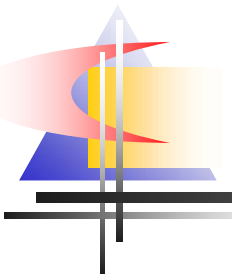
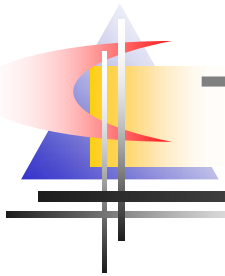


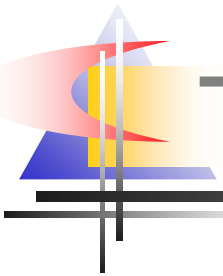
What's new in Python 2.3

Alex Martelli



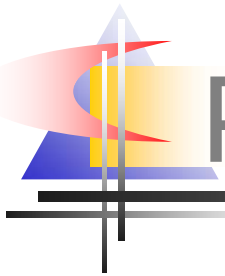
This talk's audience...:

- "fair" to "excellent" grasp of Python and Python 2.2
- "none" to "fair" grasp of Python 2.3
- wants to learn more about: Python 2.3 / upgrading benefits and issues



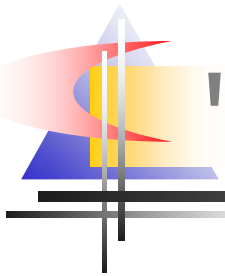
This talk's outline...:

- generalia (performance, ...)
- enhancements to 2.3's built-ins
- standard library: enh. modules
- enhancements to: importing, threading, pickling, distutils, internals (memory alloc, gc)
- new modules ('major'/'other')



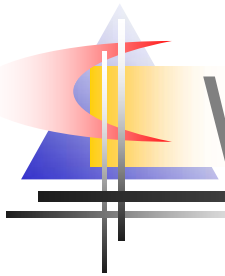
Python 2.3 roadmap

- Beta 1 was out in late April
- Beta 2 was out in July
- final 2.3 will be out in August



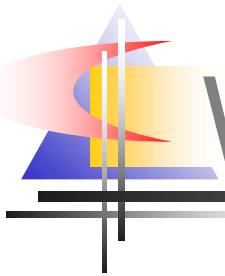
"Should I upgrade"...?

- Yes!
- the Python language is very stable
- you can keep programming to 2.2 and get 2.3's benefits
- about 15%-20% extra performance!
- your programs won't break!
- check your programs at once...!!!



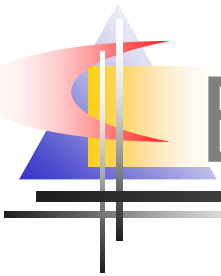
What benefits in upgrading?

- Performance
- bug fixes (more than in 2.2.3!)
- generators always enabled, w/o
  `from __future__ import generators`
- minor enhancements to built-ins
  - slicing,bool,in,file,dict,enumerate,sum
- many standard library enhancements
- tools (timeit, new IDLE, ...)



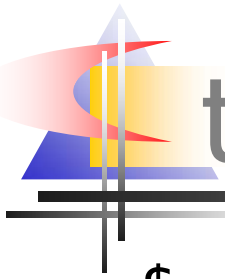
What issues with upgrading?

- new bugs? [current Cygwin troubles]
- generators always enabled, w/o `from __future__ import generators`
- int/long unification is proceeding
- non-ASCII sources
- sundry minor differences
  - unlikely to bite: assigning to None, ...
- no Bastion / rexec
  - were unsafe in 2.2 (not in 2.2.3 either!)



Extra performance

- new module timeit lets you measure micro-performance accurately
- timeit.py runs fine under 2.2, too
- a precious little new tool
- so, don't wait: download it now!
- `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/*checkout*/python/python/dist/src/Lib/timeit.py?rev=1.9&content-type=text/plain` (if you don't have 2.3...)



