What's New in Python

Release 3.1.2

A. M. Kuchling

September 28, 2010

Python Software Foundation
Email: docs@python.org

Contents

Author Raymond Hettinger

Release 3.1.2

Date September 28, 2010

This article explains the new features in Python 3.1, compared to 3.0.

1 PEP 372: Ordered Dictionaries

Regular Python dictionaries iterate over key/value pairs in arbitrary order. Over the years, a number of authors have written alternative implementations that remember the order that the keys were originally inserted. Based on the experiences from those implementations, a new `collections.OrderedDict` class has been introduced.

The OrderedDict API is substantially the same as regular dictionaries but will iterate over keys and values in a guaranteed order depending on when a key was first inserted. If a new entry overwrites an existing entry, the original insertion position is left unchanged. Deleting an entry and reinserting it will move it to the end.

The standard library now supports use of ordered dictionaries in several modules. The `configparser` module uses them by default. This lets configuration files be read, modified, and then written back in their original order. The *_asdict()* method for `collections.namedtuple()` now returns an ordered dictionary with the values appearing in the same order as the underlying tuple indicies. The `json` module is being built-out with an *object_pairs_hook* to allow OrderedDicts to be built by the decoder. Support was also added for third-party tools like PyYAML.

See Also:

PEP 372 - Ordered Dictionaries PEP written by Armin Ronacher and Raymond Hettinger. Implementation written by Raymond Hettinger.

Since an ordered dictionary remembers its insertion order, it can be used in conjuction with sorting to make a sorted dictionary:

```
>>> # regular unsorted dictionary
>>> d = {'banana': 3, 'apple':4, 'pear': 1, 'orange': 2}

>>> # dictionary sorted by key
>>> OrderedDict(sorted(d.items(), key=lambda t: t[0]))
OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])

>>> # dictionary sorted by value
>>> OrderedDict(sorted(d.items(), key=lambda t: t[1]))
OrderedDict([('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)])

>>> # dictionary sorted by length of the key string
>>> OrderedDict(sorted(d.items(), key=lambda t: len(t[0])))
OrderedDict([('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)])
```

The new sorted dictionaries maintain their sort order when entries are deleted. But when new keys are added, the keys are appended to the end and the sort is not maintained.

2 PEP 378: Format Specifier for Thousands Separator

The builtin `format()` function and the `str.format()` method use a mini-language that now includes a simple, non-locale aware way to format a number with a thousands separator. That provides a way to humanize a program's output, improving its professional appearance and readability:

```
>>> format(1234567, ',d')
'1,234,567'
>>> format(1234567.89, ',.2f')
'1,234,567.89'
>>> format(12345.6 + 8901234.12j, ',f')
'12,345.600000+8,901,234.120000j'
>>> format(Decimal('1234567.89'), ',f')
'1,234,567.89'
```

The supported types are `int`, `float`, `complex` and `decimal.Decimal`.

Discussions are underway about how to specify alternative separators like dots, spaces, apostrophes, or underscores. Locale-aware applications should use the existing *n* format specifier which already has some support for thousands separators.

See Also:

PEP 378 - Format Specifier for Thousands Separator PEP written by Raymond Hettinger and implemented by Eric Smith and Mark Dickinson.

3 Other Language Changes

Some smaller changes made to the core Python language are:

- Directories and zip archives containing a `__main__.py` file can now be executed directly by passing their name to the interpreter. The directory/zipfile is automatically inserted as the first entry in sys.path. (Suggestion and initial patch by Andy Chu; revised patch by Phillip J. Eby and Nick Coghlan; issue 1739468.)

- The `int()` type gained a `bit_length` method that returns the number of bits necessary to represent its argument in binary:

```
>>> n = 37
>>> bin(37)
'0b100101'
>>> n.bit_length()
6
>>> n = 2**123-1
>>> n.bit_length()
123
>>> (n+1).bit_length()
124
```

  (Contributed by Fredrik Johansson, Victor Stinner, Raymond Hettinger, and Mark Dickinson; issue 3439.)

- The fields in `format()` strings can now be automatically numbered:

```
>>> 'Sir {} of {}'.format('Gallahad', 'Camelot')
'Sir Gallahad of Camelot'
```

  Formerly, the string would have required numbered fields such as: `'Sir {0} of {1}'`.

  (Contributed by Eric Smith; issue 5237.)

- The `string.maketrans()` function is deprecated and is replaced by new static methods, `bytes.maketrans()` and `bytearray.maketrans()`. This change solves the confusion around which types were supported by the `string` module. Now, `str`, `bytes`, and `bytearray` each have their own **maketrans** and **translate** methods with intermediate translation tables of the appropriate type.

