What's New in Python 2.0

Author: A.M. Kuchling and Moshe Zadka

Introduction

A new release of Python, version 2.0, was released on October 16, 2000. This article covers the exciting new features in 2.0, highlights some other useful changes, and points out a few incompatible changes that may require rewriting code.

Python's development never completely stops between releases, and a steady flow of bug fixes and improvements are always being submitted. A host of minor fixes, a few optimizations, additional docstrings, and better error messages went into 2.0; to list them all would be impossible, but they're certainly significant. Consult the publicly-available CVS logs if you want to see the full list. This progress is due to the five developers working for PythonLabs are now getting paid to spend their days fixing bugs, and also due to the improved communication resulting from moving to SourceForge.

What About Python 1.6?

Python 1.6 can be thought of as the Contractual Obligations Python release. After the core development team left CNRI in May 2000, CNRI requested that a 1.6 release be created, containing all the work on Python that had been performed at CNRI. Python 1.6 therefore represents the state of the CVS tree as of May 2000, with the most significant new feature being Unicode support. Development continued after May, of course, so the 1.6 tree received a few fixes to ensure that it's forward-compatible with Python 2.0. 1.6 is therefore part of Python's evolution, and not a side branch.

So, should you take much interest in Python 1.6? Probably not. The 1.6final and 2.0beta1 releases were made on the same day (September 5, 2000), the plan being to finalize Python 2.0 within a month or so. If you have applications to maintain, there seems little point in breaking things by moving to 1.6, fixing them, and then having another round of breakage within a month by moving to 2.0; you're better off just going straight to 2.0. Most of the really interesting features described in this document are only in 2.0, because a lot of work was done between May and September.

New Development Process

The most important change in Python 2.0 may not be to the code at all, but to how Python is developed: in May 2000 the Python developers began using the tools made available by SourceForge for storing source code, tracking bug reports, and managing the queue of patch submissions. To report bugs or submit patches for Python 2.0, use the bug tracking and patch manager tools available from Python's project page, located at https://sourceforge.net/projects/python/.

The most important of the services now hosted at SourceForge is the Python CVS tree, the version-controlled repository containing the source code for Python. Previously, there were roughly 7 or so people who had write access to the CVS tree, and all patches had to be inspected and checked in by one of the people on this short list. Obviously, this wasn't very scalable. By moving the CVS tree to SourceForge, it became possible to grant write access to more people; as of September 2000 there were 27 people able to check in changes, a fourfold increase. This makes possible large-scale changes that wouldn't be attempted if they'd have to be filtered through the small group of core developers. For example, one day Peter Schneider-Kamp took it into his head to drop K&R C compatibility and convert the C source for Python to ANSI C. After getting approval on the python-dev mailing list, he launched into a flurry of checkins that lasted about a week, other developers joined in to help, and the job was done. If there were only 5 people with write access, probably that task would have been viewed as "nice, but not worth the time and effort needed" and it would never have gotten done.

The shift to using SourceForge's services has resulted in a remarkable increase in the speed of development. Patches now get submitted, commented on, revised by people other than the original submitter, and bounced back and forth between people until the patch is deemed worth checking in. Bugs are tracked in one central location and can be assigned to a specific person for fixing, and we can count the number of open bugs to measure progress. This didn't come without a cost: developers now have more e-mail to deal with, more mailing lists to follow, and special tools had to be written for the new environment. For example, SourceForge sends default patch and bug notification e-mail messages that are completely unhelpful, so Ka-Ping Yee wrote an HTML screen-scraper that sends more useful messages.

The ease of adding code caused a few initial growing pains, such as code was checked in before it was ready or without getting clear agreement from the developer group. The approval process that has emerged is somewhat similar to that used by the Apache group. Developers can vote +1, +0, -0, or -1 on a patch; +1 and -1 denote acceptance or rejection, while +0 and -0 mean the developer is mostly indifferent to the change, though with a slight positive or negative slant. The most significant change from the Apache model is that the voting is essentially advisory, letting Guido van Rossum, who has Benevolent Dictator For Life status, know what the general opinion is. He can still ignore the result of a vote, and approve or reject a change even if the community disagrees with him.

Producing an actual patch is the last step in adding a new feature, and is usually easy compared to the earlier task of coming up with a good design. Discussions of new features can often explode into lengthy mailing list threads, making the discussion hard to follow, and no one can read every posting to python-dev. Therefore, a relatively formal process has been set up to write Python Enhancement Proposals (PEPs), modelled on the internet RFC process. PEPs are draft documents that describe a proposed new feature, and are

continually revised until the community reaches a consensus, either accepting or rejecting the proposal. Quoting from the introduction to PEP 1, "PEP Purpose and Guidelines":

> PEP stands for Python Enhancement Proposal. A PEP is a design document providing information to the Python community, or describing a new feature for Python. The PEP should provide a concise technical specification of the feature and a rationale for the feature.
>
> We intend PEPs to be the primary mechanisms for proposing new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

Read the rest of PEP 1 for the details of the PEP editorial process, style, and format. PEPs are kept in the Python CVS tree on SourceForge, though they're not part of the Python 2.0 distribution, and are also available in HTML form from https://peps.python.org/. As of September 2000, there are 25 PEPS, ranging from PEP 201, "Lockstep Iteration", to PEP 225, "Elementwise/Objectwise Operators".

Unicode

The largest new feature in Python 2.0 is a new fundamental data type: Unicode strings. Unicode uses 16-bit numbers to represent characters instead of the 8-bit number used by ASCII, meaning that 65,536 distinct characters can be supported.

The final interface for Unicode support was arrived at through countless often-stormy discussions on the python-dev mailing list, and mostly implemented by Marc-AndrÃ© Lemburg, based on a Unicode string type implementation by Fredrik Lundh. A detailed explanation of the interface was written up as PEP 100, "Python Unicode Integration". This article will simply cover the most significant points about the Unicode interfaces.

In Python source code, Unicode strings are written as `u"string"`. Arbitrary Unicode characters can be written using a new escape sequence, `\uHHHH`, where *HHHH* is a 4-digit hexadecimal number from 0000 to FFFF. The existing `\xHHHH` escape sequence can also be used, and octal escapes can be used for characters up to U+01FF, which is represented by `\777`.

Unicode strings, just like regular strings, are an immutable sequence type. They can be indexed and sliced, but not modified in place. Unicode strings have an `encode( [encoding] )` method that returns an 8-bit string in the desired encoding. Encodings are named by strings, such as `'ascii'`, `'utf-8'`, `'iso-8859-1'`, or whatever. A codec API is defined for implementing and registering new encodings that are then available throughout a Python program. If an encoding isn't specified, the default encoding is usually 7-bit ASCII, though it can be changed for your Python installation by calling the `sys.setdefaultencoding(encoding)` function in a customized version of :file:`site.py`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 161); *backlink***
>
> Unknown interpreted text role "file".

