Python 3: The Next Generation

Wesley J. Chun
wescpy@gmail.com
http://corepython.com
Winter 2010

About the Speaker

- Software engineer by profession
  - Over a decade as a full-time Python developer
  - Original Yahoo!Mail engineering team (AB & SC)
  - Came from C, Unix, networking background
  - Currently in Developer Relations at Google
- Course instructor: teaching Python since 1998
  - Private Corporate Training & Public Courses
- Community volunteer
  - User groups: BayPIGgies and SF Python Meetup
  - Other: Tutor mailing list; Python conferences
- Author/co-author (books, articles, blog, etc.)
  - *Core Python Programming* ([2009,] 2007, 2001)
  - *Python Fundamentals* LiveLessons DVD (2009)
  - *Python Web Development with Django* (2009)

Books

About You and This Talk

- Assumes some Python knowledge/experience
  - Will not cover Python basics here
- Today focused on Python 3
  - Differences between Python 2 and 3
  - Role of remaining Python 2.x releases
  - Timeline and Transitioning

Questions

- What does it all mean?
- Are all my Python programs going to break?
- Will I have to rewrite everything?
- How much time do I have?
- When is Python 2 going to be EOL'd?
- Is Python being rewritten completely and will I even recognize it?
- What are the changes between Python 2 and 3 anyway?
- Are migration plans or transition tools available?
- Should I start w/Python 2 or Python 3 if I want to learn Python?
- Are all Python 2 books obsolete?

Fact of Fiction? Rumors all TRUE...

- Python 3 does exist
- There are some users of Python 3
- Most corporations still using Python 2
- Some projects have been ported to Python 3
- More projects have started porting to Python 3
- I am not a Python 3 user (yet)

Python 2 and Python 3

- Python stands at a crossroads
- In transition to next generation
  - I (+courses & books) promote version-independence
  - They're all about language itself
  - Not focused on syntax differences
- BUT
  - Cannot ignore 3.x backwards-incompatibility

Python 3: The What and the Why

- Justifying the existence of 3.x
  - Fix early design flaws
  - Provide more universal data types
  - Clean up language and library
  - Some new features, many small improvements
- Plan
  - Timeline: 2.x will live on for some time
  - 2.x and 3.x developed in parallel
  - Migration tools (i.e., `2to3` , Python 2.6+)
- More information in PEPs 3000 and 3100

3.x Not Backwards-Compatible

- Are all my Python programs going to break? YES, MOST OF THEM
- Will I have to rewrite everything? HOPEFULLY NOT
  - Major porting shouldn't be required
  - Will discuss transtion tools/plans soon
- Causes the most (negative) buzz in industry
- Won't execute most code written for 1.x/2.x intepreters
- Is Python being rewritten completely and will I even recognize it? NO, YES
  - General syntax still quite similar
  - Easily broken when `print` becomes a function (vs. stmt)

Stability Over the Years

- Backwards-compatibility never really been an issue
- Steadfast determination to preserve compatibility
- In 2000, Python 2.0 ran 1.5.2 software just fine
- 2.0a released on same day as 1.6 (Why? ASFAT.)
- 2.6 developed at same time as 3.0 (Why? Wait.)
- Cost: passes on "sticky" flaws & deprecated features

Python "Regrets" and "Warts"

Why is Python Changing?

- Why **isn't** Python changing?
  - It usually doesn't!
  - Has always been backwards compatible
  - Python 3 still recognizable
  - Not being rewritten/redesigned from scratch
- Not a standard (yet)
  - Backwards-incompatible for the future's sake
  - Must drop "sticky" flaws and deprecated features
  - Iterate, improve, evolve, etc.

Python 3 Breakage

- 1st release that deliberately breaks compatibility
  - No promise that it will not ever happen again
  - But it took 18 years for this first one to occur
- "Backcompat" always top priority *except* this time
  - BTW, it's still a high priority
- Python follows agile method of continuous iteration
  - Interpreter development follows methodology too
  - 3.0 just a bit larger of a hop

Python 2 vs. 3: Key Differences

- `print` & `exec` changed to functions
- Strings: Unicode; `bytes/bytearray` types
- True division
  ```
  1/2 == 0.5
  ```
- Updated Syntax for Exceptions
- Iteration upgrades/Iterables Everywhere
- Various Type Updates
  - One class type
  - Updates to integers
  - Cannot compare mixed types
  - New "construction"
- Other Minor Changes
  - Fixes, Deprecation, Improvements