  (Contributed by Georg Brandl; issue 5675.)

- The syntax of the `with` statement now allows multiple context managers in a single statement:

```
>>> with open('mylog.txt') as infile, open('a.out', 'w') as outfile:
...     for line in infile:
...         if '<critical>' in line:
...             outfile.write(line)
```

  With the new syntax, the `contextlib.nested()` function is no longer needed and is now deprecated.

  (Contributed by Georg Brandl and Mattias Brändström; appspot issue 53094.)

- `round(x, n)` now returns an integer if *x* is an integer. Previously it returned a float:

```
>>> round(1123, -2)
1100
```

  (Contributed by Mark Dickinson; issue 4707.)

- Python now uses David Gay's algorithm for finding the shortest floating point representation that doesn't change its value. This should help mitigate some of the confusion surrounding binary floating point numbers.

  The significance is easily seen with a number like `1.1` which does not have an exact equivalent in binary floating point. Since there is no exact equivalent, an expression like `float('1.1')`

evaluates to the nearest representable value which is `0x1.199999999999ap+0` in hex or `1.1000000000000008881784197001252323890533447265625` in decimal. That nearest value was and still is used in subsequent floating point calculations.

What is new is how the number gets displayed. Formerly, Python used a simple approach. The value of `repr(1.1)` was computed as `format(1.1, '.17g')` which evaluated to `'1.1000000000000001'`. The advantage of using 17 digits was that it relied on IEEE-754 guarantees to assure that `eval(repr(1.1))` would round-trip exactly to its original value. The disadvantage is that many people found the output to be confusing (mistaking intrinsic limitations of binary floating point representation as being a problem with Python itself).

The new algorithm for `repr(1.1)` is smarter and returns `'1.1'`. Effectively, it searches all equivalent string representations (ones that get stored with the same underlying float value) and returns the shortest representation.

The new algorithm tends to emit cleaner representations when possible, but it does not change the underlying values. So, it is still the case that `1.1 + 2.2 != 3.3` even though the representations may suggest otherwise.

The new algorithm depends on certain features in the underlying floating point implementation. If the required features are not found, the old algorithm will continue to be used. Also, the text pickle protocols assure cross-platform portability by using the old algorithm.

(Contributed by Eric Smith and Mark Dickinson; issue 1580)

4 New, Improved, and Deprecated Modules

- Added a `collections.Counter` class to support convenient counting of unique items in a sequence or iterable:

```
>>> Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])
Counter({'blue': 3, 'red': 2, 'green': 1})
```

(Contributed by Raymond Hettinger; issue 1696199.)

- Added a new module, `tkinter.ttk` for access to the Tk themed widget set. The basic idea of ttk is to separate, to the extent possible, the code implementing a widget's behavior from the code implementing its appearance.

(Contributed by Guilherme Polo; issue 2983.)

- The `gzip.GzipFile` and `bz2.BZ2File` classes now support the context manager protocol:

```
>>> # Automatically close file after writing
>>> with gzip.GzipFile(filename, "wb") as f:
...     f.write(b"xxx")
```

(Contributed by Antoine Pitrou.)

- The `decimal` module now supports methods for creating a decimal object from a binary `float`. The conversion is exact but can sometimes be surprising:

```
>>> Decimal.from_float(1.1)
Decimal('1.1000000000000008881784197001252323890533447265625')
```

The long decimal result shows the actual binary fraction being stored for *1.1*. The fraction has many digits because *1.1* cannot be exactly represented in binary.

(Contributed by Raymond Hettinger and Mark Dickinson.)

- The `itertools` module grew two new functions. The `itertools.combinations_with_replacement()` function is one of four for generating combinatorics including permutations and Cartesian products. The `itertools.compress()` function mimics its namesake from APL. Also, the existing `itertools.count()` function now has an optional *step* argument and can accept any type of counting sequence including `fractions.Fraction` and `decimal.Decimal`:

```
>>> [p+q for p,q in combinations_with_replacement('LOVE', 2)]
['LL', 'LO', 'LV', 'LE', 'OO', 'OV', 'OE', 'VV', 'VE', 'EE']

>>> list(compress(data=range(10), selectors=[0,0,1,1,0,1,0,1,0,0]))
[2, 3, 5, 7]

>>> c = count(start=Fraction(1,2), step=Fraction(1,6))
>>> [next(c), next(c), next(c), next(c)]
[Fraction(1, 2), Fraction(2, 3), Fraction(5, 6), Fraction(1, 1)]
```

  (Contributed by Raymond Hettinger.)

