What's New in Python 2.7

リリース *2.7ja1*

A. M. Kuchling

2011 年 12 月 25 日

Python Software Foundation
Email: docs@python.org

目次

Author A.M. Kuchling (amk at amk.ca)

Release 2.7ja1

Date 2011 年 12 月 25 日

この文書は Python 2.7 の新機能について解説します。Python 2.7 は 2010 年 7 月 3 日にリリースされました。

数値周りの扱いが、浮動小数点数でも `Decimal` クラスでもいろいろな点で改良されました、標準ライブラリにいくつかの有用な追加が行われました。 `unittest` モジュールが大幅に改良され、コマンドラインオプション解析の `argpase` モジュールが追加され、 `collections` モジュールに便利な `OrderedDict` と `Counter` が追加され、他にもたくさんの改良が行われています。

Python 2.7 は 2.x の最終リリースになることが予定されているので、私たちはこのリリースを長期にわたって良いものになるように努力してきました。Python 3 への移植を助けるため、いくつかの新機能が Python 3.x から Python 2.7 に取り込まれています。

このドキュメントは新機能の完全な詳細を提供するのではなくて、簡易な概要を提供することを目的にしています。完全な詳細が知りたければ、 http://docs.python.org の Python 2.7 のドキュメントを参照してください。設計と実装の根拠を理解したい場合は、新機能に関する PEP を参照するか、 http://bugs.python.org に議論したい機能について登録してください。可能な限り、 “What’s New in Python” は各変更の bug や patch に対してリンクしています。

1 Python 2.x の今後

Python 2.7 は 2.x シリーズの最後のメジャーリリースになります。Python のメンテナは今後は Python 3.x シリーズの開発に集中する予定です。

これは、 2.7 が長期間残り続け、 Python 3.x への対応がされないプロダクションサービスを動かし続けることを意味します。2.7 が長期的に重要なリリースになるので、2 つの結論があります:

- これまでの 2.x バージョンに比べて長期間のメンテナンスが行われます。Python 2.7 は 3.x への移行が行われている間はメンテナンスされ続けます。Python 開発者はさらにそこから 2 年のバグフィックス

リリースをする予定です。

- 開発者向けの警告を抑止するポリシーが取られました。`DeprecationWarning` とそれを継承した警告は、指定されない限りは無視され、アプリケーションのユーザーがその警告を見ないですむようになりました。この変更は Python 3.2 ブランチにも適用されています。(stdlib-sig で討議され、 issue 7319 になりました)

  以前のリリースでは、`DeprecationWarning` メッセージはデフォルトで有効になっており、Python 開発者に、将来のバージョンの Python でそのコードのどこが動かなくなるかを明確に知らせていました。

  しかし、 Python で開発されたアプリケーションの利用者であって開発には携わっていない人がどんどん増えてきました。 `DeprecationWarning` メッセージはそれらのユーザーには無関係で、彼らをアプリケーションが本当に正しく動いているのか不安にさせ、ユーザーの質問に応えるための開発者の負担も増やしていました。

  `DeprecationWarning` メッセージの表示を有効にするには、*`-Wdefault`* (短い形式: *`-Wd`*) スイッチか、Python を実行する前に `PYTHONWARNINGS` 環境変数を `"default"` (か、 `"d"`) に設定します。Python コードから有効にする場合は `warnings.simplefilter('default')` とします。

2 Python 3.1 の機能

Python 2.6 は Python 3.0 から多くの機能を取り込み、 Python 2.7 は Python 3.1 の新しい機能のいくつかを取り込みました。2.x シリーズは 3.x シリーズへの移行のためのツールを提供し続けています。

Python 2.7 に取り込まれた 3.1 の機能の不完全なリスト:

- 集合のリテラル文法 (`{1,2,3}` は mutable set になります)
- 辞書と集合の内包表記 (`{i: i*2 for i in range(3)}`).
- 1 つの `with` 文で複数のコンテキストマネージャを扱えるように。
- `io` ライブラリの新バージョン。パフォーマンスのために C 言語で描き直されています。
- *PEP 372: collections* **に順序付き辞書を追加** で解説されている順序付き辞書
- *PEP 378: 1000* **区切りのための書式指定子** で解説されている新しい `","` フォーマット指定子
- `memoryview` オブジェクト
- `importlib` モジュールの小さいサブセット。 下に説明があります
- float `x` の `repr()` が多くの場合に短くなりました。これは `x` に戻せることが保証される最小の 10 進文字列です。以前のバージョンの Python では、 `float(repr(x))` が `x` になることだけが保証されていました。

- float から文字列、文字列から float への変換が正しく丸められるようになりました。`round()` 関数も正しく丸めるようになりました。
- 拡張モジュールが C API を提供するための、 `PyCapsule` 型
- `PyLong_AsLongAndOverflow()` C API 関数

新しい Python3 モード Warning:

- `operator.isCallable()` と `operator.sequenceIncludes()` は 3.x ではサポートされず、warning になります。
- *`-3`* オプションは自動的に *`-Qwarn`* スイッチを有効にして、整数や長整数に対する古い形式の除算が警告を出すようになります。

3 PEP 372: collections に順序付き辞書を追加

通常の Python 辞書は、 key/value ペアを不定の順序でイテレートします。何年にもわたり、いろいろな人が key の挿入順を保存する辞書の別実装を書いてきました。その経験に基づき、2.7 は `collections` モジュールに新しい `OrderedDict` クラスを追加しました。

`OrderedDict` API は通常の辞書と同じインタフェースを提供していますが、key/value をイテレートするときに key が最初に挿入された順番になることが保証されています。

```
>>> from collections import OrderedDict
>>> d = OrderedDict([('first', 1),
...                  ('second', 2),
...                  ('third', 3)])
>>> d.items()
[('first', 1), ('second', 2), ('third', 3)]
```

新しいエントリが既存のエントリを上書きした場合は、元の挿入順序が保持されます。

```
>>> d['second'] = 4
>>> d.items()
[('first', 1), ('second', 4), ('third', 3)]
```

