ZOPE CORPORATION

State of the Python Union

OSCON, July 24, 2002

Guido van Rossum
Director of PythonLabs at Zope Corporation

guido@zope.com
guido@python.org

Recent Releases

- Python 2.2
  - iterators!
  - generators!!
  - new-style classes!!!
  - and too much to summarize here...
- Python 2.1.3
  - bug fix release for 2.1; focus on stability
- Python 2.2.1
  - bug fix release for 2.2; ditto
- What's with this stability focus...? (...see later!)

Python Organizations

- Python Software Foundation
  - www.python.org/psf
  - US non-profit for research and education
  - owns the current Python copyright
  - looking for donations and sponsors
- Python Business Forum
  - www.python-in-business.org
  - EU non-profit for businesses based on Python
  - plans:
    - Python in a Tie
    - Compile farm
- Python Secret Undergr

Python in a Tie

- Result of stability discussion on c.l.py
- Plan:
  - pick a release and maintain it for 18+ months
  - bleeding edge development releases continue
- Purpose:
  - have a reliable target for commercial users
  - stability more important than latest features
- Which release will wear the tie?
  - Python 2.2!

PBF Compile Farm

- Joint venture of PBF and Lysator
    - Lysator: oldest Swedish computer (student) society
  - Lysator owns a very diverse hardware farm
  - PBF provides motivation, funding
- Goals:
  - testing on many platforms
    - Python-in-a-Tie as well as bleeding edge code
    - core Python as well as 3rd party extensions
  - build binary releases for Python-in-a-Tie
    - hopefully "sumo releases"

Python Conferences in 2003

- EuroPython will probably be repeated
- Python11 will be at OSCON 2003
- Yet Another Python Conference (YAPyC)
  - these plans are ***tentative***
  - co-organizers: YAS and PSF
  - registration fee: <= $150; expect 300 attendees
  - time: January/February 2003
  - location: downtown Washington, DC (GWU)
  - format: workshop-like
  - *looking for volunteers to help organize!*

Python.Org HTTP Statistics

- May 2002
  - 7.9M HTTP requests from 257K hosts
    - 291K hits for "/"
  - 52K downloads of Python 2.2.1
    - about 70% Windows installer
- Feb 2001
  - 5.5M HTTP requests from 164K hosts
    - 212K hits for "/"
  - 23K downloads of Python 2.0
    - over 70% Windows installer

Controversy of the Year

- Yearly recap of a recent flame war
- This year's topic:
  - to bool or not to bool

Why bool()?

- I always regretted having left it out
- If it's not built-in, people define their own
- Explicit is better than implicit: "return True"
- A bool result is distinguished in output

  -
    ```
    >>> x == y
    True
    >>>
    ```
- "bool(x)" normalizes Booleans
  - was "not not x"
- RPC tools can special-case Booleans

Why Not bool()?

All misunderstandings (in my opinion)

- Will "if x:" require x to be a bool? (***Never!***)
- Some people write "if x == True:" (Yuck)
- "No function should return a bool" (Huh?)
- It's confusing to teach
  - I don't buy this:
    - You need to explain the Boolean concept anyway
    - You need to pick representatives anyway
    - You need to explain that (almost) all types have a Boolean interpretation anyway

How To bool()?

- bool is a new built-in type
- True and False are the only values
  - singletons like None ("dualtons"?)
- Cannot be subtyped
- bool is a subtype of int, for compatibility
  - True + 1 == 2
  - True == 1
  - str(True) == "True" # The only incompatibility
  - will stay this way in Python 3.0
    - it's useful and harmless

Lessons Learned

- It's a growth opportunity!
- Everything is controversial
  - QOTY: "When a group becomes large enough there are no uncontroversial topics any more."
    - Erik van Blokland (in personal email)
- Anticipate potential misunderstandings
  - explain in advance
  - I *thought* the PEP was clear - not so :-(
- In the end, do what you think is right
  - can't please everyone

The Future: Python 2.3

- No new syntax, except yield w/o __future__
- Library focus, e.g.:
    - support extended slices, e.g. "dlrow olleh"[::-1]
    - bool() and enumerate()
    - more callable types; basestring
    - import from zip files
    - timeouts for sockets
    - logging module
    - gnu_getopt and option parser modules
    - new compiler package
    - berkeleydb module
- Fixing bugs

PendingDeprecationWarning

- Discourage certain things in new code
  - But don't warn about them normally
  - Use:
    - warnings.warn("your message here", PendingD...)
      - No output by default (unlike other warnings)
  - To see the warnings:
    - python -Wall::PendingDeprecationWarning
- ***Potential*** examples:
  - string module (use string methods)
  - types module (use built-in type names)
  - has_key() (use 'in' operator)

2.3 Release Schedule

- Surprise: we have none!
- Focus on feature completeness, not dates
- Hope: alpha soon, final before 2002 ends
- See PEP 283 for details

Pace of Change

- Users demand a stop to all new features
- Except for their personal favorite
  - this contradiction seems unavoidable
- What do do about this?
- Is Python-in-a-Tie sufficient?

"Would You Rather..." [1]

– Learn more syntax; *or*
– use a library module?

– Understand a deep concept; *or*
– live with fuzzy rules?

– Fix design mistakes; *or*
– be backwards compatible?

[1] http://barry.wooz.org/poems/wyr.html

Example: String Interpolation

- Problem: % interpolation is cumbersome
  - print x, "+", y, "=", x+y
  - "%s + %s = %s" % (x, y, x+y)
  - "%(x)s + %(y)s = %(z)s" % vars()
  - str(x) + " + " + str(y) + " = " + str(x+y)
- The print form is most readable
  - but not general enough (doesn't return a string)
- The other forms leave a lot to desire
- This is a very common need
  - so a clean solution would be nice; hence PEP 292

Solutions Explored

- Solution 1: "$foo".sub()   # runtime
  - `"$x + $y = $z".sub()`
- Solution 2: x"$foo"   # compile-time
  - x"$x + $y = $(x+y)"
- Alternatives: %x, `x`, <<x>>, ?x?, @x@
- Solution 3: *func*(foo) # no new notation
  - *func*(x, " + ", y, " = ", x+y)
- None of these are satisfactory!
- Even more issues when considering i18n

Why Is This Important To Me?

- Preserve the "sweet spot". Python is:
  - small enough to learn and remember easily
  - convenient for expressing common patterns
  - powerful for advanced usage
- Improving 2 or 3 often threatens 1
- Compatibility requirement prevents throwing away failed experiments
  - like \`back ticks\` or lambda
- No obvious solution

Python 3.0

- No release schedule either :-)
- Not within two years
- Question: what to focus on???
- Zope 3 experience may be relevant
  - Rebuild from scratch
    - Refactor mercilessly during development
    - No concern for backwards compatibility
      - But learn from past: good ideas, bad ideas
    - Use coding "sprints"
  - Later, add compatibility (Zope 3x -> Zope 3)
  - Or: Later, merge best features back into 2.x

Open Mike

It's your turn!