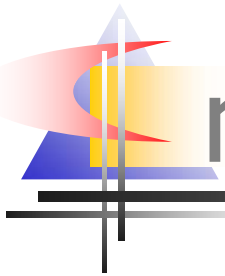
timeit.py example, 2.2 vs 2.3

```
$ python2.2 -O \
> /usr/local/lib/python2.3/timeit.py \
> '11223344556677889 *
> 11223344556677889'
1000000 loops, time: 1.151 usec
$ python2.3 -O \
> /usr/local/lib/python2.3/timeit.py \
> '11223344556677889 *
> 11223344556677889'
1000000 loops, time: 0.665 usec
```

and by the way...: x*x vs x**2

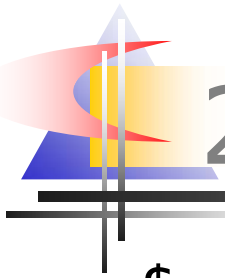
```
$ python2.2 -O \
> /usr/local/lib/python2.3/timeit.py \
> '11223344556677889 ** 2'
1000000 loops, time: 2.727 usec
$ python2.3 -O \
> /usr/local/lib/python2.3/timeit.py \
> '11223344556677889 ** 2'
1000000 loops, time: 1.349 usec
```

(this has long been known to users of languages with ** operators, such as Fortran -- x*x is faster!)



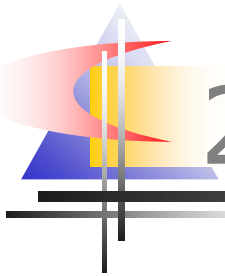
minor language-level changes

- MRO of new-style classes enhanced
- names of extension types now include the module's name
- `__name__` & `__bases__` can now be re-bound on new-style classes (always could be on classic ones)
- `super` built-in has been enhanced



2.3: extended slicing, bool, str 'in'

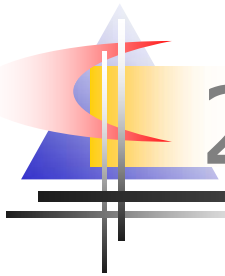
```
$ python2.3 -c 'print "ciao"[::-1]'
oaic
$ python2.3 -c 'print "arrivederci"[::2]'
arvdri
$ python2.2 -c 'print True, False'
1 0
$ python2.3 -c 'print True, False'
True False
$ python2.3 -c 'print "riv" in "arrivo"'
True
```



2.3: extended-slicing assignment

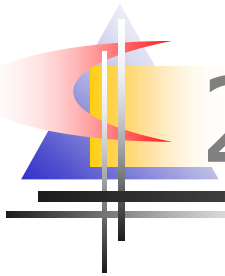
```
>>> x = list('arrivederci')
>>> ''.join(x[::2])
'arvdri'
>>> ''.join(x[::-2])
'irdvra'
>>> x[::2] = x[::-2]
>>> ''.join(x)
'irrideverca'
```

Note: assignment to an extended slicing cannot change the length of the list



2.3: 'slice' built-in type

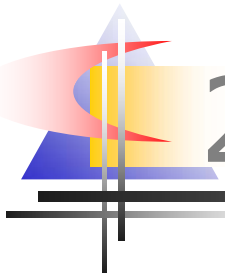
```
>>> # Can index sequences (not dicts):
>>> a = slice(1, 6, 2)
>>> print 'capperi'[a]
apr
>>> # has 'indices' method as helper to
>>> # implement your own sequences' slicing
>>> print a.indices(9), a.indices(4)
(1, 6, 2) (1, 4, 2)
>>> range(*a.indices(4))
[3]
```



2.3 enhancements to 'file'

- new opening mode 'U' ('universal newlines' -- accepts \r, \n, \r\n)
- a file IS, not just HAS, an iterator

```
# sample task: "emit a file, skipping its
# first lines up to the first white one"
f = open('some.txt', 'U')
for line in f:
    if line.isspace(): break
for line in f:
    print line,
```