エントリを削除して再挿入すると、順序は一番最後に移動します。

```
>>> del d['second']
>>> d['second'] = 5
>>> d.items()
[('first', 1), ('third', 3), ('second', 5)]
```

`popitem()` メソッドは、オプションの *last* 引数を持ち、でフォルトで True になっています。*last* が True の場合、一番最近に追加された key が返され、削除されます。False の場合、最も古い key が選ばれます。

```
>>> od = OrderedDict([(x,0) for x in range(20)])
>>> od.popitem()
(19, 0)
>>> od.popitem()
(18, 0)
>>> od.popitem(last=False)
(0, 0)
>>> od.popitem(last=False)
(1, 0)
```

2 つの順序付き辞書を比較するときは、 key/value だけでなく、挿入順も比較されます。

```
>>> od1 = OrderedDict([('first', 1),
...                    ('second', 2),
...                    ('third', 3)])
>>> od2 = OrderedDict([('third', 3),
...                    ('first', 1),
...                    ('second', 2)])
>>> od1 == od2
False
>>> # Move 'third' key to the end
>>> del od2['third']; od2['third'] = 3
>>> od1 == od2
True
```

`OrderedDict` を通常の辞書と比較すると、挿入順は無視されて単に key/value だけが比較されます。

`OrderedDict` の実装はこうなっています。key の 2 重リンクリストを管理し、新しい key が挿入されるときにリストに新しい key を追加します。2 つ目の辞書が key を対応するリストノードにマップします。なので、削除時にリンクリストを操作する必要はなく、コストは O(1) に保たれています。

標準ライブラリのいくつかのモジュールで、順序付き辞書の利用がサポートされています。

- `ConfigParser` モジュールはデフォルトで順序付き辞書を使います。設定ファイルを読み込み、編集した後、元の順序で書きなおすことができます。
- `collections.namedtuple()` の `_asdict()` メソッドは、タプルの順序と同じ順序の順序付き辞書を返すようになりました。
- `json` モジュールのデコーダーが `OrderedDict` をビルドするのをサポートするために、`JSONDecoder` クラスのコンストラクタに *object_pairs_hook* 引数が追加されました。PyYAML などの外部のライブラリでもサポートされています。

参考:

PEP 372 - Adding an ordered dictionary to collections PEP written by Armin Ronacher and Raymond Hettinger; implemented by Raymond Hettinger.

4 PEP 378: 1000 区切りのための書式指定子

大きい数値に区切り文字を追加して、 18446744073709551616 の代わりに 18,446,744,073,709,551,616 と出力すると、プログラムの出力を読みやすくできます。

これを行う一般的な方法は `locale` モジュールを使うことで、複数の区切り文字 (北米では ”,” で、ヨーロッパでは ”.”) を使ったり、複数のグループの大きさを使うことができます。しかし、 `locale` の利用方法は複雑ですし、スレッドごとに異なるロケールの出力を行うプログラムでは利用することができません。

そのため、シンプルなカンマによるグループ化機構が `str.format()` メソッドのミニ言語に追加されました。浮動小数点数をフォーマットする場合、シンプルにカンマを幅と精度の間に置きます。

```
>>> '{:20,.2f}'.format(18446744073709551616.0)
'18,446,744,073,709,551,616.00'
```

整数をフォーマットする場合は、幅の後にカンマを追加します。

```
>>> '{:20,d}'.format(18446744073709551616)
'18,446,744,073,709,551,616'
```

この機構は全く柔軟性を持っていません。区切り文字は常にカンマですし、グループは常に数字 3 つになります。カンマ書式機構は `locale` ほど汎用ではありませんが、手軽に使うことができます。

参考:

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

5 PEP 389: コマンドライン解析のための argparse モジュール

コマンドライン引数の解析のための `argparse` モジュールが、`optparse` モジュールのより協力な代替として追加されました。

これにより、 Python はコマンドライン引数の解析のために 3 つの異なるモジュール、 `getopt`, `optparse`, `argparse` を持つことになります。`getopt` モジュールは C 言語用ライブラリの `getopt()` 関数に似せてあるので、あとで C 言語で書き直すかもしれないプログラムのプロトタイプを Python で書く場合に役に立ちます。`optparse` は冗長になってしまいましたが、まだたくさんのスクリプトが利用していて、それらのスクリプトを自動的に更新する手段が無いので、削除される予定はありません。(`argparse` API を `optparse` のインタフェースに適合させる方法も検討されましたが、多くの複雑さと難点のために却下されました)

古いバージョンの Python との互換性を気にすること無く新しいスクリプトを書く時は、`optparse` の代わりに `argparse` を使ってください。

例:

```
import argparse

parser = argparse.ArgumentParser(description='Command-line example.')

# Add optional switches
parser.add_argument('-v', action='store_true', dest='is_verbose',
                    help='produce verbose output')
parser.add_argument('-o', action='store', dest='output',
                    metavar='FILE',
                    help='direct output to FILE instead of stdout')
parser.add_argument('-C', action='store', type=int, dest='context',
                    metavar='NUM', default=0,
                    help='display NUM lines of added context')

# Allow any number of additional arguments.
parser.add_argument(nargs='*', action='store', dest='inputs',
                    help='input filenames (default is stdin)')

args = parser.parse_args()
print args.__dict__
```

オーバーライドしない限り、 *-h* と *--help* スイッチが自動的に追加され、綺麗にフォーマットした出力を生成します。

```
-> ./python.exe argparse-example.py --help
usage: argparse-example.py [-h] [-v] [-o FILE] [-C NUM] [inputs [inputs ...]]

Command-line example.

positional arguments:
  inputs      input filenames (default is stdin)

optional arguments:
  -h, --help  show this help message and exit
  -v          produce verbose output
  -o FILE     direct output to FILE instead of stdout
  -C NUM      display NUM lines of added context
```

optparse と同じく、コマンドラインスイッチと引数は、 *dest* 引数の名前の属性をもったオブジェクトとして返されます。

```
-> ./python.exe argparse-example.py -v
{'output': None,
 'is_verbose': True,
 'context': 0,
 'inputs': []}

-> ./python.exe argparse-example.py -v -o /tmp/output -C 4 file1 file2
{'output': '/tmp/output',
 'is_verbose': True,
```

```
 'context': 4,
 'inputs': ['file1', 'file2']}
```

argparse は optparse よりも多くの便利なバリデーションを持っています。引数の正確な数を整数で指定したり、'*' で 0 以上の数を指定したり、'+' で 1 以上の数を指定したり、'?' でオプションにしたりできます。トップレベルのパーサーはサブパーサーを持つことができ、svn commit, svn update のように異なるオプションを持ったサブコマンドを定義できます。引数のタイプに FileType を指定することで、自動でファイルを開き、'-' が指定されたときに標準入出力だと判断することができます。

参考:

argparse のドキュメント argparse モジュールのドキュメント

argparse-from-optparse optparse を使うコードを変換する方法を説明した Python のドキュメントの一部

PEP 389 - argparse - New Command Line Parsing Module PEP written and implemented by Steven Bethard.