`print` : Statement to Function

- Easiest way to slip up in Python 3
  - Especially in interactive interpreter
  - Need to get used to adding parentheses
- Why the change?
  - As a statement, limits improvements to it
- As a function...
  - Behavior can be overridden w/keyword parameters
  - New keyword parameters can be added
  - Can be replaced if desired, just like any other BIF*
- More information in PEP 3105
- (*) BIF = built-in function, FF = factory function

`print` in Python (1 and) 2

- Using the "old" `print`

```
>>> i = 1
>>> print 'Python' 'is', 'number', i
Pythonis number 1
```

- Using the "new" `print` in 2.6+

```
>>> from __future__ import print_function
>>> print
<built-in function print>
>>> print('foo', 'bar')
foo bar
```

`print` () in Python 3

- Using the "new" `print` in 3.0+

```
>>> i = 1
>>> print('Python' 'is', 'number', i)
Pythonis number 1
```

- (Deliberate exclusion of comma b/w 'Python' & 'is')

Strings: Unicode by Default

- This change couldn't come soon enough
- People have daily issues w/Unicode vs. ASCII
- Does the following look familiar?

```
UnicodeEncodeError: 'ascii' codec can't
  encode character u'\xae' in position 0:
  ordinal not in range(128)
```

- Results from non-ASCII characters in valid 8-bit strings
- More Unicode info:
  http://docs.python.org/3.0/howto/unicode.html

New String Model

- Users shouldn't even use those terms any more
  - It's not Unicode vs. ASCII; it's *text* vs. *data*
  - Text represented by Unicode... real "strings"
  - Data refers to ASCII, bytes, 8-bit strings, binary data
- Changes
  - `str` type now `bytes` (new `b` literal)
  - `unicode` type now `str` (no more `u` literal)
  - `basestring` deprecated (former base class)
  - New mutable `bytesarray`
- More information in PEPs 358, 3112, 3137, 3138

Single Class Type

- 2.2: first step taken to unify classes & types
  - Since then, there have been 2 class types
- Original classes called "classic classes"
- Second generation classes called "new-style classes"
- Python 3 deprecates classic classes
  - They no longer exist
  - All classes are of the same type
- More information in PEPs 252 and 253

Classic Classes

- "Normal" classes in typical OOP languages
  - Classes: types
  - Instances: objects of those types
- Problem: Python classic classes *not* normal
  - Classes: "class objects"
  - Instances: "instance objects"
- Existing Python types can't be subclassed (not classes!)
  - Common programmer desire to modify existing types
  - Handicapped versions of certain types had to be created
    - `UserList`, `UserDict`, etc.

Classic vs. New-style classes

- Syntactically, difference is `object`

```
class ClassicClass:
    pass
```

- vs

```
class NewStyleClass(object):
    pass
```

- In Python 3, both idioms create same class type

Updated Syntax for Exceptions

- In Python (1 and) 2, multiple idioms...
  - For raising exceptions
  - For handling exceptions
- In Python 3, syntax...
  - Improved, consolidated, less confusing
- More information in PEP 3109 and 3110

Exception Handling

- Catching/Handling One Exception
  `except ValueError, e:`
- Catching/Handling Multiple Exceptions
  `except (ValueError, TypeError), e:`
- `e` : exception instance usually has error string
- Mistakes easily made as parentheses required!!
  - Developers attempt the invalid:
    `except ValueError, TypeError, e:`
  - Code does not compile (`SyntaxError`)

Improving Handling Mechanism

- (New) `as` keyword helps avoid confusion
- Parentheses **still** required
- Equivalents to earlier `except` statements:

```
except ValueError as e:
except (ValueError, TypeError) as e:
```

- Required in 3.0+
- Available in 2.6+ as transition tool
  - Yes, 2.6+ accepts **both** idioms
- More information in PEP 3110

Consolidated Exception Throwing/Raising

- How do I say this?
- Python has more than one way to throw exceptions
  - 12(!) actually if you're counting
- The most popular over the years:

```
raise ValueError:
raise ValueError, e:
```

- Remember:
  - "There should be one -- and preferably only one -- obvious way to do it."
  - From the Zen of Python (`import this`)

New Idiom with Exception Classes

- Exceptions used to be strings
- Changed to classes in 1.5
- Enabled these new ones:

```
raise ValueError()
raise ValueError(e)
```

- Required in 3.0+
- Available in 1.5+ as transition tool :-)
- (Changed to new-style classes in 2.5)