Combining 8-bit and Unicode strings always coerces to Unicode, using the default ASCII encoding; the result of `'a' + u'bc'` is `u'abc'`.

New built-in functions have been added, and existing built-ins modified to support Unicode:

- `unichr(ch)` returns a Unicode string 1 character long, containing the character *ch*.
- `ord(u)`, where *u* is a 1-character regular or Unicode string, returns the number of the character as an integer.
- `unicode(string [, encoding] [, errors] )` creates a Unicode string from an 8-bit string. `encoding` is a string naming the encoding to use. The `errors` parameter specifies the treatment of characters that are invalid for the current encoding; passing `'strict'` as the value causes an exception to be raised on any encoding error, while `'ignore'` causes errors to be silently ignored and `'replace'` uses U+FFFD, the official replacement character, in case of any problems.
- The `exec` statement, and various built-ins such as `eval()`, `getattr()`, and `setattr()` will also accept Unicode strings as well as regular strings. (It's possible that the process of fixing this missed some built-ins; if you find a built-in function that accepts strings but doesn't accept Unicode strings at all, please report it as a bug.)

A new module, :mod:`unicodedata`, provides an interface to Unicode character properties. For example, `unicodedata.category(u'A')` returns the 2-character string 'Lu', the 'L' denoting it's a letter, and 'u' meaning that it's uppercase. `unicodedata.bidirectional(u'\u0660')` returns 'AN', meaning that U+0660 is an Arabic number.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 198); *backlink***
>
> Unknown interpreted text role "mod".

The :mod:`codecs` module contains functions to look up existing encodings and register new ones. Unless you want to implement a new encoding, you'll most often use the `codecs.lookup(encoding)` function, which returns a 4-element tuple: `(encode_func, decode_func, stream_reader, stream_writer)`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 204); *backlink*

Unknown interpreted text role "mod".

- *encode_func* is a function that takes a Unicode string, and returns a 2-tuple `(string, length)`. *string* is an 8-bit string containing a portion (perhaps all) of the Unicode string converted into the given encoding, and *length* tells you how much of the Unicode string was converted.
- *decode_func* is the opposite of *encode_func*, taking an 8-bit string and returning a 2-tuple `(ustring, length)`, consisting of the resulting Unicode string *ustring* and the integer *length* telling how much of the 8-bit string was consumed.
- *stream_reader* is a class that supports decoding input from a stream. *stream_reader(file_obj)* returns an object that supports the :meth:`read`, :meth:`readline`, and :meth:`readlines` methods. These methods will all translate from the given encoding and return Unicode strings.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 219); *backlink*

  Unknown interpreted text role "meth".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 219); *backlink*

  Unknown interpreted text role "meth".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 219); *backlink*

  Unknown interpreted text role "meth".

- *stream_writer*, similarly, is a class that supports encoding output to a stream. *stream_writer(file_obj)* returns an object that supports the :meth:`write` and :meth:`writelines` methods. These methods expect Unicode strings, translating them to the given encoding on output.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 224); *backlink*

  Unknown interpreted text role "meth".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 224); *backlink*

  Unknown interpreted text role "meth".

For example, the following code writes a Unicode string into a file, encoding it as UTF-8:

```
import codecs

unistr = u'\u0660\u2000ab ...'

(UTF8_encode, UTF8_decode,
 UTF8_streamreader, UTF8_streamwriter) = codecs.lookup('UTF-8')

output = UTF8_streamwriter( open( '/tmp/output', 'wb') )
output.write( unistr )
output.close()
```

The following code would then read UTF-8 input from the file:

```
input = UTF8_streamreader( open( '/tmp/output', 'rb') )
print repr(input.read())
input.close()
```

Unicode-aware regular expressions are available through the :mod:`re` module, which has a new underlying implementation called

SRE written by Fredrik Lundh of Secret Labs AB.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 249); *backlink*

Unknown interpreted text role "mod".

A `-U` command line option was added which causes the Python compiler to interpret all string literals as Unicode string literals. This is intended to be used in testing and future-proofing your Python code, since some future version of Python may drop support for 8-bit strings and provide only Unicode strings.

List Comprehensions

Lists are a workhorse data type in Python, and many programs manipulate a list at some point. Two common operations on lists are to loop over them, and either pick out the elements that meet a certain criterion, or apply some function to each element. For example, given a list of strings, you might want to pull out all the strings containing a given substring, or strip off trailing whitespace from each line.

The existing :func:`map` and :func:`filter` functions can be used for this purpose, but they require a function as one of their arguments. This is fine if there's an existing built-in function that can be passed directly, but if there isn't, you have to create a little function to do the required work, and Python's scoping rules make the result ugly if the little function needs additional information. Take the first example in the previous paragraph, finding all the strings in the list containing a given substring. You could write the following to do it:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 271); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 271); *backlink*

Unknown interpreted text role "func".

```
# Given the list L, make a list of all strings
# containing the substring S.
sublist = filter( lambda s, substring=S:
                    string.find(s, substring) != -1,
                  L)
```

Because of Python's scoping rules, a default argument is used so that the anonymous function created by the :keyword:`lambda` expression knows what substring is being searched for. List comprehensions make this cleaner:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 286); *backlink*

Unknown interpreted text role "keyword".

```
sublist = [ s for s in L if string.find(s, S) != -1 ]
```

List comprehensions have the form:

```
[ expression for expr in sequence1
             for expr2 in sequence2 ...
             for exprN in sequenceN
             if condition ]
```

The :keyword:`!for`...:keyword:`!in` clauses contain the sequences to be iterated over. The sequences do not have to be the same length, because they are *not* iterated over in parallel, but from left to right; this is explained more clearly in the following paragraphs. The elements of the generated list will be the successive values of *expression*. The final :keyword:`!if` clause is optional; if present, *expression* is only evaluated and added to the result if *condition* is true.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 299); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 299); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 299); *backlink*

Unknown interpreted text role "keyword".

To make the semantics very clear, a list comprehension is equivalent to the following Python code:

```
for expr1 in sequence1:
    for expr2 in sequence2:
    ...
        for exprN in sequenceN:
            if (condition):
                # Append the value of
                # the expression to the
                # resulting list.
```

This means that when there are multiple :keyword:`!for`...:keyword:`!in` clauses, the resulting list will be equal to the product of the lengths of all the sequences. If you have two lists of length 3, the output list is 9 elements long:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 319); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 319); *backlink*

Unknown interpreted text role "keyword".

```
seq1 = 'abc'
seq2 = (1,2,3)
>>> [ (x,y) for x in seq1 for y in seq2]
[('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b', 3), ('c', 1),
('c', 2), ('c', 3)]
```

To avoid introducing an ambiguity into Python's grammar, if *expression* is creating a tuple, it must be surrounded with parentheses. The first list comprehension below is a syntax error, while the second one is correct:

```
# Syntax error
[ x,y for x in seq1 for y in seq2]
# Correct
[ (x,y) for x in seq1 for y in seq2]
```

The idea of list comprehensions originally comes from the functional programming language Haskell (https://www.haskell.org). Greg Ewing argued most effectively for adding them to Python and wrote the initial list comprehension patch, which was then discussed for a seemingly endless time on the python-dev mailing list and kept up-to-date by Skip Montanaro.