6 PEP 391: logging の辞書ベースの設定

logging モジュールは非常に柔軟です。アプリケーションは logging のサブシステムのツリーを定義できます。このツリーの各ロガーはいくつかのメッセージをフィルターし、異なるフォーマットを行い、メッセージを沢山の種類のハンドラーに渡します。

この柔軟性は、多くの設定を必要とします。オブジェクトを生成してプロパティを設定する Python コードを書くこともできますが、複雑なセットアップをしようとすると退屈なコードを書かないといけなくなります。logging は設定ファイルのパースを行う fileConfig() 関数を提供していますが、このファイルフォーマットはフィルターの設定をサポートしていませんし、プログラムで生成するのはさらに面倒になります。

Python 2.7 は logging の設定のために辞書を使う dictConfig() 関数を追加しました。いろいろな入力から辞書を作成する方法があります。コードで作ったり、 JSON ファイルをパースしたり、YAML のパーサーをインストールしてあればそれを使うことができます。詳しい情報は *logging-config-api* を参照してください。

次の例は 2 つのロガー、 root logger と “network” という名前の logger root logger に送られたメッセージは syslog プロトコルを利用してシステムに送られ、 “network” logger に送られたメッセージは 1MB ごとにローテートされる network.log ファイルに書きこまれます。

```
import logging
import logging.config

configdict = {
 'version': 1,     # Configuration schema in use; must be 1 for now
 'formatters': {
     'standard': {
         'format': ('%(asctime)s %(name)-15s '
                    '%(levelname)-8s %(message)s')}},
```

```
 'handlers': {'netlog': {'backupCount': 10,
                         'class': 'logging.handlers.RotatingFileHandler',
                         'filename': '/logs/network.log',
                         'formatter': 'standard',
                         'level': 'INFO',
                         'maxBytes': 1000000},
                'syslog': {'class': 'logging.handlers.SysLogHandler',
                           'formatter': 'standard',
                           'level': 'ERROR'}},

 # Specify all the subordinate loggers
 'loggers': {
               'network': {
                           'handlers': ['netlog']
               }
 },
 # Specify properties of the root logger
 'root': {
            'handlers': ['syslog']
 },
}

# Set up configuration
logging.config.dictConfig(configdict)

# As an example, log two error messages
logger = logging.getLogger('/')
logger.error('Database not found')

netlogger = logging.getLogger('network')
netlogger.error('Connection failed')
```

他にも、 `logging` モジュールには Vinary Sajip によって実装された 3 つの改良があります。

- `SysLogHandler` クラスは TCP 経由の syslog をサポートします。コンストラクタの *socktype* 引数は使用するソケットの種類として、 UDP を使う `socket.SOCK_DGRAM` と TCP を使う `socket.SOCK_STREAM` のどちらかを取ります。デフォルトは UDP のままです。
- `Logger` インスタンスに `getChild()` メソッドが追加されました。これは、相対パスで下位の logger を返します。例えば、 `log = getLogger('app')` として logger を取得した後、 `log.getChild('network.listen')` は `getLogger('app.network.listen')` と同じになります。
- `LoggerAdapter` クラスに `isEnabledFor()` メソッドが追加されました。 *level* を引数に取り、ベースの logger がその重要度レベルのメッセージを処理するかどうかを返します。

参考:

PEP 391 - Dictionary-Based Configuration For Logging PEP written and implemented by Vinay Sajip.

7 PEP 3106: 辞書 View

辞書の `keys()`, `values()`, `items()` メソッドは Python 3.x では動作が代わり、完全に実体化されたリストの代わりに、*view* と呼ばれるオブジェクトを返すようになりました。

Python 2.7 では、 `keys()`, `values()`, `items()` の動作を変えてしまうと、既存の大量のコードが動かなくなってしまうので、Python 3.x と同じ動作に合わせることはできません。なので、 Python 3.x のメソッドと同じ動作をするメソッドを、別の `viewkeys()`, `viewvalues()`, `viewitems()` という名前で追加しました。

```
>>> d = dict((i*10, chr(65+i)) for i in range(26))
>>> d
{0: 'A', 130: 'N', 10: 'B', 140: 'O', 20: ..., 250: 'Z'}
>>> d.viewkeys()
dict_keys([0, 130, 10, 140, 20, 150, 30, ..., 250])
```

View はイテレートするだけでなく、 set と似た利用をすることもできます。`&` 演算子で共通部分集合を、 `|` 演算子で話集合を取ることができます。

```
>>> d1 = dict((i*10, chr(65+i)) for i in range(26))
>>> d2 = dict((i**.5, i) for i in range(1000))
>>> d1.viewkeys() & d2.viewkeys()
set([0.0, 10.0, 20.0, 30.0])
>>> d1.viewkeys() | range(0, 30)
set([0, 1, 130, 3, 4, 5, 6, ..., 120, 250])
```

view は辞書とその辞書の変化に追随しています。

```
>>> vk = d.viewkeys()
>>> vk
dict_keys([0, 130, 10, ..., 250])
>>> d[260] = '&'
>>> vk
dict_keys([0, 130, 260, 10, ..., 250])
```

ただし、 view をイテレートしている間に key の追加や削除ができないことに気を付けてください。

```
>>> for k in vk:
...     d[k*2] = k
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size during iteration
```

Python 2.x で view メソッドを利用すると、2to3 が自動的にそれを通常の `keys()`, `values()`, `items()` メソッドに書き換えてくれます。

参考:

PEP 3106 - Revamping dict.keys(), .values() and .items() PEP written by Guido van Rossum. Backported to 2.7 by Alexandre Vassalotti; issue 1967.