Single Integer Type

- The past: two different integer types
- `int` -- unsigned 32- (or 64-bit) integers
  - Had `OverflowError`
- `long` -- unlimited in size except for VM
  - `L` or `l` designation for differentiation
- Starting in 2.2, both unified into single integer type
  - No overflow issues and still unlimited in size
  - `L` or `l` syntax deprecated in 3.0
- More information in PEP 237

Changing the Division Operator (/)

- Executive summary
  - Doesn't give expected answer for new programmers
  - Changed so that it does
- Terminology
  - Classic Division
  - Floor Division
  - True Division
- Controversy with this change:
  - Programmers used to floor division for integers

Classic Division

- Default 2.x division symbol (/) operation
- `int` operands: floor division (truncates fraction)
- One `float`: / performs float/"true" division
  - Result: `float` even if one operand an `int`
  - `int` "coerced" to other's type before operation
- Classic division operation

```
>>> 1 / 2
0
>>> 1.0 / 2.0
0.5
```

True Division

- Default 3.x division symbol (/) operation
- Always perform real division, returning a `float`
- Easier to explain to new programmer or child...
  - ...why one divide by two is a half rather than zero
- True division operation

```
>>> 1 / 2
0.5
>>> 1.0 / 2.0
0.5
```

Floor Division

- "New" division operator (//)... added in Python 2.2
- Always floor division regardless of operand types
- Floor division operation

```
>>> 1 // 2
0
>>> 1.0 // 2.0
0.0
>>> -1 // 2
-1
```

Accessing True Division

- To use true division in Python 2.2+:
  `from __future__ import division`
- True division default starting with 3.0
- Division -Q option
  - `old` -- always classic division
  - `new` -- always true division
  - `warn` -- warn on `int/int` division
  - `warnall` -- warn on all division operations
- More information in PEP 238

Update to Integer Literals

- Inspired by existing hexadecimal format
  - Values prefixed with leading `0x` (or `0X`)

  `0x80, 0xffff, 0XDEADBEEF...`
- Modified octal literals
- New binary literals
- Required in 3.0+
- Available in 2.6+ as transition tool
- More information in PEP 3127

New Binary Literals

- New integer literal format
  - Never existing in any previous version
  - Ruins some existing exercises :P
- Values prefixed with leading `0b`
  `0b0110`
- New corresponding BIF `bin`
- Modified corresponding BIFs `oct` & `hex`

Modified Octal Literals

- "Old" octal representation
  - Values prefixed with leading single `0`
  - Confusing to some users, especially new programmers
- Modified with an additional "o"
- Values prefixed with leading `0o`
- Python (1.x and) 2.x: `0177`
- Python 2.6+ and 3.x: `0o177`
- Modified corresponding BIFs `oct` & `hex`

Python 2.6+ Accepts Them All

```
>>> 0177
127
>>> 0o177
127
>>> 0b0110
6
>>> oct(87)
'0127'
>>> from future_builtins import *
>>> oct(87)
'0o127'
```

Iterables Everywhere

- Another 3.x theme: memory-conservation
- Iterators much more efficient
  - Vs. having entire data structures in memory
  - Especially objects created solely for iteration
  - No need to waste memory when it's not necessary
- Dictionary methods
- BIF (Built-in Function) replacements

Dictionary Methods

- `dict.keys`, `dict.items`, `dict.values`
  - Return lists in Python (1 and) 2
- `dict.iterkeys`, `dict.iteritems`, `dict.itervalues`
  - Iterable equivalents replace originals in Python 3
  - `iter` * names are deprecated
- If you really want a list of keys for `d`:

```
keys = list(d)
```

- If you really want a sorted list of keys for `d`:

```
keys = sorted(d)
```

- More information in PEP 3106

Updates to Built-Ins

- Changes similar to dictionary method updates
- Built-ins returning lists in 2.x return iterators in 3.x
  - `map`, `filter`, `xrange`, `zip`
- Other built-ins: new, changed, moved, or removed
  - In addition to iteration changes above
  - `reduce` moves to `functools` module
  - `raw_input` replaces and becomes `input`
  - More information in PEP 3111

*3.x Type Updates

- Integers
  - (Already discussed)
- Files
  - New `io` classes replace `file` object
  - More information in PEP 3116
- Dictionaries
  - (Method changes already discussed)
  - New dictionary comprehensions "dictcomps"
- Sets
  - New set comprehensions "setcomps"
- Tuples
  - Methods for the first time ever

Dictionary Comprehensions

- Inspired by `dict` () call passing in 2-tuples
  - Builds `dict` w/1st & 2nd tuple elements as key & value, resp.
- Now can use the equivalent but more flexible

```
{k: v for k, v in two_tuples}
```