Augmented Assignment

Augmented assignment operators, another long-requested feature, have been added to Python 2.0. Augmented assignment operators include `+=`, `-=`, `*=`, and so forth. For example, the statement `a += 2` increments the value of the variable `a` by 2, equivalent to the slightly lengthier `a = a + 2`.

The full list of supported assignment operators is `+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `&=`, `|=`, `^=`, `>>=`, and `<<=`. Python classes can override the augmented assignment operators by defining methods named :meth:`__iadd__`, :meth:`__isub__`, etc. For example, the following :class:`Number` class stores a number and supports using += to create a new instance with an incremented value.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 356); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 356); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 356); backlink

Unknown interpreted text role "class".

```
class Number:
    def __init__(self, value):
        self.value = value
    def __iadd__(self, increment):
        return Number( self.value + increment)

n = Number(5)
n += 3
print n.value
```

The :meth:`__iadd__` special method is called with the value of the increment, and should return a new instance with an appropriately modified value; this return value is bound as the new value of the variable on the left-hand side.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 377); backlink

Unknown interpreted text role "meth".

Augmented assignment operators were first introduced in the C programming language, and most C-derived languages, such as :program:`awk`, C++, Java, Perl, and PHP also support them. The augmented assignment patch was implemented by Thomas Wouters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 381); backlink

Unknown interpreted text role "program".

String Methods

Until now string-manipulation functionality was in the :mod:`string` module, which was usually a front-end for the :mod:`strop` module written in C. The addition of Unicode posed a difficulty for the :mod:`strop` module, because the functions would all need to be rewritten in order to accept either 8-bit or Unicode strings. For functions such as :func:`string.replace`, which takes 3 string arguments, that means eight possible permutations, and correspondingly complicated code.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 392); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 392); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 392); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 392); backlink

Unknown interpreted text role "func".

Instead, Python 2.0 pushes the problem onto the string type, making string manipulation functionality available through methods on both 8-bit strings and Unicode strings.

```
>>> 'andrew'.capitalize()
'Andrew'
>>> 'hostname'.replace('os', 'linux')
'hlinuxtname'
>>> 'moshe'.find('sh')
2
```

One thing that hasn't changed, a noteworthy April Fools' joke notwithstanding, is that Python strings are immutable. Thus, the string

methods return new strings, and do not modify the string on which they operate.

The old :mod:`string` module is still around for backwards compatibility, but it mostly acts as a front-end to the new string methods.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 415); *backlink***
>
> Unknown interpreted text role "mod".

Two methods which have no parallel in pre-2.0 versions, although they did exist in JPython for quite some time, are :meth:`startswith` and :meth:`endswith`. `s.startswith(t)` is equivalent to `s[:len(t)] == t`, while `s.endswith(t)` is equivalent to `s[-len(t):] == t`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 418); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 418); *backlink***
>
> Unknown interpreted text role "meth".

One other method which deserves special mention is :meth:`join`. The :meth:`join` method of a string receives one parameter, a sequence of strings, and is equivalent to the :func:`string.join` function from the old :mod:`string` module, with the arguments reversed. In other words, `s.join(seq)` is equivalent to the old `string.join(seq, s)`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 423); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 423); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 423); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 423); *backlink***
>
> Unknown interpreted text role "mod".

Garbage Collection of Cycles

The C implementation of Python uses reference counting to implement garbage collection. Every Python object maintains a count of the number of references pointing to itself, and adjusts the count as references are created or destroyed. Once the reference count reaches zero, the object is no longer accessible, since you need to have a reference to an object to access it, and if the count is zero, no references exist any longer.

Reference counting has some pleasant properties: it's easy to understand and implement, and the resulting implementation is portable, fairly fast, and reacts well with other libraries that implement their own memory handling schemes. The major problem with reference counting is that it sometimes doesn't realise that objects are no longer accessible, resulting in a memory leak. This happens when there are cycles of references.

Consider the simplest possible cycle, a class instance which has a reference to itself:

```
instance = SomeClass()
instance.myself = instance
```

After the above two lines of code have been executed, the reference count of `instance` is 2; one reference is from the variable named `'instance'`, and the other is from the `myself` attribute of the instance.

If the next line of code is `del instance`, what happens? The reference count of `instance` is decreased by 1, so it has a reference

count of 1; the reference in the `myself` attribute still exists. Yet the instance is no longer accessible through Python code, and it could be deleted. Several objects can participate in a cycle if they have references to each other, causing all of the objects to be leaked.

Python 2.0 fixes this problem by periodically executing a cycle detection algorithm which looks for inaccessible cycles and deletes the objects involved. A new :mod:`gc` module provides functions to perform a garbage collection, obtain debugging statistics, and tuning the collector's parameters.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 466); *backlink*
>
> Unknown interpreted text role "mod".

Running the cycle detection algorithm takes some time, and therefore will result in some additional overhead. It is hoped that after we've gotten experience with the cycle collection from using 2.0, Python 2.1 will be able to minimize the overhead with careful tuning. It's not yet obvious how much performance is lost, because benchmarking this is tricky and depends crucially on how often the program creates and destroys objects. The detection of cycles can be disabled when Python is compiled, if you can't afford even a tiny speed penalty or suspect that the cycle collection is buggy, by specifying the :option:`!--without-cycle-gc` switch when running the :program:`configure` script.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 471); *backlink*
>
> Unknown interpreted text role "option".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 471); *backlink*
>
> Unknown interpreted text role "program".

Several people tackled this problem and contributed to a solution. An early implementation of the cycle detection approach was written by Toby Kelsey. The current algorithm was suggested by Eric Tiedemann during a visit to CNRI, and Guido van Rossum and Neil Schemenauer wrote two different implementations, which were later integrated by Neil. Lots of other people offered suggestions along the way; the March 2000 archives of the python-dev mailing list contain most of the relevant discussion, especially in the threads titled "Reference cycle collection for Python" and "Finalization again".

Other Core Changes

Various minor changes have been made to Python's syntax and built-in functions. None of the changes are very far-reaching, but they're handy conveniences.