8 PEP 3137: memoryview オブジェクト

memoryview オブジェクトは、他のオブジェクトのメモリの内容に対する bytes 型のインタフェースに合わせた view を提供します。

```
>>> import string
>>> m = memoryview(string.letters)
>>> m
<memory at 0x37f850>
>>> len(m)           # Returns length of underlying object
52
>>> m[0], m[25], m[26]    # Indexing returns one byte
('a', 'z', 'A')
>>> m2 = m[0:26]          # Slicing returns another memoryview
>>> m2
<memory at 0x37f080>
```

view の内容は bytes 型の文字列か整数のリストに変換することができます。

```
>>> m2.tobytes()
'abcdefghijklmnopqrstuvwxyz'
>>> m2.tolist()
[97, 98, 99, 100, 101, 102, 103, ... 121, 122]
>>>
```

memoryview オブジェクトは、対象となる背後のオブジェクトが変更可能 (mutable) な場合は、その変更を許可しています。

```
>>> m2[0] = 75
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot modify read-only memory
>>> b = bytearray(string.letters)  # Creating a mutable object
>>> b
bytearray(b'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
>>> mb = memoryview(b)
>>> mb[0] = '*'          # Assign to view, changing the bytearray.
>>> b[0:5]               # The bytearray has been changed.
bytearray(b'*bcde')
>>>
```

参考:

PEP 3137 - Immutable Bytes and Mutable Buffer PEP written by Guido van Rossum. Implemented by Travis Oliphant, Antoine Pitrou and others. Backported to 2.7 by Antoine Pitrou; issue 2396.

9 その他の言語の変更

Python 言語コアにいくつかの小さな変更が加えられました。

- set リテラルのためのシンタックスが Python 3.x からバックポートされました。内容を波括弧で囲うと mutable set になります。 dict リテラルとの区別は、コロンと value が存在しないことで行われます。なので、 `{}` は引き続き空の dict になります。空の set を作るときには `set()` を使ってください。

  ```
  >>> {1, 2, 3, 4, 5}
  set([1, 2, 3, 4, 5])
  >>> set() # empty set
  set([])
  >>> {}    # empty dict
  {}
  ```

  Backported by Alexandre Vassalotti; issue 2335.

- dict と set の内包表記も Python 3.x からバックポートされました。list と generator の内包表記を set と dict にも使えるように一般化させています。

  ```
  >>> {x: x*x for x in range(6)}
  {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
  >>> {('a'*x) for x in range(6)}
  set(['', 'a', 'aa', 'aaa', 'aaaa', 'aaaaa'])
  ```

- `with` 文が 1 つの文で複数のコンテキストマネージャーを使えるようになりました。コンテキストマネージャーは左から右へ処理され、それぞれが新しい `with` 文の開始となるように扱われます。つまり:

  ```
  with A() as a, B() as b:
      ... suite of statements ...
  ```

  このコードは、次のコードと等しくなります:

  ```
  with A() as a:
      with B() as b:
          ... suite of statements ...
  ```

  `contextlib.nested()` 関数は非常に似た機能を提供していたので、もう必要なくなり廃止予定となりました。

  (Proposed in http://codereview.appspot.com/53094; implemented by Georg Brandl.)

- 浮動小数点数と文字列の間の変換がほとんどのプラットフォームで正しく丸められるようになりました。この変換はいろいろな場面で発生します: float 型や complex 型に対する `str()` 関数の適用、数値フォーマット、 `marshal`, `pickle`, `json` モジュールを使ってのシリアライズとデシリアライズ、Python コード中の float や imaginary リテラルの解析、`Decimal` から float への変換などです。

  これに関連して、 浮動小数点数 *x* の `repr()` は、 (round-half-to-even 丸めモードで) 正しく丸め処理をした場合に元の *x* に戻せる最小の 10 進文字列になりました。以前は *x* を 17 桁の 10 進文字列に丸めていました。

丸めライブラリが、この改善が Windows や gcc, icc, suncc を使った Unix 環境で動かす役目をおっています。このライブラリの正確な動作が保証できない少しの環境があるので、そういったシステムではこのライブラリは利用されません。 `sys.float_repr_style` が `short` なら新しいコードが利用されていて、 `legacy` なら利用されていません。

Implemented by Eric Smith and Mark Dickinson, using David Gay's `dtoa.c` library; issue 7117.

- 多倍長整数や通常の整数から浮動小数点への変換でも丸め処理が変更され、元の数に一番近い浮動小数点値が返されるようになりました。これは浮動小数点へ完全に変換できる小さい整数では問題になりませんが、桁数に対して精度がどうしても足りない場合に関係します。Python 2.7 はより正確に近似するようになりました。例えば、 Python 2.6 では次のように計算されていました:

```
>>> n = 29514790517935289l
>>> float(n)
2.9514790517935283e+20
>>> n - long(float(n))
65535L
```

Python 2.7 の浮動小数点への変換結果は元の数値より大きくなりますが、元の値により近くなっています:

```
>>> n = 29514790517935289l
>>> float(n)
2.9514790517935289e+20
>>> n - long(float(n))
-1L
```

(Implemented by Mark Dickinson; issue 3166.)

この丸めの動作により、整数同士の true division (`from __future__ import division`) の結果もより正確になりました。(Also implemented by Mark Dickinson; issue 1811.)

- 複素数に対する暗黙の型強制は削除されました。インタプリタは complex オブジェクトに対して `__coerce__()` メソッドを呼び出そうとしません。(Removed by Meador Inge and Mark Dickinson; issue 5211.)