2.3 enhancements to 'list'

`list.insert(i, value)` finally fixed

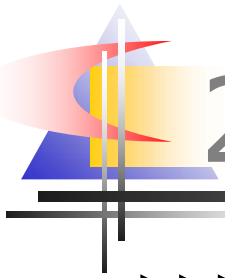
when i < 0 ...:

```
x = range(5)
x.insert(-1, 9)
print x
# in 2.2 emitted: [9, 0, 1, 2, 3, 4]
# in 2.3 emits: [0, 1, 2, 3, 9, 4]
```

list.sort made faster and stable

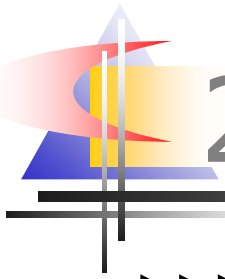
Changes to 'int' ('long' unif.)

```
$ python2.2 -c 'print int(123456*123456)'
warning: integer multiplication
OverflowError: long int too large to convert
$ python2.3 -c 'print int(123456*123456)'
15241383936
$ python2.3 -c 'print 0xFFFFFFFF'
<string>:1: FutureWarning: hex/oct constants
  > sys.maxint will return positive values in
  Python 2.4 and up
-1
```



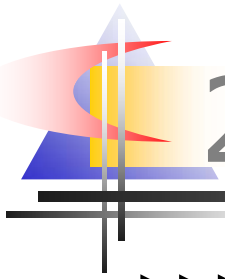
2.3 enhancements to 'dict'

```
>>> # two new ways to build a dict:
>>> dict(a=23,b=45,c=67)
{'a':23, 'b':45, 'c':67}
>>> dict.fromkeys(range(4), 'ho')
{0:'ho', 1:'ho', 2:'ho', 3:'ho'}
>>> # one new way to pull a dict apart:
>>> dd = dict(a=23,b=45,c=67)
>>> dd.pop('b')
45
>>> dd
{'a':23, 'c':67}
```



2.3 dict.pop: optional 2nd arg

```
>>> dd = dict(a=23,b=45,c=67,d=89)
>>> dd.pop('z')       # raises KeyError
>>> dd.pop('z', 11)   # dd unmodified
11
>>> dd.pop('c', 11)   # dd['c'] removed
67
>>> dd
{'a':23, 'b':45, 'd':89}
```



2.3 new built-in: enumerate

```
>>> for x in enumerate('ciao'): print x,
...
(0,'c') (1,'i') (2,'a') (3,'o')
>>> # just like...:
>>> def enumerate(iterable):
...     index = 0
...     for item in iterable:
...         yield index, item
...         item = item + 1
```

2.3 new built-in: sum

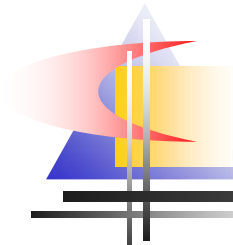
```
>>> sum(range(5))
10
```

Like `reduce(operator.__add__,range(5))`, but:

```
$ python timeit.py -s'x=range(5)' \
> 'sum(s)'
1000000 loops, best of 3: 1.64 usec perloop
$ python timeit.py -s'x=range(5)' \
> 'reduce(operator.__add__, s)'
1000000 loops, best of 3: 4.08 usec perloop
```

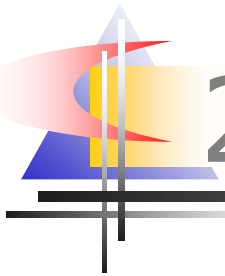
Numbers only (not strings: use ''.join(s) instead!)

There won't be a 'product' built-in to match!