Minor Language Changes

A new syntax makes it more convenient to call a given function with a tuple of arguments and/or a dictionary of keyword arguments. In Python 1.5 and earlier, you'd use the :func:`apply` built-in function: `apply(f, args, kw)` calls the function :func:`f` with the argument tuple *args* and the keyword arguments in the dictionary *kw*. :func:`apply` is the same in 2.0, but thanks to a patch from Greg Ewing, `f(*args, **kw)` is a shorter and clearer way to achieve the same effect. This syntax is symmetrical with the syntax for defining functions:

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 504); *backlink*
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 504); *backlink*
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 504); *backlink*
>
> Unknown interpreted text role "func".

```
def f(*args, **kw):
    # args is a tuple of positional args,
    # kw is a dictionary of keyword args
    ...
```

The `print` statement can now have its output directed to a file-like object by following the `print` with `>> file`, similar to the redirection operator in Unix shells. Previously you'd either have to use the :meth:`write` method of the file-like object, which lacks the convenience and simplicity of `print`, or you could assign a new value to `sys.stdout` and then restore the old value. For sending output to standard error, it's much easier to write this:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 518); *backlink***
>
> Unknown interpreted text role "meth".

```
print >> sys.stderr, "Warning: action field not supplied"
```

Modules can now be renamed on importing them, using the syntax `import module as name` or `from module import name as othername`. The patch was submitted by Thomas Wouters.

A new format style is available when using the `%` operator; '%r' will insert the :func:`repr` of its argument. This was also added from symmetry considerations, this time for symmetry with the existing '%s' format style, which inserts the :func:`str` of its argument. For example, `'%r %s' % ('abc', 'abc')` returns a string containing `'abc' abc`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 532); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 532); *backlink***
>
> Unknown interpreted text role "func".

Previously there was no way to implement a class that overrode Python's built-in :keyword:`in` operator and implemented a custom version. `obj in seq` returns true if *obj* is present in the sequence *seq*; Python computes this by simply trying every index of the sequence until either *obj* is found or an :exc:`IndexError` is encountered. Moshe Zadka contributed a patch which adds a :meth:`__contains__` magic method for providing a custom implementation for :keyword:`!in`. Additionally, new built-in objects written in C can define what :keyword:`!in` means for them via a new slot in the sequence protocol.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 538); *backlink***
>
> Unknown interpreted text role "keyword".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 538); *backlink***
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 538); *backlink***
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 538); *backlink***
>
> Unknown interpreted text role "keyword".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 538); *backlink***
>
> Unknown interpreted text role "keyword".

Earlier versions of Python used a recursive algorithm for deleting objects. Deeply nested data structures could cause the interpreter to fill up the C stack and crash; Christian Tismer rewrote the deletion logic to fix this problem. On a related note, comparing recursive objects recursed infinitely and crashed; Jeremy Hylton rewrote the code to no longer crash, producing a useful result instead. For example, after this code:

```
a = []
b = []
```

```
a.append(a)
b.append(b)
```

The comparison a==b returns true, because the two recursive data structures are isomorphic. See the thread "trashcan and PR#7" in the April 2000 archives of the python-dev mailing list for the discussion leading up to this implementation, and some useful relevant links. Note that comparisons can now also raise exceptions. In earlier versions of Python, a comparison operation such as cmp(a,b) would always produce an answer, even if a user-defined :meth:`__cmp__` method encountered an error, since the resulting exception would simply be silently swallowed.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 559); *backlink*
>
> Unknown interpreted text role "meth".

Work has been done on porting Python to 64-bit Windows on the Itanium processor, mostly by Trent Mick of ActiveState. (Confusingly, sys.platform is still 'win32' on Win64 because it seems that for ease of porting, MS Visual C++ treats code as 32 bit on Itanium.) PythonWin also supports Windows CE; see the Python CE page at http://pythonce.sourceforge.net/ for more information.

Another new platform is Darwin/MacOS X; initial support for it is in Python 2.0. Dynamic loading works, if you specify "configure --with-dyld --with-suffix=.x". Consult the README in the Python source distribution for more instructions.

An attempt has been made to alleviate one of Python's warts, the often-confusing :exc:`NameError` exception when code refers to a local variable before the variable has been assigned a value. For example, the following code raises an exception on the print statement in both 1.5.2 and 2.0; in 1.5.2 a :exc:`NameError` exception is raised, while 2.0 raises a new :exc:`UnboundLocalError` exception. :exc:`UnboundLocalError` is a subclass of :exc:`NameError`, so any existing code that expects :exc:`NameError` to be raised should still work.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 581); *backlink*
>
> Unknown interpreted text role "exc".

```
def f():
    print "i=",i
    i = i + 1
f()
```

Two new exceptions, :exc:`TabError` and :exc:`IndentationError`, have been introduced. They're both subclasses of :exc:`SyntaxError`, and are raised when Python code is found to be improperly indented.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 595); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 595); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 595); *backlink*

Unknown interpreted text role "exc".

Changes to Built-in Functions

A new built-in, `zip(seq1, seq2, ...)`, has been added. :func:`zip` returns a list of tuples where each tuple contains the i-th element from each of the argument sequences. The difference between :func:`zip` and `map(None, seq1, seq2)` is that :func:`map` pads the sequences with `None` if the sequences aren't all of the same length, while :func:`zip` truncates the returned list to the length of the shortest argument sequence.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 603); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 603); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 603); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 603); *backlink*

Unknown interpreted text role "func".

The :func:`int` and :func:`long` functions now accept an optional "base" parameter when the first argument is a string. `int('123', 10)` returns 123, while `int('123', 16)` returns 291. `int(123, 16)` raises a :exc:`TypeError` exception with the message "can't convert non-string with explicit base".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 610); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 610); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 610); *backlink*

Unknown interpreted text role "exc".

A new variable holding more detailed version information has been added to the :mod:`sys` module. `sys.version_info` is a tuple `(major, minor, micro, level, serial)` For example, in a hypothetical 2.0.1beta1, `sys.version_info` would be `(2, 0, 1, 'beta', 1)`. *level* is a string such as `"alpha"`, `"beta"`, or `"final"` for a final release.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 616); *backlink*

Unknown interpreted text role "mod".

Dictionaries have an odd new method, `setdefault(key, default)`, which behaves similarly to the existing :meth:`get` method. However, if the key is missing, :meth:`setdefault` both returns the value of *default* as :meth:`get` would do, and also inserts it into the dictionary as the value for *key*. Thus, the following lines of code:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 622); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 622); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 622); *backlink*

Unknown interpreted text role "meth".

```
if dict.has_key( key ): return dict[key]
else:
    dict[key] = []
    return dict[key]
```

can be reduced to a single `return dict.setdefault(key, [])` statement.

The interpreter sets a maximum recursion depth in order to catch runaway recursion before filling the C stack and causing a core dump or GPF.. Previously this limit was fixed when you compiled Python, but in 2.0 the maximum recursion depth can be read and modified using :func:`sys.getrecursionlimit` and :func:`sys.setrecursionlimit`. The default value is 1000, and a rough maximum value for a given platform can be found by running a new script, :file:`Misc/find_recursionlimit.py`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 635); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 635); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 635); *backlink*

Unknown interpreted text role "file".