- `str.format()` メソッドは置換フィールドに対する自動ナンバリングをサポートするようになりました。これにより `str.format()` をより `%s` と同じように使えるようになりました。

```
>>> '{}:{}:{}'.format(2009, 04, 'Sunday')
'2009:4:Sunday'
>>> '{}:{}:{day}'.format(2009, 4, day='Sunday')
'2009:4:Sunday'
```

自動ナンバリングは左のフィールドから右のフィールドへと行われるので、最初の `{...}` 指定は `str.format()` メソッドの最初の引数を利用し、次の指定は次の引数を利用します。自動ナンバリングと明示的なナンバリングを混ぜて使うことはできず、全ての置換指定に手動でナンバリングするか、全てを自動ナンバリングに任せなければなりません。ただし、上の例の 2 つめのように、自動ナンバリングと名前フィールドを混ぜて使うことは可能です。(Contributed by Eric Smith; issue 5237.)

複素数が format() をサポートするようになりました。デフォルトでは右寄せになります。精度やカンマ区切りの指定は、実部と虚部のそれぞれに対して適用されます。アラインの指定は 1.5+3j のような複素数全体に対して適用されます。(Contributed by Eric Smith; issue 1588 and issue 7988.)

'F' 書式符号は常に大文字を使って出力するようになり、'INF' や 'NAN' が出力されるようになりました。(Contributed by Eric Smith; issue 3382.)

低レベルな変更: object.__format__() メソッドは PendingDeprecationWarning を発生させるようになりました。 object に対する __format__() メソッドはオブジェクトを文字列表現に変換してそれをフォーマットするからです。以前は、このメソッドは書式文字列を受け取り文字列表現を返していましたが、この仕様は Python コードのミスを隠してしまう可能性がありました。アライメントや精度のようなフォーマット指定情報を渡した時、そのなにかそのオブジェクトに合ったフォーマットがあることを期待しているはずなのに、 object.__format__() メソッドはそれを無視します。(Fixed by Eric Smith; issue 7994.)

- int() と long() 型に bit_length メソッドが追加されました。このメソッドはその値を 2 進数で表現したときに何ビット必要になるかを返します:

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

(Contributed by Fredrik Johansson and Victor Stinner; issue 3439.)

- import 文が、絶対インポート (例: from .os import sep) が失敗したときに相対インポートを試さなくなりました。これはバグ修正ですが、いままでたまたま動いていただけの import 文を動かなくしてしまう可能性があります。(Fixed by Meador Inge; issue 7902.)

- ビルトインの unicode 型のサブクラスが __unicode__() メソッドをオーバーライドできるようになりました。(Implemented by Victor Stinner; issue 1583863.)

- bytearray 型の translate() メソッドが第一引数に None を受け入れるようになりました。(Fixed by Georg Brandl; issue 4759.)

- @classmethod や @staticmethod を使ってクラスメソッドやスタティックメソッドを作成した時、そのラッパーオブジェクトがラップ対象となる関数を __func__ 属性で公開するようになりました。(Contributed by Amaury Forgeot d'Arc, after a suggestion by George Sakkis; issue 5982.)

- __slots__ を利用して属性を制限した場合に、設定されていない属性を del したときに AttributeError にならなかったのを修正しました。(Fixed by Benjamin Peterson; issue 7604.)

- 2 つの新しいエンコーディングが追加されました: "cp720" は主にアラビア文字に使われます。 "cp858"

は cp850 の一種ですが、ユーロ記号を追加しています。(CP720 contributed by Alexander Belchenko and Amaury Forgeot d'Arc in issue 1616979; CP858 contributed by Tim Hatch in issue 8016.)

- `file` オブジェクトは、POSIX 環境に置いてディレクトリを開こうとした時に発生する `IOError` 例外に `filename` 属性を設定するようになりました。(noted by Jan Kaliszewski; issue 4764) また、読み取りのみのファイルに対する書き込みを、C ライブラリ側のエラー検出・報告に頼るのではなく、明示的にチェックするようになりました。(fixed by Stefan Krah; issue 5677).
- Python の字句解析機は改行をそれ自身に変換するようになりました。それにより、組み込みの `compile()` 関数がどの改行コードを利用したコードでも受け付けるようになりました。また、コードの終端の改行も不要になりました。
- 関数宣言中の余分な丸括弧は Python 3.x では不正になりました。つまり、 `def f((x)): pass` は syntax error になります。Python3 警告モードでは、 Python 2.7 はこの利用方法について警告を出すようになりました。(Noted by James Lingard; issue 7362.)
- 古いスタイルのクラスのオブジェクトに対して弱参照を作れるようになりました。新スタイルクラスには以前から弱参照を利用できます。(Fixed by Antoine Pitrou; issue 8268.)
- モジュールオブジェクトが GC されたとき、モジュールの辞書は、他にその辞書への参照を持っているものがない場合にのみクリアされるようになりました。(issue 7140).

9.1 インタプリタの変更

新しい環境変数 `PYTHONWARNINGS` を使って警告を制御できるようになりました。この環境変数には、 *-W* スイッチに使われる警告の設定を、カンマ区切りの文字列として指定します。(Contributed by Brian Curtin; issue 7301.)

例えば、以下の設定は警告が発生するたびにそれを表示しますが、`Cookie` モジュールからの警告はエラーにします。(環境変数を設定するための文法は OS とシェルに依存します)

```
export PYTHONWARNINGS=all,error:::Cookie:0
```

9.2 最適化

いくつかの場面でパフォーマンスが向上しています。

- `with` 文の初期セットアップを行うための新しいオペコードが追加されました。`__enter__()` と `__exit__()` メソッドの検索を行います。(Contributed by Benjamin Peterson.)
- たくさんのオブジェクトを開放せずに確保していくという、よくある使い方の 1 つにおいて、GC のパフォーマンスが向上しました。以前はこの場合に GC にかかる時間はオブジェクトの数の 2 乗に比例していました。現在はフル GC の数は増えたヒープ上のオブジェクトの数に比例しています。新しい方式では、3 世代中 2 世代目に対する GC が 10 回実行されたうえで、2 世代目から 3 世代目に移っ

たオブジェクトの数が 3 世代目のオブジェクトの数の 10% を超えたときにフル GC が発生します。(Suggested by Martin von L　　wis and implemented by Antoine Pitrou; issue 4074.)