- Example

```
>>> list(zip(range(5), range(-4, 1)))
[(0, -4), (1, -3), (2, -2), (3, -1), (4, 0)]
>>> {k: v*2 for k, v in zip(range(5), range(-
  4, 1))}
{0: -8, 1: -6, 2: -4, 3: -2, 4: 0}
```

Sets

- Set Literals

  `{1, 10, 100, 1000}`
  - Reflects similarity/relationship sets have with `dict` s
  - { } still represents an empty dict
  - Must still use `set` FF/BIF to create an empty set
- Set Comprehensions
  - Follow listcomp, genexp, and dictcomp syntax

```
>>> {10 ** i for i in range(5)}
{1000, 1, 10, 100, 10000}
```

  - Reminder: `dict` s and `set` s unordered (hashes)

Tuple Methods

- For the first time ever, tuples will now have methods
- Specifically `count` and `index`
- More convenient alternative to duplicating to a list
  - Just to find out how many times an object appears in it
  - Where it is in the list if it appears at all
- Logical since read-only ops on an immutable data type

Reserved Words

- Includes statements, constants, keywords
- Added
  - `as` , `with` , `nonlocal` , `True` , `False`
- Removed
  - `print` , `exec`

Built-Ins

- Functions & methods, but not factory functions
- BIFs
  - Added: `ascii` , `bin` , `exec` , `memoryview` , `next` , `print`
  - Moved: `reduce`
  - Removed: `apply` , `callable` , `cmp` , `coerce` , `execfile` , `intern` , `raw_input` , `reduce` , `reload` , `unichr` , `xrange`
  - Replaced: `map` , `filter` , `hex` , `input` , `oct` , `range` , `zip`
- BIMs
  - Added: New string and tuple methods
  - Replaced: Altered `dict` methods and `file` object (+methods) replaced by `io` classes (+methods)

Operator and Type Changes

- Operators
  - Removed
    - <> , ` `
- Types/Factory Functions
  - Added
    - `bytes` , `bytearray` , *range*
  - Removed
    - `basestring` , `buffer` , *file* , `long` , `unicode` , *xrange*

Various Porting/Migration Guides/Articles

- http://docs.python.org/3.0/whatsnew/3.0.html
- http://wiki.python.org/moin/PortingToPy3k
- http://lucumr.pocoo.org/2010/2/11/porting-to-python-3-a-guide
- http://diveintopython3.org/porting-code-to-python-3-with-2to3.html
- http://peadrop.com/blog/2009/04/05/porting-your-code-to-python-3/
- http://www.linuxjournal.com/content/python-python-python-aka-python-3

Recommended Transition Plan

- From "What's New in Python 3.0" document (see above)
- Wait for your dependencies to port to Python 3
  - Pointless to start before this except as exercise
- Start w/excellent coverage: ensure solid test suites
- Port to latest Python 2.x (2.6+)
- Use -3 command line switch (warns against incompats)
- Run `2to3` tool
- Make final fixes and ensure all tests pass
- How much time do I have? LOTS
- When is Python 2 going to be EOL'd? "COUPLE OF YEARS"

`2to3` Tool

- Examples of what it does
  - Changes backtick-quoted strings ` ` to `repr`
  - Converts `print` statement to function
  - Removes `L` long suffix
  - Replaces <> with !=
  - Changes `callable` (obj) to `hasattr` (obj, '__call__')
- Not a crystal ball... what it **doesn't** do
  - Stop using obsolete modules
  - Start using new modules
  - Start using class decorators
  - Start using iterators and generators
- http://docs.python.org/3.0/library/2to3.html

`3to2` Tool

- Refactors valid 3.x syntax into valid 2.x syntax
- (if a syntactical conversion is possible)
- http://bitbucket.org/amentajo/lib3to2/
- http://pypi.python.org/pypi/3to2
- http://us.pycon.org/2010/conference/posters/accepted/ (P9)

Python 2.6+

- Python 2.x not EOL'd (yet)... quite the opposite actually
- Remaining 2.x releases play a significant role
  - Because of compatbility issue
- 2.x & 3.x being developed in parallel
  - 2.6 & 3.0 were almost released at the same time(!)
  - Will live on for several more years
  - Keep 2.x alive for as long as it takes to migrate users
- 2.6: the first and most pivotal of such releases
  - First with specific 3.x features backported
  - Represents first time users can start coding against 3.x
- 2.6+: hybrid interpreters... they run
  - Some 1.x code, all 2.x code, some 3.x code