Porting to 2.0

New Python releases try hard to be compatible with previous releases, and the record has been pretty good. However, some changes are considered useful enough, usually because they fix initial design decisions that turned out to be actively mistaken, that breaking backward compatibility can't always be avoided. This section lists the changes in Python 2.0 that may cause old Python code to break.

The change which will probably break the most code is tightening up the arguments accepted by some methods. Some methods would take multiple arguments and treat them as a tuple, particularly various list methods such as :meth:`append` and :meth:`insert`. In earlier versions of Python, if `L` is a list, `L.append( 1,2 )` appends the tuple `(1,2)` to the list. In Python 2.0 this causes a :exc:`TypeError` exception to be raised, with the message: 'append requires exactly 1 argument; 2 given'. The fix is to simply add an extra set of parentheses to pass both values as a tuple: `L.append( (1,2) )`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 656); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 656); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 656); *backlink*

Unknown interpreted text role "exc".

The earlier versions of these methods were more forgiving because they used an old function in Python's C interface to parse their arguments; 2.0 modernizes them to use :func:`PyArg_ParseTuple`, the current argument parsing function, which provides more helpful error messages and treats multi-argument calls as errors. If you absolutely must use 2.0 but can't fix your code, you can edit :file:`Objects/listobject.c` and define the preprocessor symbol `NO_STRICT_LIST_APPEND` to preserve the old behaviour; this isn't recommended.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 665); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 665); *backlink*

Unknown interpreted text role "file".

Some of the functions in the :mod:`socket` module are still forgiving in this way. For example, :func:`socket.connect( ('hostname', 25) )` is the correct form, passing a tuple representing an IP address, but :func:`socket.connect( 'hostname', 25 )` also works. :func:`socket.connect_ex` and :func:`socket.bind` are similarly easy-going. 2.0alpha1 tightened these functions up, but because the documentation actually used the erroneous multiple argument form, many people wrote code which would break with the stricter checking. GvR backed out the changes in the face of public reaction, so for the :mod:`socket` module, the documentation was fixed and the multiple argument form is simply marked as deprecated; it *will* be tightened up again in a future Python version.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 673); *backlink*

Unknown interpreted text role "mod".

The `\x` escape in string literals now takes exactly 2 hex digits. Previously it would consume all the hex digits following the 'x' and take

the lowest 8 bits of the result, so `\x123456` was equivalent to `\x56`.

The :exc:`AttributeError` and :exc:`NameError` exceptions have a more friendly error message, whose text will be something like `'Spam' instance has no attribute 'eggs'` or `name 'eggs' is not defined`. Previously the error message was just the missing attribute name `eggs`, and code written to take advantage of this fact will break in 2.0.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 688); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 688); *backlink*

Unknown interpreted text role "exc".

Some work has been done to make integers and long integers a bit more interchangeable. In 1.5.2, large-file support was added for Solaris, to allow reading files larger than 2 GiB; this made the :meth:`tell` method of file objects return a long integer instead of a regular integer. Some code would subtract two file offsets and attempt to use the result to multiply a sequence or slice a string, but this raised a :exc:`TypeError`. In 2.0, long integers can be used to multiply or slice a sequence, and it'll behave as you'd intuitively expect it to; `3L * 'abc'` produces 'abcabcabc', and `(0,1,2,3)[2L:4L]` produces (2,3). Long integers can also be used in various contexts where previously only integers were accepted, such as in the :meth:`seek` method of file objects, and in the formats supported by the `%` operator (`%d`, `%i`, `%x`, etc.). For example, `"%d" % 2L**64` will produce the string `18446744073709551616`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 694); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 694); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 694); *backlink*

Unknown interpreted text role "meth".

The subtlest long integer change of all is that the :func:`str` of a long integer no longer has a trailing 'L' character, though :func:`repr` still includes it. The 'L' annoyed many people who wanted to print long integers that looked just like regular integers, since they had to go out of their way to chop off the character. This is no longer a problem in 2.0, but code which does `str(longval)[:-1]` and assumes the 'L' is there, will now lose the final digit.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 708); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 708); *backlink*

Unknown interpreted text role "func".

Taking the :func:`repr` of a float now uses a different formatting precision than :func:`str`. :func:`repr` uses `%.17g` format string for C's :func:`sprintf`, while :func:`str` uses `%.12g` as before. The effect is that :func:`repr` may occasionally show more decimal places than :func:`str`, for certain numbers. For example, the number 8.1 can't be represented exactly in binary, so `repr(8.1)` is `'8.0999999999999996'`, while str(8.1) is `'8.1'`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 716); *backlink*

Unknown interpreted text role "func".

The `-X` command-line option, which turned all standard exceptions into strings instead of classes, has been removed; the standard exceptions will now always be classes. The :mod:`exceptions` module containing the standard exceptions was translated from Python to a built-in C module, written by Barry Warsaw and Fredrik Lundh.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 724); *backlink*

Unknown interpreted text role "mod".

Extending/Embedding Changes

Some of the changes are under the covers, and will only be apparent to people writing C extension modules or embedding a Python interpreter in a larger application. If you aren't dealing with Python's C API, you can safely skip this section.

The version number of the Python C API was incremented, so C extensions compiled for 1.5.2 must be recompiled in order to work with 2.0. On Windows, it's not possible for Python 2.0 to import a third party extension built for Python 1.5.x due to how Windows DLLs work, so Python will raise an exception and the import will fail.

Users of Jim Fulton's ExtensionClass module will be pleased to find out that hooks have been added so that ExtensionClasses are now supported by :func:`isinstance` and :func:`issubclass`. This means you no longer have to remember to write code such as `if type(obj) == myExtensionClass`, but can use the more natural `if isinstance(obj, myExtensionClass)`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 753); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 753); *backlink*

Unknown interpreted text role "func".

The :file:`Python/importdl.c` file, which was a mass of #ifdefs to support dynamic loading on many different platforms, was cleaned up and reorganised by Greg Stein. :file:`importdl.c` is now quite small, and platform-specific code has been moved into a bunch of :file:`Python/dynload_\*.c` files. Another cleanup: there were also a number of :file:`my\*.h` files in the Include/ directory that held various portability hacks; they've been merged into a single file, :file:`Include/pyport.h`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 759); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 759); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 759); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 759); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 759); *backlink*

Unknown interpreted text role "file".

Vladimir Marangozov's long-awaited malloc restructuring was completed, to make it easy to have the Python interpreter use a custom allocator instead of C's standard :func:`malloc`. For documentation, read the comments in :file:`Include/pymem.h` and :file:`Include/objimpl.h`. For the lengthy discussions during which the interface was hammered out, see the web archives of the 'patches' and 'python-dev' lists at python.org.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 767); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 767); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 767); *backlink*

Unknown interpreted text role "file".