- GC が循環参照になりえないシンプルなコンテナ型の追跡を避けるようになりました。Python 2.7 では、アトミックな型 (整数、文字列など) のみを含むタプルと辞書は追跡されません。推移的に、アトミックな型のみを含むタプルを含む辞書も、同じく追跡されません。これにより、GC によって巡回されチェックされるオブジェクトの数が減るので、GC のコストを減らすことができます。(Contributed by Antoine Pitrou; issue 4688.)

- 多倍長整数の内部格納方式が、 2**15 ベースと 2**30 ベースのどちらかをビルド時に選択するようになりました。以前は常に 2**15 ベースで格納していました。2**30 ベースにすると 64bit マシンに置いては確実にパフォーマンスが向上しますが、32bit マシンではパフォーマンスが向上する場合も低下する場合もあります。なので、デフォルトでは 64bit マシンでは 2**30 ベースで、32bit マシンでは 2**15 ベースになるようになります。Unix では新しい configure オプションとして `--enable-big-digis` が追加され、このデフォルトの選択をオーバーライドできるようになっています。

  この変更はパフォーマンスの向上以外の形ではユーザーに見えないはずですが、1 つだけ例外があります。テストとデバッグを目的として、 `sys.long_info` というデータが追加され、内部フォーマットとして digit あたりのビット数と、それに使う C のデータ型のバイト数の情報を提供します:

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

  (Contributed by Mark Dickinson; issue 4258.)

  その他の変更により、多倍長整数オブジェクトのサイズは数バイト小さくなりました。32bit システムでは 2byte, 64bit システムでは 6byte 小さくなりました。(Contributed by Mark Dickinson; issue 5260.)

- 多倍長整数の除算アルゴリズムが、内部のループの軽量化や乗算からシフト演算への置き換え、不要なイテレーションの削除により高速化されました。いくつかのベンチマークによると、多倍長整数の剰余演算は 50% から 150% 高速化されています。(Contributed by Mark Dickinson; issue 5512.) ビット演算も高速化されています。(initial patch by Gregory Smith; issue 1087418)

- `%` の実装が、左側のオペランドが文字列型かどうかをチェックしてその場合の処理を特殊化しました。これにより文字列に対して `%` を頻繁に使う、テンプレートライブラリなどのアプリケーションのパフォーマンスが 1-3% 向上します。(Implemented by Collin Winter; issue 5176.)

- `if` 条件付きのリスト内包表記がより高速なバイトコードにコンパイルされるようになりました。(Patch by Antoine Pitrou, back-ported to 2.7 by Jeffrey Yasskin; issue 4715.)

- 整数や多倍長整数から 10 進文字列への変換を、任意の基数をサポートした汎用の変換関数を使う代わりに 10 進数に特殊化した処理を行うことで高速化しました。(Patch by Gawain Bolton; issue 6713.)

- 文字列等 (str, unicode, `bytearray`) の `split()`, `replace()`, `rindex()`, `rpartition()`, `rsplit()` メソッドが、1 文字ずつのスキャンの代わりに高速な逆方向スキャンアルゴリズムを使うようになりました。これにより、場合によっては 10 倍レベルの高速化になります。(Added by

Florent Xicluna; issue 7462 and issue 7622.)

- `pickle` と `cPickle` モジュールが、属性名に使われている文字列を自動的にインターンするようになりました。これにより unpickle 後のオブジェクトのメモリ使用量が減ります。(Contributed by Jake McGuire; issue 5084.)

- `cPickle` モジュールが辞書に対する特殊化を行い、 pickle にかかる時間をおよそ半分に減らしました。(Contributed by Collin Winter; issue 5670.)

10 新しいモジュールと改良されたモジュール

全てのリリースに置いて、 Python の標準ライブラリはたくさんの改良とバグ修正がされてきました。ここでは一部の注目に値する変更を、モジュール名で辞書順ソートしてリストアップしています。可燃な変更が見たければ、ソースツリー内の `Misc/NEWS` ファイルか、全ての完全な詳細が入っている Subversion のログを参照してください。

- `bdb` モジュールの基底デバッガクラス `Bdb` に、モジュールをスキップする機能が追加されました。コンストラクタは `django.*` のような glob スタイルのパターンをもった iterable を引数として受け取ります。デバッガはパターンのどれかにマッチするスタックフレームにステップインしません。(Contributed by Maru Newby after a suggestion by Senthil Kumaran; issue 5142.)

- `binascii` モジュールが buffer API をサポートするようになり、`memoryview` インスタンスやその他のバッファオブジェクトともに利用できるようになりました。(Backported from 3.x by Florent Xicluna; issue 7703.)

- `bsddb` モジュールが the pybsddb package の 4.7.2devel9 から 4.8.4 に更新されました。新しいバージョンはより Python 3.x との互換性がよくなり、いくつかのバグ修正と、いくつかの新しい BerkeleyDB のフラグとメソッドが追加されています。(Updated by Jes s Cea Avi n; issue 8156. The pybsddb changelog can be read at http://hg.jcea.es/pybsddb/file/tip/ChangeLog.)

- `bz2` モジュールの `BZ2File` がコンテキストマネージャープロトコルをサポートするようになりました。これにより、`with bz2.BZ2File(...) as f:` といった書き方ができます。(Contributed by Hagen F rstenau; issue 3860.)