3.x Features Available in 2.6+

- New-style classes
- True division
- Changes to exception handling & raising idioms
- No integer overflow, integer literal changes
- `bytes` type and literals/strings (synonym for `str`)
- Class decorators
- Access to *some* 3.x BIF/BIM changes
- Access to some new modules/packages

Non-Autocompat Features

- Not all 3.x features backwards-portable to 2.x
- Not all work in parallel w/original 2.x functionality
- `print` must stay a statement
  - Must explicitly switch to BIF

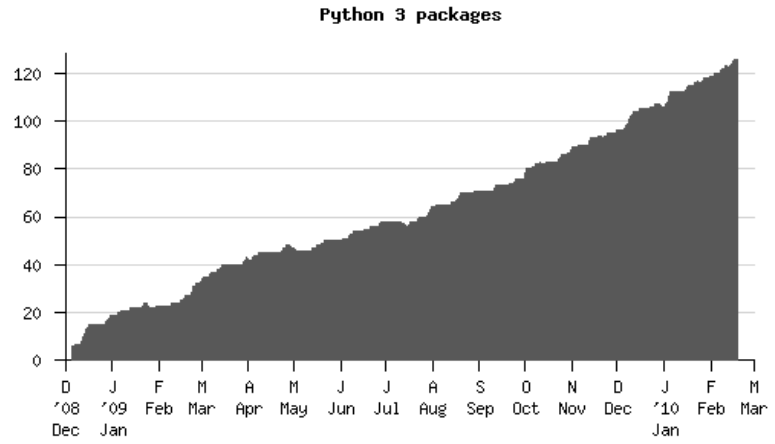
```
from __future__ import print_function
```

- Built-in functions w/new 3.x behavior must be imported
  - `ascii` , `filter` , `hex` , `map` , `oct` , `zip` , etc.
  - Import from `future_builtins` module

Python 3 Status

- Operating Systems (c=current, f=future, e=experimental)
  - http://oswatershed.org/pkg/python3.1
  - Arch, Debian, Fedora, Gentoo, OpenSuSE, Ubuntu
  - Also IUS/Rackspace RHEL/CentOS 5
- ~125 packages total (in PyPI) have been ported to 3.x
  - http://pypi.python.org/pypi?:action=browse&c=533&show=all
  - bsddb (bsddb3), coverage, cx_Oracle, Cython, gmpy, jsonlib, lxml, Markdown, py-postgresql, Pygments, Jinja2
- CherryPy, SWIG, mod_wsgi, PyWin32, Docutils

http://dev.pocoo.org/~gbrandl/py3pkgs.png



Futures

- 2.x developed in parallel with 3.x
- Remainder of 2.x will continue to gain 3.x features
- `2to3` improves w/every succeeding 2.x release
- 3.1 released in Jun 2009 (3.1.1 in Aug 2009)
  - Some new features (but more importantly...)
  - Major performance issue fixes
- 2.7 currently scheduled for Jun 2010
  - Will have 3.1-backported features
- 3.2 currently scheduled for Dec 2010 (PEP 3003 in effect)
  - No new language features and syntax
  - Should be released 18-24 mos after 3.1

Books and Learning Python

- Are all Python 2 books obsolete?
- If I want to learn Python, what version? *2* (for now)
  - Most codebases out there not ported yet
  - If for work, whatever version company using
  - If for hobby without prior code, Python 3 okay
- Most online/in-print books & tutorials: Python 2
  - There are some Python 3 books, but...
    - They're probably obsolete, e.g., 3.0
    - Not really all that useful (yet)
  - Best to stick with the best Python 2 books
  - Good chance of hybrid books in next 18-24 mos

Conclusion

- Python 3: the language evolving
  - It (the future) is here (but 2.x is *still* here!)
  - Backwards-incompatible but not in earth-shattering ways
  - Improve, evolve, remove sticky flaws
- To ease transition
  - 2.x sticking around for the near-term
  - 2.6+ releases contain 3.x-backported features
  - Use -3 switch and migration tools
- You will enjoy Python even more
  - But need to wait a little bit more to port

Recent+Upcoming Events

- Jun 9-11: PyCon Asia Pacific 2010, Singapore
  - http://apac.pycon.org
- May 10-12: Intro+Intermediate Python, SFO/San Bruno, CA
  - http://cyberwebconsulting.com
- Feb 17-21: PyCon 2010, Atlanta, GA, USA
  - http://us.pycon.org
- Nov 7: "Introduction to Python" ACM seminar, Cupertino, CA
  - http://www.sfbayacm.org/?p=852