Recent versions of the GUSI development environment for MacOS support POSIX threads. Therefore, Python's POSIX threading support now works on the Macintosh. Threading support using the user-space GNU `pth` library was also contributed.

Threading support on Windows was enhanced, too. Windows supports thread locks that use kernel objects only in case of contention; in the common case when there's no contention, they use simpler functions which are an order of magnitude faster. A threaded version of Python 1.5.2 on NT is twice as slow as an unthreaded version; with the 2.0 changes, the difference is only 10%. These improvements were contributed by Yakov Markovitch.

Python 2.0's source now uses only ANSI C prototypes, so compiling Python now requires an ANSI C compiler, and can no longer be done using a compiler that only supports K&R C.

Previously the Python virtual machine used 16-bit numbers in its bytecode, limiting the size of source files. In particular, this affected the maximum size of literal lists and dictionaries in Python source; occasionally people who are generating Python code would run into this limit. A patch by Charles G. Waldman raises the limit from `2**16` to `2**32`.

Three new convenience functions intended for adding constants to a module's dictionary at module initialization time were added: :func:`PyModule_AddObject`, :func:`PyModule_AddIntConstant`, and :func:`PyModule_AddStringConstant`. Each of these functions takes a module object, a null-terminated C string containing the name to be added, and a third argument for the value to be assigned to the name. This third argument is, respectively, a Python object, a C long, or a C string.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 796); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 796); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 796); *backlink*

Unknown interpreted text role "func".

A wrapper API was added for Unix-style signal handlers. :func:`PyOS_getsig` gets a signal handler and :func:`PyOS_setsig` will set a new handler.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 804); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 804); *backlink*

Unknown interpreted text role "func".

Distutils: Making Modules Easy to Install

Before Python 2.0, installing modules was a tedious affair -- there was no way to figure out automatically where Python is installed, or what compiler options to use for extension modules. Software authors had to go through an arduous ritual of editing Makefiles and configuration files, which only really work on Unix and leave Windows and MacOS unsupported. Python users faced wildly differing installation instructions which varied between different extension packages, which made administering a Python installation something of a chore.

The SIG for distribution utilities, shepherded by Greg Ward, has created the Distutils, a system to make package installation much easier. They form the :mod:`distutils` package, a new part of Python's standard library. In the best case, installing a Python module from source will require the same steps: first you simply mean unpack the tarball or zip archive, and the run "`python setup.py install`". The platform will be automatically detected, the compiler will be recognized, C extension modules will be compiled, and the distribution installed into the proper directory. Optional command-line arguments provide more control over the installation process, the distutils package offers many places to override defaults -- separating the build from the install, building or installing in non-default directories, and more.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 821); *backlink*

Unknown interpreted text role "mod".

In order to use the Distutils, you need to write a :file:`setup.py` script. For the simple case, when the software contains only .py files, a minimal :file:`setup.py` can be just a few lines long:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 833); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 833); *backlink*

Unknown interpreted text role "file".

```
from distutils.core import setup
setup (name = "foo", version = "1.0",
       py_modules = ["module1", "module2"])
```

The :file:`setup.py` file isn't much more complicated if the software consists of a few packages:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 841); backlink

Unknown interpreted text role "file".

```
from distutils.core import setup
setup (name = "foo", version = "1.0",
       packages = ["package", "package.subpackage"])
```

A C extension can be the most complicated case; here's an example taken from the PyXML package:

```
from distutils.core import setup, Extension

expat_extension = Extension('xml.parsers.pyexpat',
     define_macros = [('XML_NS', None)],
     include_dirs = [ 'extensions/expat/xmltok',
                      'extensions/expat/xmlparse' ],
     sources = [ 'extensions/pyexpat.c',
                 'extensions/expat/xmltok/xmltok.c',
                 'extensions/expat/xmltok/xmlrole.c', ]
       )
setup (name = "PyXML", version = "0.5.4",
       ext_modules =[ expat_extension ] )
```

The Distutils can also take care of creating source and binary distributions. The "sdist" command, run by "`python setup.py sdist`", builds a source distribution such as :file:`foo-1.0.tar.gz`. Adding new commands isn't difficult, "bdist_rpm" and "bdist_wininst" commands have already been contributed to create an RPM distribution and a Windows installer for the software, respectively. Commands to create other distribution formats such as Debian packages and Solaris :file:`.pkg` files are in various stages of development.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 864); backlink

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 864); backlink

Unknown interpreted text role "file".

All this is documented in a new manual, *Distributing Python Modules*, that joins the basic set of Python documentation.

XML Modules

Python 1.5.2 included a simple XML parser in the form of the :mod:`xmllib` module, contributed by Sjoerd Mullender. Since 1.5.2's release, two different interfaces for processing XML have become common: SAX2 (version 2 of the Simple API for XML) provides an event-driven interface with some similarities to :mod:`xmllib`, and the DOM (Document Object Model) provides a tree-based interface, transforming an XML document into a tree of nodes that can be traversed and modified. Python 2.0 includes a SAX2 interface and a stripped-down DOM interface as part of the :mod:`xml` package. Here we will give a brief overview of these new interfaces; consult the Python documentation or the source code for complete details. The Python XML SIG is also working on improved documentation.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 882); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 882); backlink

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 882); backlink

Unknown interpreted text role "mod".

SAX2 Support

SAX defines an event-driven interface for parsing XML. To use SAX, you must write a SAX handler class. Handler classes inherit

from various classes provided by SAX, and override various methods that will then be called by the XML parser. For example, the :meth:`startElement` and :meth:`endElement` methods are called for every starting and end tag encountered by the parser, the :meth:`characters` method is called for every chunk of character data, and so forth.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 898); *backlink*

Unknown interpreted text role "meth".

The advantage of the event-driven approach is that the whole document doesn't have to be resident in memory at any one time, which matters if you are processing really huge documents. However, writing the SAX handler class can get very complicated if you're trying to modify the document structure in some elaborate way.

For example, this little example program defines a handler that prints a message for every starting and ending tag, and then parses the file :file:`hamlet.xml` using it:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 912); *backlink*

Unknown interpreted text role "file".

```
from xml import sax

class SimpleHandler(sax.ContentHandler):
    def startElement(self, name, attrs):
        print 'Start of element:', name, attrs.keys()

    def endElement(self, name):
        print 'End of element:', name

# Create a parser object
parser = sax.make_parser()

# Tell it what handler to use
handler = SimpleHandler()
parser.setContentHandler( handler )

# Parse a file!
parser.parse( 'hamlet.xml' )
```

For more information, consult the Python documentation, or the XML HOWTO at http://pyxml.sourceforge.net/topics/howto/xml-howto.html.