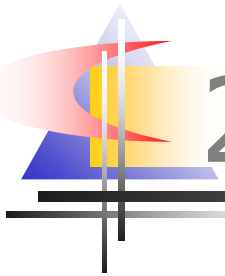
2.3 non-ASCII sources

```
$ cat a.py
print "olá"
$ python2.3 a.py
sys:1: DeprecationWarning: Non-ASCII
character '\xe1' in file a.py on line 1 [&c]
olá
$ cat b.py
# -*- coding: Latin-1 -*-
print "olá"
$ python2.3 b.py
olá
```



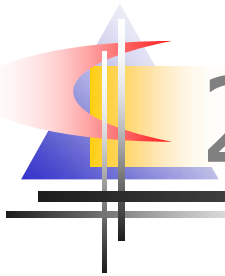
2.2/2.3: restricted execution

- rexec & Bastion not safe in 2.2
- new-style classes not guarded
- both modules ripped out of 2.3 and 2.2.3 -- currently no safe way to execute untrusted code
- Some hope for the future...: http://www.procoders.net/download.php?fname=SandBox.py



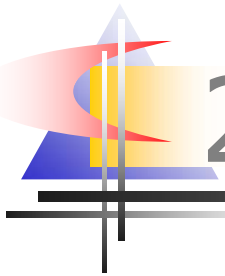
2.3: enhanced modules [1]

- socket: s.settimeout(t)
- array: added 'u', +=, *=
- minidom: d.toxml(encoding='...')
- random: random.sample(popul,k)
  - module now uses Mersenne Twister algorithm
- math: math.degrees, math.radians
- bsddb: supports latest BerkeleyDB
- pyexpat: parsers can buffer_text



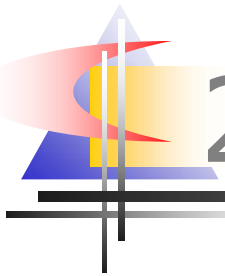
2.3: enhanced modules [2]

- shutil: shutil.move(src, dest)
- readline: new history functions
- time: time.strptime is now cross-platform, pure-Python, solid
- UserDict: DictMixin class helps you make your mappings more canonic
- codecs: register_error, lookup_error, new 'backslashreplace' and 'xmlcharrefreplace' strategies



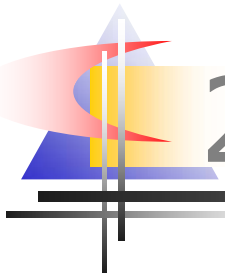
2.3: import enhancements

- new `sys` attrs: `meta_path` lists importers tried before the `path`, `path_hooks` lists importer-factories tried on each dir on the `path`, `importer_cache` caches those
- an *importer*'s method `find_module` returns a *loader* object, w/method `load_module`
- already used (via `zipimport`) to import modules from any zipfile



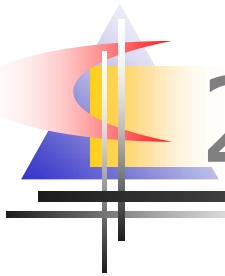
2.3: threading enhancements

- new modules dummy_thread and dummy_threading
  - applications of Null Object DP
  - same API as thread and threading, but no-operation implementations
  - handy for non-threading platforms
- Tkinter threading enhanced
  - cross-thread access now either works,
  - or raises a clear exception
- interrupting other threads (w/C API)



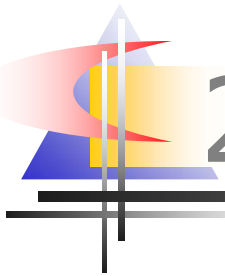
2.3: pickling enhancements

- pickling now takes a *protocol* argument rather than a *binary* flag
- protocol **0** is the old text one
- protocol **1** the old binary one
- new protocol **2** is more efficient
- new support for `__getstate__`, `__setstate__`, `__getnewargs__`
- unpickling officially declared unsafe (**don't** unpickle untrusted data!)



