Python 2 vs. Python 3
A retrospective

Guido van Rossum
guido@python.org
Hackers 2013

Why a new version

- "Python warts" and my own regrets
  - some syntactic, others semantic
- Can't always f x in backwards compatible way
  - the right spelling has the wrong meaning
- Too many ways to do it :-)
  - e.g. old- and new-style classes
- The Unicode mess
  - 8-bit strings are ambiguous and may be toxic

Why *not* a new version

- People don't like change
  - this appears to be a universal trait
- People positively **hate** incompatible changes
  - especially bad for dynamic languages
- People don't understand the reasons
  - or they don't care

So we did it anyway

- Core developers all on board
  - No new features in 2.7, no 2.8 release ever
- Community is excited
  - large projects moving (e.g. Django, NumPy)
- Expect another 5 years
  - we're halfway through!
- Never again this way
  - the future is static analysis and annotations

Overview of changes

- "Do all the incompatible stuff at once"
- Long-planned house cleaning
- Unicode redesign
- Other design warts
- "Premium features"

House cleaning

- Kill old-style classes
- Kill separate long type
- Drop "raise E, value" —> raise E(value)
- Drop backticks: `x` -> repr(x)
- Kill <> operator —> !=
- Make exec a function once again
- Rename __builtin__ module to builtins

More house cleaning

- None, True, False become keywords
- Forbid "from mod import *" inside function
- Rename func_name —> __name__, etc
- Numeric tower
- Rename .next() —> .__next__()

Small improvements

- No default ordering: 1 < 'x' raises TypeError
- except E, e: —> except E as e:
- Absolute import by default
- New octal notation: 0o755
- map(), filter(), zip() become iterators
- range() mutates into what was xrange()
- dict.keys() returns a "view", not a list

More small improvements

- Improved super()
- Set notation {1, 2, 3} (but {} is an empty dict)
- Set and dict comprehensions
  - `{x**2 for x in range(10)}`
  - `{x: x**2 for x in range(10)}`
- Exception chaining; e.__traceback__
- Metaclass improvements

Why reduce() must die

- "So now reduce(). This is actually the one I've always hated most, because, apart from a few examples involving + or *, almost every time I see a reduce() call with a non-trivial function argument, I need to grab pen and paper to diagram what's actually being fed into that function before I understand what the reduce() is supposed to do. So in my mind, the applicability of reduce() is pretty much limited to associative operators, and in all other cases it's better to write out the accumulation loop explicitly."
  - me, in 2005

No, really!

- "One-liner word-wrap function" recipe :-(

```
def wrap(text, width):
    return reduce(lambda line, word, width=width: '%s%s
%s' %
                  (line,
                   ' \n'[(len(line)-line.rfind('\n')-1
                         + len(word.split('\n',1)[0]
                              ) >= width)],
                   word),
                  text.split(' ')
                 )
```

Why int/int should return float

- ```
  def avg(x, y):
      return (x+y) / 2
  ```
- `avg(3.1, 3.3) -> 3.2`
- `avg(3, 4) -> 3`

- Use // or divmod() for truncating division
  ```

Why print() must be a function

- You all know:
  - print 'height=', height, 'width=', width
- But do you know what these do?
  - print word,
  - print >>sys.stderr, 'invalid input'
- These are hard!
  - feature request: option to force flush()
  - edit task: change all prints to log calls

What was wrong with Unicode

- ```
  def full_name(first, last):
      return first + u' ' + last
  ```
- ```
  full_name('Guido', 'van Rossum')
  ```
- ```
  full_name('Charles-François',
  'Natali')
  ```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in full_name
UnicodeDecodeError: 'ascii' codec can't decode
byte 0xc3 in position 12: ordinal not in
range(128)
```
```

Nonlocal variables

```
def make_counter(incr=1):
    count = 0
    def counter():
        nonlocal count
        count += incr
        return count
    return counter
```

Performance

- Python 3.0 and 3.1 were pretty bad
- 3.3 is on a par with 2.7; uses less memory
  - dict-sharing keys
  - adaptive-width Unicode string representation
  - across the board small improvements

Speculative features

- Function annotations
  - `def foo(x: int, y: 'whatever') -> MyClass:`
    `...`
  - cool, but not much use yet
  - maybe mypy-lang.org will change that
- PEP 380: yield from
  - I'm building Tulip (asyncio) to use this

Ho-hum changes

- New formatting "total {} bugs".format(nbugs)
  - Considered too verbose; % will never go away
- Stdlib restructuring (not enough of it, really)
- Unify list comps and generator exprs
  - speed of list comps remains an issue
- Drop unbound methods
  - C.foo is now a plain function (breaks stuff)

More ho-hum

- Bytes are ints, e.g. b'abc'[0] == 97
- WSGI 1.0.1

Things we missed

- Reverse slices are a mess:

  'abcde'[4:0:-1] == 'edcb'

- Float mod with negative arg loses precision:

  1e-10 % -1 == -0.9999999999

- Some APIs still use str instead of bytes

  e.g. XML, HTTP

- unrecognized \ in strings, e.g. '\c' == '\\c'

Things still unsolved

- Multi-line lambda
  - problem is mixing indentation and expressions without breaking backward compatibility

Another Python 3 talk

- "Python 3.3: Trust Me, It's Better Than Python 2.7" by Dr. Brett Cannon (PyCon 2013)
  - https://speakerdeck.com/pyconslides/python-3-dot-3-trust-me-its-better-than-python-2-dot-7-by-dr-brett-cannon