DOM Support

The Document Object Model is a tree-based representation for an XML document. A top-level :class:`Document` instance is the root of the tree, and has a single child which is the top-level :class:`Element` instance. This :class:`Element` has children nodes representing character data and any sub-elements, which may have further children of their own, and so forth. Using the DOM you can traverse the resulting tree any way you like, access element and attribute values, insert and delete nodes, and convert the tree back into XML.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 942); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 942); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 942); *backlink*

Unknown interpreted text role "class".

The DOM is useful for modifying XML documents, because you can create a DOM tree, modify it by adding new nodes or rearranging subtrees, and then produce a new XML document as output. You can also construct a DOM tree manually and convert it to XML, which can be a more flexible way of producing XML output than simply writing `<tag1>`...`</tag1>` to a file.

The DOM implementation included with Python lives in the :mod:`xml.dom.minidom` module. It's a lightweight implementation of the Level 1 DOM with support for XML namespaces. The :func:`parse` and :func:`parseString` convenience functions are provided for generating a DOM tree:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 956); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 956); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 956); *backlink*

Unknown interpreted text role "func".

```
from xml.dom import minidom
doc = minidom.parse('hamlet.xml')
```

`doc` is a :class:`Document` instance. :class:`Document`, like all the other DOM classes such as :class:`Element` and :class:`Text`, is a subclass of the :class:`Node` base class. All the nodes in a DOM tree therefore support certain common methods, such as :meth:`toxml` which returns a string containing the XML representation of the node and its children. Each class also has special methods of its own; for example, :class:`Element` and :class:`Document` instances have a method to find all child elements with a given tag name. Continuing from the previous 2-line example:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 964); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 964); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 964); *backlink*

Unknown interpreted text role "class".

```
perslist = doc.getElementsByTagName( 'PERSONA' )
print perslist[0].toxml()
print perslist[1].toxml()
```

For the *Hamlet* XML file, the above few lines output:

```
<PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
```

The root element of the document is available as `doc.documentElement`, and its children can be easily modified by deleting, adding, or removing nodes:

```
root = doc.documentElement

# Remove the first child
root.removeChild( root.childNodes[0] )

# Move the new first child to the end
root.appendChild( root.childNodes[0] )

# Insert the new first child (originally,
# the third child) before the 20th child.
root.insertBefore( root.childNodes[0], root.childNodes[20] )
```

Again, I will refer you to the Python documentation for a complete listing of the different :class:`Node` classes and their various methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 997); *backlink*

Unknown interpreted text role "class".

Relationship to PyXML

The XML Special Interest Group has been working on XML-related Python code for a while. Its code distribution, called PyXML, is available from the SIG's web pages at https://www.python.org/community/sigs/current/xml-sig. The PyXML distribution also used the package name `xml`. If you've written programs that used PyXML, you're probably wondering about its compatibility with the 2.0 :mod:`xml` package.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1004); *backlink*

Unknown interpreted text role "mod".

The answer is that Python 2.0's :mod:`xml` package isn't compatible with PyXML, but can be made compatible by installing a recent version PyXML. Many applications can get by with the XML support that is included with Python 2.0, but more complicated applications will require that the full PyXML package will be installed. When installed, PyXML versions 0.6.0 or greater will replace the :mod:`xml` package shipped with Python, and will be a strict superset of the standard package, adding a bunch of additional features. Some of the additional features in PyXML include:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1010); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1010); *backlink*

Unknown interpreted text role "mod".

- 4DOM, a full DOM implementation from FourThought, Inc.
- The xmlproc validating parser, written by Lars Marius Garshol.
- The :mod:`sgmlop` parser accelerator module, written by Fredrik Lundh.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1023); *backlink*
>
> Unknown interpreted text role "mod".

Module changes

Lots of improvements and bugfixes were made to Python's extensive standard library; some of the affected modules include :mod:`readline`, :mod:`ConfigParser`, :mod:`cgi`, :mod:`calendar`, :mod:`posix`, :mod:`readline`, :mod:`xmllib`, :mod:`aifc`, :mod:`chunk, wave`, :mod:`random`, :mod:`shelve`, and :mod:`nntplib`. Consult the CVS logs for the exact patch-by-patch details.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1031); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1031); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1031); *backlink*

Unknown interpreted text role "mod".

Brian Gallew contributed OpenSSL support for the :mod:`socket` module. OpenSSL is an implementation of the Secure Socket Layer, which encrypts the data being sent over a socket. When compiling Python, you can edit :file:`Modules/Setup` to include SSL support, which adds an additional function to the :mod:`socket` module: `socket.ssl(socket, keyfile, certfile)`, which takes a socket object and returns an SSL socket. The :mod:`httplib` and :mod:`urllib` modules were also changed to support `https://` URLs, though no one has implemented FTP or SMTP over SSL.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1037); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1037); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1037); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1037); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1037); *backlink*

Unknown interpreted text role "mod".

The :mod:`httplib` module has been rewritten by Greg Stein to support HTTP/1.1. Backward compatibility with the 1.5 version of :mod:`httplib` is provided, though using HTTP/1.1 features such as pipelining will require rewriting code to use a different set of interfaces.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1046); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1046); *backlink*

Unknown interpreted text role "mod".

The :mod:`Tkinter` module now supports Tcl/Tk version 8.1, 8.2, or 8.3, and support for the older 7.x versions has been dropped. The Tkinter module now supports displaying Unicode strings in Tk widgets. Also, Fredrik Lundh contributed an optimization which makes operations like `create_line` and `create_polygon` much faster, especially when using lots of coordinates.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1051); *backlink*

Unknown interpreted text role "mod".

The :mod:`curses` module has been greatly extended, starting from Oliver Andrich's enhanced version, to provide many additional functions from ncurses and SYSV curses, such as colour, alternative character set support, pads, and mouse support. This means the module is no longer compatible with operating systems that only have BSD curses, but there don't seem to be any currently maintained OSes that fall into this category.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1057); *backlink*

Unknown interpreted text role "mod".

As mentioned in the earlier discussion of 2.0's Unicode support, the underlying implementation of the regular expressions provided by the :mod:`re` module has been changed. SRE, a new regular expression engine written by Fredrik Lundh and partially funded by Hewlett Packard, supports matching against both 8-bit strings and Unicode strings.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1064); *backlink*

Unknown interpreted text role "mod".

New modules

A number of new modules were added. We'll simply list them with brief descriptions; consult the 2.0 documentation for the details of a particular module.

- :mod:`atexit`: For registering functions to be called before the Python interpreter exits. Code that currently sets `sys.exitfunc` directly should be changed to use the :mod:`atexit` module instead, importing :mod:`atexit` and calling :func:`atexit.register` with the function to be called on exit. (Contributed by Skip Montanaro.)

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1080); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1080); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1080); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1080); *backlink*

  Unknown interpreted text role "func".

