Python 3000

(ACCU Conference / Oxford, UK / April 19, 2006)

Guido van Rossum

guido@python.org

guido@google.com

Talk Overview□

- Python 3000 Philosophy
  Python 3000 Process
- Python 3000 Features
- Questions

Python 3000 Philosophy

- Don’t design a whole new language!
- Fix early design bugs (many from 1990-1991!)
- Allow incompatible changes; within reason
- Get rid of deprecated ☐features
  - especially when that creates incompatibilities
    - e.g. int division
- Consider what’s the best feature going forward

- NB: Python 3000 = Python 3.0 = Py3k

Python 3000 Process

- Without some amount of process we're lost!
- Too many proposals competing for time
- Don't want to become the next Perl 6
- Examples of process/meta-questions:
  - when should we aim to release Python 3000?
  - how long will we maintain 2.x and 3.x side by side?
  - exactly how incompatible is Py3k allowed to be?
  - how will we migrate 2.x code to 3.x?
  - is anything competely off the table?
  - how to merge 3.0 features back into 2.x?

Release Schedule

- First alpha: not before next year
- Final release: probably a year after that
- May release 3.1, 3.2 relatively soon after

- Python 2.x and 3.x to be developed in parallel
  - Don't want users holding breath waiting for 3.0
  - Provide good support while 3.0 matures
  - 2.6: certainly (expected before 3.0)
  - 2.7: likely; may contain some 3.0 backports
  - 2.9 is as far as we'll go (running out of digits :-)

How Incompatible Can It Be?

- New keywords allowed
- dict.keys(), range(), zip() won’t return lists
  - killing dict.iterkeys(), xrange(), itertools.izip()
- All strings Unicode; mutable ‘bytes’ data type
- Binary file I/O redesign
- Drop <> as alias for !=
- But not (for example):
  - dict.keys instead of dict.keys()
  - change meaning of else-clause on for/while
  - change operator priorities

How to Migrate Code?

- Can’t do perfect mechanical translation
  - Many proposed changes are semantic, not syntactic
  - For example, in “f(x.keys())”
    - is x a dict?
    - does f() expect its argument to be a list?
  - Answers may be guessed by e.g. pychecker
    - but can’t always be decided with certainty
- Most likely approach:
  - use pychecker-like tool to do an 80+% job
  - create an instrumented version of Python 2.x that warns about doomed code (e.g. d.keys().sort())

“Five Is Right Out”

- From PEP 3099 (and from “The Holy Grail” :-)
- Python 3000 will not:
  - have programmable syntax / macros / etc.
  - add syntax for parallel iteration (use zip())
  - change hash, keys etc. to attributes
  - change iterating over a dict to yield key-value pairs
- In general we shouldn’t:
  - change the look and feel of the language too much
  - make gratuitous or confusing changes
  - add radical new features (can always do that later)

Python 3000 Features

- Scratching just the surface
- Few things ☐have been decided for certain
- Read PEP 3100 for a much larger laundry list
  - see python.org/dev/peps/

- Follow the list: python-3000@python.org
  - sign up at python.org/mailman/listinfo
- Read my blog: artima.com/weblogs/

Basic Cleanup

- Kill classic classes
- Exceptions must derive from BaseException
- int/int will return a float
- Remove last differences between int and long
- Absolute import by default
- Kill sys.exc_type and friends (exc_info() stays)
- Kill dict.has_key(), file.xreadlines()
- Kill apply(), input(), buffer(), coerce(), ...
- Kill ancient library modules

Minor Syntactic Changes

- exec becomes a function again
- kill \`x\` in favor of repr(x)
- change except clause syntax to
  ```
  except E1, E2, E3 as err:
  ```
  - this avoids the bug in
    ```
    except E1, E2: # meant except (E1, E2)
    ```
- [f(x) for x in S] becomes syntactic sugar for
  ```
  list(f(x) for x in S) # a generator expression
  ```
  - subtle changes in need for parentheses
  - x no longer leaks into surrounding scope
- kill raise E, arg in favor of raise E(arg)

range() Becomes xrange()

- range() will no longer return a list
  - I□t won’t return an iterator either
- It will return an “iterator well”
  - that’s just what xrange() is, anyway
- xrange() doesn’t support long values yet
  - we’ll fix that :-)
- Think of this as similar to dict views (see later)

zip() Becomes izip()

- That’s what it should’ve been all along
- But zip() was introduced in Python 2.0
  - which didn’t have iterators
- There’s no need for zip() to be a “view” etc.
  - 99% use case is parallel iteration
  - similar to enumerate()

lambda Lives!

- Last year, I said I wanted lambda to die
  - but I was open to a better version
- We still don’t have a better version
  - despite a year of trying
- So lambda lives!
  - let’s stop wasting time looking for alternatives
  - it’s as good as it gets; trust me!

String Types Reform

- bytes and str instead of str and unicode
  - bytes is a mutable array of int (in range(256))
  - encode/decode API? bytes(s, "Latin-1")?
  - bytes have some str-ish methods (e.g. b1.find(b2))
  - but not others (e.g. not b.upper())
- All data is either binary or text
  - all text data is represented as Unicode
  - conversions happen at I/O time
- Different APIs for binary and text streams
  - how to establish file encoding? (Platform decides)

New Standard I/O Stack

- C stdio has too many problems
  - don’t know how many bytes are buffered
  - write after read unsafe (libc doesn’t promise safety)
  - Windows “text” mode nightmares
  - Universal newlines hacked in (for input only)
- bytes/str gives an opportunity to fix all this
  - learn from Java streams API
  - stackable components: buffering, encoding, ...
  - see sandbox/sio/sio.py for an early prototype

Print Becomes a Function

- print x, y, x becomes print(x, y, z)
- print >>f, x, y, z becomes print(x, y, z, file=f)
- Alternative(s?) to skip the space/newline
  - printf(format, x, y, z)?
  - printraw(x, y, z)? or print(x, y, z, raw=True)?
- Why not f.print(x, y, z)?
  - because that requires support in every file type
- Why change at all?
  - print-as-statement is a barrier to evolution of your program *and* of the language (see link in PEP 3100)

Dict Views Instead of Lists

- dict.keys() currently returns a list
  - that’s expensive if dict is large
    - so we introduced dict.iterkeys()
      - but now the API becomes too large
- Proposal (affects dicts only!):
  - dict.keys() and dict.items() will return a set view
  - dict.values() will return a bag (multiset) view
  - a view is a wrapper object
  - can delete from but not add to a view
    - modifies the dict accordingly
  - iterating over a view creates a new iterator object

Drop Default Inequalities

- The default implementations of <, <=, >, >= aren’t very useful (compare by address!)
- Let these raise TypeError instead

- NB: the default implementations of ==, != should remain (comparing identity is useful!)

Generic / Overloaded Functions

- A single callable with multiple implementations
- Contains a registry indexed by call signature
  - i.e. tuple(type(X) for X in args)
  - this maps to an implementation for those types
  - dispatches on type of all arguments at once!
  - (solvable) complications: inheritance
- Use a decorator for registration calls
- This can replace adaptation too!
  - make the protocol object an overloaded function
  - spell adapt(X, P) / P.adapt(X) as P(X)

Got Questions?