- `collections.namedtuple()` now supports a keyword argument *rename* which lets invalid fieldnames be automatically converted to positional names in the form _0, _1, etc. This is useful when the field names are being created by an external source such as a CSV header, SQL field list, or user input:

```
>>> query = input()
SELECT region, dept, count(*) FROM main GROUPBY region, dept

>>> cursor.execute(query)
>>> query_fields = [desc[0] for desc in cursor.description]
>>> UserQuery = namedtuple('UserQuery', query_fields, rename=True)
>>> pprint.pprint([UserQuery(*row) for row in cursor])
[UserQuery(region='South', dept='Shipping', _2=185),
 UserQuery(region='North', dept='Accounting', _2=37),
 UserQuery(region='West', dept='Sales', _2=419)]
```

  (Contributed by Raymond Hettinger; issue 1818.)

- The `re.sub()`, `re.subn()` and `re.split()` functions now accept a flags parameter.

  (Contributed by Gregory Smith.)

- The `logging` module now implements a simple `logging.NullHandler` class for applications that are not using logging but are calling library code that does. Setting-up a null handler will suppress spurious warnings such as "No handlers could be found for logger foo":

```
>>> h = logging.NullHandler()
>>> logging.getLogger("foo").addHandler(h)
```

  (Contributed by Vinay Sajip; issue 4384).

- The `runpy` module which supports the `-m` command line switch now supports the execution of packages by looking for and executing a `__main__` submodule when a package name is supplied.

  (Contributed by Andi Vajda; issue 4195.)

- The `pdb` module can now access and display source code loaded via `zipimport` (or any other conformant **PEP 302** loader).

  (Contributed by Alexander Belopolsky; issue 4201.)

- `functools.partial` objects can now be pickled.

  (Suggested by Antoine Pitrou and Jesse Noller. Implemented by Jack Diedrich; issue 5228.)

- Add `pydoc` help topics for symbols so that `help('@')` works as expected in the interactive environment.

  (Contributed by David Laban; issue 4739.)

- The `unittest` module now supports skipping individual tests or classes of tests. And it supports marking a test as a expected failure, a test that is known to be broken, but shouldn't be counted as a failure on a TestResult:

```
class TestGizmo(unittest.TestCase):

    @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
    def test_gizmo_on_windows(self):
        ...

    @unittest.expectedFailure
    def test_gimzo_without_required_library(self):
        ...
```

  Also, tests for exceptions have been builtout to work with context managers using the `with` statement:

```
def test_division_by_zero(self):
    with self.assertRaises(ZeroDivisionError):
        x / 0
```

  In addition, several new assertion methods were added including `assertSetEqual()`, `assertDictEqual()`, `assertDictContainsSubset()`, `assertListEqual()`, `assertTupleEqual()`, `assertSequenceEqual()`, `assertRaisesRegexp()`, `assertIsNone()`, and `assertIsNotNone()`.

  (Contributed by Benjamin Peterson and Antoine Pitrou.)

- The `io` module has three new constants for the `seek()` method `SEEK_SET`, `SEEK_CUR`, and `SEEK_END`.

- The `sys.version_info` tuple is now a named tuple:

```
>>> sys.version_info
sys.version_info(major=3, minor=1, micro=0, releaselevel='alpha', serial=2)
```

  (Contributed by Ross Light; issue 4285.)

- The `nntplib` and `imaplib` modules now support IPv6.

  (Contributed by Derek Morr; issue 1655 and issue 1664.)

- The `pickle` module has been adapted for better interoperability with Python 2.x when used with protocol 2 or lower. The reorganization of the standard library changed the formal reference for many objects. For example, `__builtin__.set` in Python 2 is called `builtins.set` in Python 3. This change confounded efforts to share data between different versions of Python. But now when protocol 2 or lower is selected, the pickler will automatically use the old Python 2 names for both loading and dumping. This remapping is turned-on by default but can be disabled with the *fix_imports* option:

```
>>> s = {1, 2, 3}
>>> pickle.dumps(s, protocol=0)
b'c__builtin__\nset\np0\n((lp1\nL1L\naL2L\naL3L\natp2\nRp3\n.'
>>> pickle.dumps(s, protocol=0, fix_imports=False)
b'cbuiltins\nset\np0\n((lp1\nL1L\naL2L\naL3L\natp2\nRp3\n.'
```

  An unfortunate but unavoidable side-effect of this change is that protocol 2 pickles produced by Python 3.1 won't be readable with Python 3.0. The latest pickle protocol, protocol 3, should be used when migrating data between Python 3.x implementations, as it doesn't attempt to remain compatible with Python 2.x.

  (Contributed by Alexandre Vassalotti and Antoine Pitrou, issue 6137.)

