  It uses a cursor as needed to provide DB traversal.
    """
[…]
        def next(self):  # Renamed by "2to3"
            self._checkOpen()
            self._checkCursor()
            rv = _DeadlockWrap(getattr(self.dbc, "next"))
            return rv

        if sys.version_info[0] >= 3 :  # For "2to3" conversion
            next = __next__
```

Prestaciones (I):

- Las versiones recientes de Python 3 son más rápidas que Python 2.7.
  - Computed gotos.
  - Mejoras en el código, nuevas extensiones, mejoras en el lenguaje.
  - Nuevo GIL. Gran mejora con hilos y contención.
  - “python perf.py -r -b 2n3 /usr/local/bin/python2.7 /usr/local/bin/python3.x”:
    - Valores aproximados, porque mi máquina no está “idle”:
      - 3.3: V***ER PÁGINA SIGUIENTE***
  - “pythonx.x -c "import pystone; pystone.main()”:
    - Valores aproximados, porque mi máquina no está “idle”:
      - 2.7.3: 74626.9 (con parche dtrace - http://bugs.python.org/issue13405)
      - 3.3.0: 58139.5 (con parche dtrace - http://bugs.python.org/issue13405)

- La gestión de Unicode de Python 3.3 reduce el consumo de memoria de las cadenas de texto al 25%, situándose al mismo nivel que Python 2.x.

Prestaciones (II):

- Report on Linux ubuntu 2.6.32-45-generic #101-Ubuntu SMP Mon Dec 3 15:39:38 UTC 2012 x86_64
  - Total CPU cores: 2
  -
  - ### call_method ###
  - Min: 0.670035 -> 0.568963: 1.18x faster
  - Avg: 0.719628 -> 0.594890: 1.21x faster
  -
  - ### call_method_slots ###
  - Min: 0.651521 -> 0.567072: 1.15x faster
  - Avg: 0.719392 -> 0.585391: 1.23x faster
  -
  - ### call_method_unknown ###
  - Min: 0.695694 -> 0.610098: 1.14x faster
  - Avg: 0.743794 -> 0.668352: 1.11x faster
  -
  - ### call_simple ###
  - Min: 0.534279 -> 0.439838: 1.21x faster
  - Avg: 0.639677 -> 0.470177: 1.36x faster
  -
  - ### chaos ###
  - Min: 0.459810 -> 0.540049: 1.17x slower
  - Avg: 0.481634 -> 0.576451: 1.20x slower
  -
  - ### fannkuch ###
  - Min: 1.913188 -> 2.066090: 1.08x slower
  - Avg: 1.995462 -> 2.162834: 1.08x slower
  -
  - ### fastpickle ###
  - Min: 1.050495 -> 0.960172: 1.09x faster
  - Avg: 1.072893 -> 0.982561: 1.09x faster
  -
  - ### fastunpickle ###
  - Min: 0.765980 -> 0.911968: 1.19x slower
  - Avg: 0.791235 -> 0.932776: 1.18x slower
  -
  - ### float ###
  - Min: 0.609450 -> 0.544641: 1.12x faster
  - Avg: 0.646323 -> 0.572678: 1.13x faster
  -
  - ### formatted_logging ###
  - Min: 0.420418 -> 0.534919: 1.27x slower
  - Avg: 0.435264 -> 0.554873: 1.27x slower
  -
  - ### go ###
  - Min: 0.917837 -> 1.030598: 1.12x slower
  - Avg: 0.946883 -> 1.053268: 1.11x slower
  -
  - ### hexiom2 ###
  - Min: 239.327711 -> 252.170324: 1.05x slower
  - Avg: 239.737899 -> 252.617492: 1.05x slower
  -
  - ### iterative_count ###
  - Min: 0.171500 -> 0.252649: 1.47x slower
  - Avg: 0.186699 -> 0.261984: 1.40x slower
  -
  - ### json_dump_v2 ###
  - Min: 4.836695 -> 5.235043: 1.08x slower
  - Avg: 4.983184 -> 5.374492: 1.08x slower
  -
  - ### json_load ###
  - Min: 1.378689 -> 0.751976: 1.83x faster
  - Avg: 1.417634 -> 0.783172: 1.81x faster
  -
  - ### nbody ###
  - Min: 0.547911 -> 0.468763: 1.17x faster
  - Avg: 0.575928 -> 0.488775: 1.18x faster
  -
  - ### normal_startup ###
  - Min: 0.761073 -> 0.967400: 1.27x slower
  - Avg: 0.809733 -> 1.019968: 1.26x slower
  -
  - ### nqueens ###
  - Min: 0.457993 -> 0.539339: 1.18x slower
  - Avg: 0.475843 -> 0.554685: 1.17x slower
- [...]

Conclusiones:

- La migración a Python 3.x es inevitable.

- Python 3 es un lenguaje “mejor”, más limpio y regular.

- La migración de librerías y productos de terceros “va bien”.

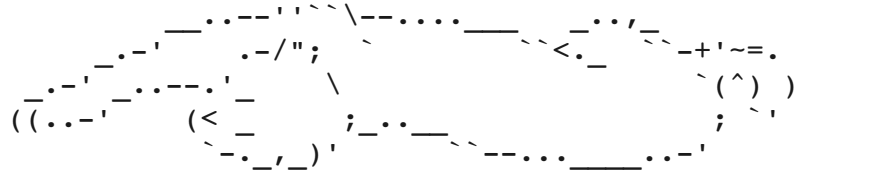
- Mantener un único “codebase” y hacer la conversión automática es simple, y la ruta recomendada.

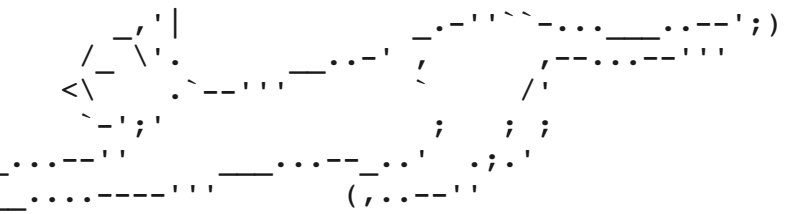
- Efecto red.

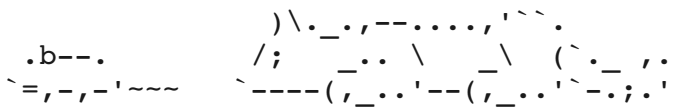
Recursos:

- http://packages.python.org/six/
  - Constantes, modelo de objetos, sintaxis, binario y cadenas, módulos y atributos renombrados,
- http://python3porting.com/
- http://getpython3.com/
- http://docs.python.org/dev/howto/pyporting.html
- https://bitbucket.org/amentajo/lib3to2/overview
- http://mail.python.org/mailman/listinfo/python-porting
- Los “What's new” de Python 3.0, 3.1, 3.2 y 3.3.

Las fotos de gatitos de rigor...







Ejemplos específicos de cambios y mejoras en Python3