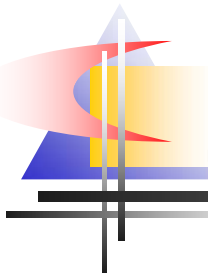
2.3: distutils enhancements

- metadata now supported -- can be registered at `www.python.org/pypi`
  - new `classifiers` attribute w/*trove* strings
- `Extension` class can have `depends`
- distutils now checks some current environment variables, so you can override Python's configuration settings (CC, CPP, CFLAGS, LDFLAGS)



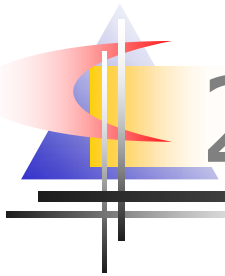
2.3: internals enhancements

- `Pymalloc` is now on by default
- memory allocation routines now cleanly classified into 2 families:
  - PyMem_{Malloc, Realloc, Free}: raw memory
  - PyObject_{..., New, NewVar, Del}: objects
- extra debugging; `pymemcompat.h`
- garbage collector interface changed
- can easily build `libpython2.3.so`



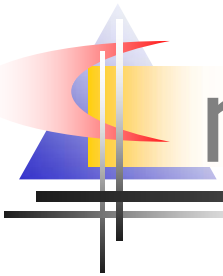
2.3: "major" new modules

- datetime
- sets
- heapq
- itertools
- logging



2.3: other new modules

- bz2: like gzip, but for the bzip2 library
- optparse: like getopt, but with more power and flexibility
- textwrap: reformat text into paragraphs "intelligently"
- platform: platform-version info
- tarfile: read/write **.tar** archive files
- csv: r/w "comma-separated" files



new module: datetime

- date/time arithmetic and formatting
- always Gregorian-calendar
- no "leap seconds" support
- `datetime` objects & timezones/dst:
  - **naive** `datetime` object: ignores tz/dst
  - **aware** `datetime` object: has `x.tzinfo` attribute (of `datetime.tzinfo` abstract class: you must supply concrete subclass)
  - `date` and `timedelta` objects don't care

datetime.timedelta

- stores normalized integer `days`, `seconds`, `microseconds`
  - `resolution = 1 microsecond`
  - `range = +/- 1 billion days`
- may also build with `weeks`, `hours`, `minutes`, `milliseconds`, `floats`
  - `timedelta` normalizes and rounds
- supports limited "sensible" arithmetic, comparisons, hashing, pickling

datetime.date

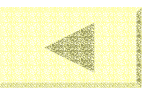
- stores `year`, `month`, `day`
  - `range = years 1 - 9999`
- class method (factories) `today`, `fromtimestamp`, `fromordinal`
- supports limited "sensible" arithmetic, `str`, comparisons, hashing, pickling
- many instance methods (accessors) e.g. `toordinal`, `weekday`, ...

datetime.time

- stores `hour`, `minute`, `second`, `microsecond`, `tzinfo`
- class method (factories) `today`, `fromtimestamp`, `fromordinal`
- supports `str`, comparisons, hashing, pickling
- many instance methods (accessors) e.g. `strftime`, `isoformat`, ...

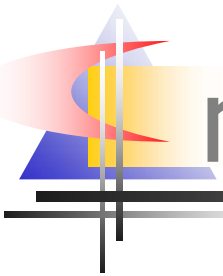
datetime.datetime

- subclasses `date`, adds a `time`
- class method (factories) `now`, `fromtimestamp`, `combine`, ...
- supports limited "sensible" arithmetic, `str`, comparisons, hashing, pickling
- many instance methods (accessors) e.g. `date`, `time`, `timetz`, ...