- :mod:`codecs`, :mod:`encodings`, :mod:`unicodedata`: Added as part of the new Unicode support.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1086); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1086); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1086**); ***backlink***

Unknown interpreted text role "mod".

- :mod:`filecmp`: Supersedes the old :mod:`cmp`, :mod:`cmpcache` and :mod:`dircmp` modules, which have now become deprecated. (Contributed by Gordon MacMillan and Moshe Zadka.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1089**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1089**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1089**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1089**); ***backlink***

Unknown interpreted text role "mod".

- :mod:`gettext`: This module provides internationalization (I18N) and localization (L10N) support for Python programs by providing an interface to the GNU gettext message catalog library. (Integrated by Barry Warsaw, from separate contributions by Martin von LÃ¶wis, Peter Funk, and James Henstridge.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1093**); ***backlink***

Unknown interpreted text role "mod".

- :mod:`linuxaudiodev`: Support for the :file:`/dev/audio` device on Linux, a twin to the existing :mod:`sunaudiodev` module. (Contributed by Peter Bosch, with fixes by Jeremy Hylton.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1098**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1098**); ***backlink***

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1098**); ***backlink***

Unknown interpreted text role "mod".

- :mod:`mmap`: An interface to memory-mapped files on both Windows and Unix. A file's contents can be mapped directly into memory, at which point it behaves like a mutable string, so its contents can be read and modified. They can even be passed to functions that expect ordinary strings, such as the :mod:`re` module. (Contributed by Sam Rushing, with some extensions by A.M. Kuchling.)

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1102); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1102); *backlink*

  Unknown interpreted text role "mod".

- :mod:`pyexpat`: An interface to the Expat XML parser. (Contributed by Paul Prescod.)

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1108); *backlink*

  Unknown interpreted text role "mod".

- :mod:`robotparser`: Parse a :file:`robots.txt` file, which is used for writing web spiders that politely avoid certain areas of a web site. The parser accepts the contents of a :file:`robots.txt` file, builds a set of rules from it, and can then answer questions about the fetchability of a given URL. (Contributed by Skip Montanaro.)

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1111); *backlink*

  Unknown interpreted text role "mod".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1111); *backlink*

  Unknown interpreted text role "file".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1111); *backlink*

  Unknown interpreted text role "file".

- :mod:`tabnanny`: A module/script to check Python source code for ambiguous indentation. (Contributed by Tim Peters.)

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1117); *backlink*

  Unknown interpreted text role "mod".

- :mod:`UserString`: A base class useful for deriving objects that behave like strings.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst, line 1120); *backlink*

  Unknown interpreted text role "mod".

- :mod:`webbrowser`: A module that provides a platform independent way to launch a web browser on a specific URL. For

each platform, various browsers are tried in a specific order. The user can alter which browser is launched by setting the *BROWSER* environment variable. (Originally inspired by Eric S. Raymond's patch to :mod:`urllib` which added similar functionality, but the final module comes from code originally implemented by Fred Drake as :file:`Tools/idle/BrowserControl.py`, and adapted for the standard library by Fred.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1123**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1123**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1123**); ***backlink***

Unknown interpreted text role "file".

- :mod:`_winreg`: An interface to the Windows registry. :mod:`_winreg` is an adaptation of functions that have been part of PythonWin since 1995, but has now been added to the core distribution, and enhanced to support Unicode. :mod:`_winreg` was written by Bill Tutt and Mark Hammond.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1132**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1132**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1132**); ***backlink***

Unknown interpreted text role "mod".

- :mod:`zipfile`: A module for reading and writing ZIP-format archives. These are archives produced by :program:`PKZIP` on DOS/Windows or :program:`zip` on Unix, not to be confused with :program:`gzip`-format files (which are supported by the :mod:`gzip` module) (Contributed by James C. Ahlstrom.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1137**); ***backlink***

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1137**); ***backlink***

Unknown interpreted text role "program".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, **line 1137**); ***backlink***

Unknown interpreted text role "program".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1137); *backlink*

Unknown interpreted text role "program".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1137); *backlink*

Unknown interpreted text role "mod".

- :mod:`imputil`: A module that provides a simpler way for writing customized import hooks, in comparison to the existing :mod:`ihooks` module. (Implemented by Greg Stein, with much discussion on python-dev along the way.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1142); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1142); *backlink*

Unknown interpreted text role "mod".

IDLE Improvements

IDLE is the official Python cross-platform IDE, written using Tkinter. Python 2.0 includes IDLE 0.6, which adds a number of new features and improvements. A partial list:

- UI improvements and optimizations, especially in the area of syntax highlighting and auto-indentation.
- The class browser now shows more information, such as the top level functions in a module.
- Tab width is now a user settable option. When opening an existing Python file, IDLE automatically detects the indentation conventions, and adapts.
- There is now support for calling browsers on various platforms, used to open the Python documentation in a browser.
- IDLE now has a command line, which is largely similar to the vanilla Python interpreter.
- Call tips were added in many places.
- IDLE can now be installed as a package.
- In the editor window, there is now a line/column bar at the bottom.
- Three new keystroke commands: Check module (:kbd:`Alt-F5`), Import module (:kbd:`F5`) and Run script (:kbd:`Ctrl-F5`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1177); *backlink*

Unknown interpreted text role "kbd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1177); *backlink*

Unknown interpreted text role "kbd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst, line 1177); *backlink*

Unknown interpreted text role "kbd".

Deleted and Deprecated Modules

A few modules have been dropped because they're obsolete, or because there are now better ways to do the same thing. The :mod:`stdwin` module is gone; it was for a platform-independent windowing toolkit that's no longer developed.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1186); *backlink*
>
> Unknown interpreted text role "mod".

A number of modules have been moved to the :file:`lib-old` subdirectory: :mod:`cmp`, :mod:`cmpcache`, :mod:`dircmp`, :mod:`dump`, :mod:`find`, :mod:`grep`, :mod:`packmail`, :mod:`poly`, :mod:`util`, :mod:`whatsound`, :mod:`zmod`. If you have code which relies on a module that's been moved to :file:`lib-old`, you can simply add that directory to `sys.path` to get them back, but you're encouraged to update any code that uses these modules.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "file".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*
>
> Unknown interpreted text role "mod".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main) (Doc) (whatsnew)2.0.rst`, line 1190); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst`, line 1190); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst`, line 1190); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\whatsnew\(cpython-main)(Doc)(whatsnew)2.0.rst`, line 1190); *backlink*

Unknown interpreted text role "file".

Acknowledgements

The authors would like to thank the following people for offering suggestions on various drafts of this article: David Bolen, Mark Hammond, Gregg Hauser, Jeremy Hylton, Fredrik Lundh, Detlef Lannert, Aahz Maruch, Skip Montanaro, Vladimir Marangozov, Tobias Polzin, Guido van Rossum, Neil Schemenauer, and Russ Schmidt.