- `collections` モジュールの新しい `Counter` クラスは、データの数え上げをするときに便利です。`Counter` インスタンスは辞書のように振る舞いますが、キーが存在しなかったときに `KeyError` 例外を発生させる代わりに 0 を返します。

```
>>> from collections import Counter
>>> c = Counter()
>>> for letter in 'here is a sample of english text':
...   c[letter] += 1
...
>>> c
Counter({' ': 6, 'e': 5, 's': 3, 'a': 2, 'i': 2, 'h': 2,
'l': 2, 't': 2, 'g': 1, 'f': 1, 'm': 1, 'o': 1, 'n': 1,
```

```
'p': 1, 'r': 1, 'x': 1})
>>> c['e']
5
>>> c['z']
0
```

- ConfigParser モジュールのパーサークラスのコンストラクタが *allow_no_value* 引数を受け取るようになりました。これはデフォルトでは False です。真の場合、値のないオプションが利用できるようになります。例えば:

```
>>> import ConfigParser, StringIO
>>> sample_config = """
... [mysqld]
... user = mysql
... pid-file = /var/run/mysqld/mysqld.pid
... skip-bdb
... """
>>> config = ConfigParser.RawConfigParser(allow_no_value=True)
>>> config.readfp(StringIO.StringIO(sample_config))
>>> config.get('mysqld', 'user')
'mysql'
>>> print config.get('mysqld', 'skip-bdb')
None
>>> print config.get('mysqld', 'unknown')
Traceback (most recent call last):
  ...
NoOptionError: No option 'unknown' in section: 'mysqld'
```

  (Contributed by Mats Kindahl; issue 7005.)

- contextlib.nested() 関数が廃止予定になりました。これは 1 つの with 文で 1 つ以上のコンテキストマネージャーを扱うためのものでしたが、 with 文が複数のコンテキストマネージャーをサポートするようになりました。

- cookielib モジュールが不正な、値が整数値ではないバージョンフィールドを持つ cookie を無視するようになりました。(Fixed by John J. Lee; issue 3924.)

- copy モジュールの deepcopy() 関数が、束縛ずみインスタンスメソッドを正しくコピーするようになりました。(Implemented by Robert Collins; issue 1515.)

- ctypes モジュールが、ポインタとして宣言された引数に対して None が渡された場合常に C の NULL ポインタに変換するようになりました。(Changed by Thomas Heller; issue 4606.) 基盤となっている libffi ライブラリ が 3.0.9 にアップデートされ、いくつかのプラットフォームにおけるいろいろな問題が修正されました。(Updated by Matthias Klose; issue 8142.)

- 新しいメソッド: datetime モジュールの timedelta クラスに total_seconds() メソッドが追加されました。これはその期間の秒数を返します。(Contributed by Brian Quinlan; issue 5788.)

- 新しいメソッド: Decimal クラスに、浮動小数点数から Decimal への正確な変換を

行う `from_float()` クラスメソッドが追加されました。この正確な変換は、浮動小数点数が示す値に一番近い近似となる 10 進数を求めます。なので、このメソッドを使ったとしてもある程度の誤差は残ります。例えば、 `Decimal.from_float(0.1)` は `Decimal('0.1000000000000000055511151231257827021181583404541015625')` を返します。(Implemented by Raymond Hettinger; issue 4796.)

`Decimal` のインスタンスと浮動小数点数の比較は、オペランドの値に応じた賢い結果を返すようになりました。以前はこの比較は Python のデフォルトのオブジェクト比較にフォールバックし、オペランドの型に応じて不定な結果が返されていました。ただし、 `Decimal` と浮動小数点数を加算などの他の演算で組み合わせることはいまでも許可されていません。浮動小数点数と `Decimal` の間の変換方法を明示的に指定するべきです。(Fixed by Mark Dickinson; issue 2531.)

`Decimal` のコンストラクタが浮動小数点数を受け入れるようになりました。(added by Raymond Hettinger; issue 8257) また、アラビア・インド数字などの、ヨーロッパ以外の Unicode 文字も受け入れるようになりました。(contributed by Mark Dickinson; issue 6595).

`Context` クラスのほとんどのメソッドが `Decimal` インスタンスと同じく整数を受け入れるようになりました。例外は `caonical()` と `is_canonical()` メソッドだけです。 (Patch by Juan Jos Conti; issue 7633.)

`Decimal` インスタンスを文字列の `format()` メソッドで利用した場合、デフォルトのアライメントが左揃えから、数値型に適した右揃えに変更されました。 (Changed by Mark Dickinson; issue 6857.)

signaling NaN (あるいは `sNAN`) との比較は、暗黙に比較演算に応じて真偽値を返すのではなく、`InvalidOperation` 例外を発生させるようになりました。 Quiet NaN (あるいは `NAN`) はハッシュ可能になりました。(Fixed by Mark Dickinson; issue 7279.)

- `difflib` モジュールの出力が、ファイル名を含むヘッダのセパレータにスペースではなくタブ文字を使うようになり、最近の **diff/patch** ツールとの互換性が高くなりました。(Fixed by Anatoly Techtonik; issue 7585.)

- Distutils `sdist` コマンドが、 `MANIFEST.in` や `setup.py` ファイルが変更されていなくても、ユーザーが作成したかもしれない新しいファイルが含まれるようにするために、毎回 `MANIFEST` ファイルを再生成するようになりました。(Fixed by Tarek Ziad ; issue 8688.)

- `doctest` モジュールの `IGNORE_EXCEPTION_DETAIL` フラグが、テストされている例外を含むモジュールの名前を無視するようになりました。(Patch by Lennart Regebro; issue 7490.)

- `email` モジュールの `Message` クラスが Uniode 値のペイロードを受け入れるようになり、自動的に `output_charset` で指定されたエンコーディングに変換するようになりました。(Added by R. David Murray; issue 1368247.)

- `Fraction` クラスが、コンストラクタの引数として、1 つの float 値、`Decimal` インスタンス、2 つの有理数を受け入れるようになりました。(Implemented by Mark Dickinson; rationals added in issue 5812, and float/decimal in issue 8294.)

  fraction と complex 数の間の順序比較演算 (<, <=, >, >=) が `TypeError` 例外を発生させるようになり

ました。これは過失を修正し、 `Fraction` を他の数値型とマッチさせます。

- 新しいクラス: `FTP_TLS` が `ftplib` に追加されました。認証とその後の制御・データ転送を TLS カプセル化することでセキュアな FTP 接続を提供します。(Contributed by Giampaolo Rodola; issue 2054.)

  バイナリデータのアップロードを行う `storbinary()` に *rest* 引数が追加され、アップロードのリスタートができるようになりました。(patch by Pablo Mouzo; issue 6845.)