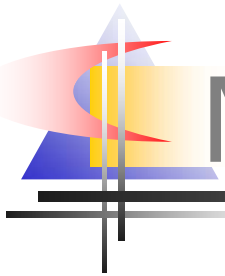
datetime.tzinfo

- abstract class -- your app must supply suitable concrete subclasses if you want to handle timezones, dst &c
- methods you may have to override:
  - utcoffset(self, date) offs from UTC (minutes)
  - dst(self, date) DST adjustment (minutes)
  - tzname(self, date) name string for timezone
- `tz.utcoffset(d)-tz.dst(d)` should be the same for any date d



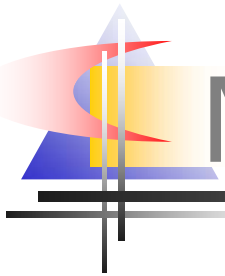
new module: sets

- sets of hashable elements
- `x in s, len(s), for x in s`
- unordered collections, **not** sequences
  - no indexing, no slicing
- class `Set` is mutable (-> no `hash`)
- class `ImmutableSet` is immutable (-> hashable, allows "set of sets")
- abstract base class `BaseSet`



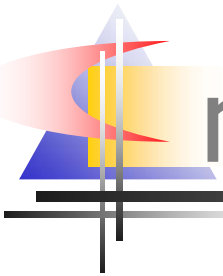
Methods common to all sets

- `issubset, issuperset`
- `union, intersection, difference (aka | & -)`
- `symmetric_difference (aka ^)`
- `copy` (shallow copy)
- `if x in s,for x in s,len(s)`
- **rich** comparisons (**not** `cmp`!)
  - `s<=t` means `s.issubset(t)`



Methods of mutable sets

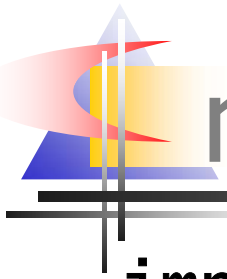
- `union_update`, etc (aka `|=` `&=` `-=` `^=` -- in-place updates)
- `add`, `remove` (may raise `KeyError`), `discard` (won't raise)
- `clear`, `pop`, `update`



new module: heapq

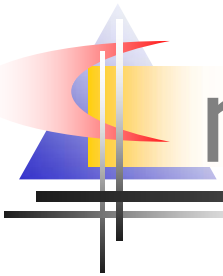
- functions to treat a list as a priority-queue ("heap")
- `heapify(L)` (O(N), in-place)
- `heappush(L,x)`, `heappop(L)`
- `heapreplace(L, x)` -- like:

```
r = heappop(L)
heappush(L, x)
return r
```



making a class from heapq

```
import heapq
class PriorityQueue(object):
    def __init__(self, seq=[]):
        self.l = list(seq)
        heapq.heapify(self.l)
    def __len__(self):
        return len(self.l)
    def push(self, x):
        heapq.heappush(self.l, x)
    def pop(self):
        return heapq.heappop(self.l)
```



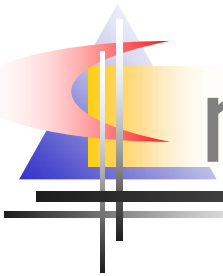
new module: itertools

- "iterator building-block" generators
- `chain(*its), count(n=0), cycle(it), dropwhile(p, it), ifilter(p, it), imap(f, *its), islice(it, ...), izip(*its), repeat(x, times=None), starmap(f, it), takewhile(p, it)`
- all highly composable & efficient

an itertools speed surprise

For the simplest iteration's overhead...:

```
$ python timeit.py \
> 'for x in range(9999): pass'
1000 loops, best of 3: 1.41e+03 usec
$ python timeit.py \
> 'for x in xrange(9999): pass'
1000 loops, best of 3: 1.11e+03 usec
$ python timeit.py -s'import itertools' \
> 'for x in itertools.repeat(0,9999): pass'
1000 loops, best of 3: 921 usec
```



new module: logging

- class Logger instances are "loggers", with hierarchical structured-names
- each logger and message has a level (debug, info, warning, error, critical, + optional custom ones) for filtering
- unfiltered msgs are LogRecord insts
- dispatched via handlers (filter + disposition: file, socket, mail, ...)

a dirt-simple use of `logging`

```
import logging
...
if user not in knownUsers:
  logging.warning(
    "User %s not known"
    % user)
```