- A new module, `importlib` was added. It provides a complete, portable, pure Python reference implementation of the `import` statement and its counterpart, the `__import__()` function. It represents a substantial step forward in documenting and defining the actions that take place during imports.

  (Contributed by Brett Cannon.)

5 Optimizations

Major performance enhancements have been added:

- The new I/O library (as defined in **PEP 3116**) was mostly written in Python and quickly proved to be a problematic bottleneck in Python 3.0. In Python 3.1, the I/O library has been entirely rewritten in C and is 2 to 20 times faster depending on the task at hand. The pure Python version is still available for experimentation purposes through the `_pyio` module.

  (Contributed by Amaury Forgeot d'Arc and Antoine Pitrou.)

- Added a heuristic so that tuples and dicts containing only untrackable objects are not tracked by the garbage collector. This can reduce the size of collections and therefore the garbage collection overhead on long-running programs, depending on their particular use of datatypes.

  (Contributed by Antoine Pitrou, issue 4688.)

- Enabling a configure option named `--with-computed-gotos` on compilers that support it (notably: gcc, SunPro, icc), the bytecode evaluation loop is compiled with a new dispatch mechanism which gives speedups of up to 20%, depending on the system, the compiler, and the benchmark.

  (Contributed by Antoine Pitrou along with a number of other participants, issue 4753).

- The decoding of UTF-8, UTF-16 and LATIN-1 is now two to four times faster.

  (Contributed by Antoine Pitrou and Amaury Forgeot d'Arc, issue 4868.)

- The `json` module now has a C extension to substantially improve its performance. In addition, the API was modified so that json works only with `str`, not with `bytes`. That change makes the module closely match the JSON specification which is defined in terms of Unicode.

  (Contributed by Bob Ippolito and converted to Py3.1 by Antoine Pitrou and Benjamin Peterson; issue 4136.)

- Unpickling now interns the attribute names of pickled objects. This saves memory and allows pickles to be smaller.

  (Contributed by Jake McGuire and Antoine Pitrou; issue 5084.)

6 IDLE

- IDLE's format menu now provides an option to strip trailing whitespace from a source file.

  (Contributed by Roger D. Serwy; issue 5150.)

7 Build and C API Changes

Changes to Python's build process and to the C API include:

- Integers are now stored internally either in base 2**15 or in base 2**30, the base being determined at build time. Previously, they were always stored in base 2**15. Using base 2**30 gives significant performance improvements on 64-bit machines, but benchmark results on 32-bit machines have been mixed. Therefore, the default is to use base 2**30 on 64-bit machines and base 2**15 on 32-bit machines; on Unix, there's a new configure option `--enable-big-digits` that can be used to override this default.

  Apart from the performance improvements this change should be invisible to end users, with one exception: for testing and debugging purposes there's a new `sys.int_info` that provides information about the internal format, giving the number of bits per digit and the size in bytes of the C type used to store each digit:

  ```
  >>> import sys
  >>> sys.int_info
  sys.int_info(bits_per_digit=30, sizeof_digit=4)
  ```

  (Contributed by Mark Dickinson; issue 4258.)

- The `PyLong_AsUnsignedLongLong()` function now handles a negative *pylong* by raising `OverflowError` instead of `TypeError`.

  (Contributed by Mark Dickinson and Lisandro Dalcrin; issue 5175.)

- Deprecated `PyNumber_Int()`. Use `PyNumber_Long()` instead.

  (Contributed by Mark Dickinson; issue 4910.)

- Added a new `PyOS_string_to_double()` function to replace the deprecated functions `PyOS_ascii_strtod()` and `PyOS_ascii_atof()`.

  (Contributed by Mark Dickinson; issue 5914.)

- Added `PyCapsule` as a replacement for the `PyCObject` API. The principal difference is that the new type has a well defined interface for passing typing safety information and a less complicated signature for calling a destructor. The old type had a problematic API and is now deprecated.

  (Contributed by Larry Hastings; issue 5630.)

8 Porting to Python 3.1

This section lists previously described changes and other bugfixes that may require changes to your code:

- The new floating point string representations can break existing doctests. For example:

```
def e():
    '''Compute the base of natural logarithms.

    >>> e()
    2.7182818284590451

    '''
    return sum(1/math.factorial(x) for x in reversed(range(30)))

doctest.testmod()

**********************************************************************
Failed example:
    e()
Expected:
    2.7182818284590451
Got:
```

```
    2.718281828459045
**********************************************************************
```

- The automatic name remapping in the pickle module for protocol 2 or lower can make Python 3.1 pickles unreadable in Python 3.0. One solution is to use protocol 3. Another solution is to set the *fix_imports* option to **False**. See the discussion above for more details.

Index

P