- 新しいクラスデコレータ: `functools` モジュールに `total_ordering()` が追加されました。`__eq__()` と、 `__lt__()`, `__le__()`, `__gt__()`, `__ge__()` のいづれか 1 つを定義したクラスを受け取り、残りの比較メソッドを生成します。`__cmp__()` メソッドが Python 3.x では廃止されたので、順序のあるクラスを定義するのを簡単にするためにこのデコレータがあります。(Added by Raymond Hettinger; issue 5479.)

  新しい関数: `cmp_to_key()` 関数は、古いスタイルの 2 引数を受け取る比較関数を受け取り、`sorted()`, `min()`, `max()` などの *key* 引数に利用できる呼び出し可能オブジェクトを返します。この関数の主な目的は、古いコードを Python 3.x へ対応させるのを手助けすることです。(Added by Raymond Hettinger.)

- 新しい関数: `gc` モジュールに `is_tracked()` 関数が追加されました。渡されたオブジェクトが GC に追跡されている場合に True を、そうでない場合に False を返します。 (Contributed by Antoine Pitrou; issue 4688.)

- `gzip` モジュールの `GzipFile` がコンテキストマネージャープロトコルをサポートしました。 `with gzip.GzipFile(...) as f:` と書くことができます。(contributed by Hagen F rstenau; issue 3860 また、 `io.BufferedIOBase` ABC を実装しています。なので、より高速な処理のために `io.BufferedReader` でラップすることができます。(contributed by Nir Aides; issue 7471). さらに、コンストラクタにオプションのタイムスタンプを指定することで、gzip ファイル内の変更時間レコードをオーバーライドすることができるようになりました。 (Contributed by Jacques Frechet; issue 4272.)

  gzip ファイルフォーマットのファイルは最後にゼロバイトによるパディングができるようになりました。 `gzip` モジュールは末尾のゼロバイトを消費します。(Fixed by Tadek Pietraszek and Brian Curtin; issue 2846.)

- 新しい属性: `hashlib` モジュールに `algorithms` 属性が追加されました。これはサポートしているアルゴリズムを含むタプルです。Python 2.7 では、`hashlib.algorithms` は `('md5', 'sha1', 'sha224', 'sha256', 'sha384', 'sha512')` が含まれています。(Contributed by Carl Chenet; issue 7418.)

- `httplib` モジュールで使われているデフォルトの `HTTPResponse` クラスがバッファリングをサポートし、HTTP レスポンスをより高速に読み込めるようになりました。(Contributed by Kristj n Valur J nsson; issue 4879.)

  `HTTPConnection` と `HTTPSConnection` クラスが *source_address* 引数として `(host, port)` の 2 要素タプルを受け取るようになりました。これは接続のソースアドレスとして利用されます。(Contributed by Eldon Ziegler; issue 3972.)

- `ihooks` モジュールが相対 import をサポートしました。`ihooks` import をカスタマイズするための古いモジュールで、Python 2.0 で追加された `imputil` モジュールに取って変わられていることに注意してください。(Relative import support added by Neil Schemenauer.)

- `imaplib` モジュールが IPv6 アドレスをサポートするようになりました。(Contributed by Derek Morr; issue 1655.)

- `inspect` モジュールに `getcallargs()` 関数が追加されました。callable とその位置引数、キーワード引数を受け取り、callable のどの仮引数がどの実引数を受け取るかを計算し、引数名から値へマップする辞書を返します。例えば:

```
>>> from inspect import getcallargs
>>> def f(a, b=1, *pos, **named):
...     pass
>>> getcallargs(f, 1, 2, 3)
{'a': 1, 'b': 2, 'pos': (3,), 'named': {}}
>>> getcallargs(f, a=2, x=4)
{'a': 2, 'b': 1, 'pos': (), 'named': {'x': 4}}
>>> getcallargs(f)
Traceback (most recent call last):
...
TypeError: f() takes at least 1 argument (0 given)
```

  Contributed by George Sakkis; issue 3135.

- 更新されたモジュール: `io` ライブラリが、Python 3.1 に同梱されるバージョンに更新されました。3.1 では、I/O ライブラリは完全に C で書き直され、処理されるタスクに依って 2 から 20 倍速くなりました。元の Python バージョンは `_pyio` モジュールに改名されました。

  結果の軽微な変更: `io.TextIOBase` クラスは、エンコーディングやデコーディングのエラーに使われるエラー設定 (`'strict'`, `'replace'`, `'ignore'` のいずれか) を与える `errors` 属性を持つようになりました。

  `io.FileIO` クラスは、無効なファイルディスクリプタを渡されたときに `OSError` を送出するようになりました。(Implemented by Benjamin Peterson; issue 4991.) `truncate()` メソッドは、ファイル位置を保存するようになりました。以前は、ファイル位置を新しいファイルの末尾に変更しました。(Fixed by Pascal Chambon; issue 6939.)

- 新しい関数: `itertools.compress(data, selectors)` は、2 つのイテレータを取ります。*data* の要素は、*selectors* の対応する値が真であれば返されます:

```
itertools.compress('ABCDEF', [1,0,1,0,1,1]) =>
  A, C, E, F
```

  新しい関数: `itertools.combinations_with_replacement(iter, r)` は、*iter* から、すべての可能な長さ *r* の要素の組合せを返します。`combinations()` とは異なり、生成された組合せに個別の要素が繰り返し出現できます:

```
itertools.combinations_with_replacement('abc', 2) =>
  ('a', 'a'), ('a', 'b'), ('a', 'c'),
  ('b', 'b'), ('b', 'c'), ('c', 'c')
```

なお要素は、実際の値ではなく、その入力内での位置に依って一意であるとみなされます。

`itertools.count()` 関数は、1 以外の値を増分できるように、*step* 引数を追加しました。`count()` はまた、キーワード引数に対応し、さらに浮動小数点数や `Decimal` インスタンスのような、非整数の値を使えるようになりました。(Implemented by Raymond Hettinger; issue 5032.)

`itertools.combinations()` および `itertools.product()` は、以前は入力のイテレート可能オブジェクトより大きい *r* に `ValueError` を返していました。これは仕様エラーと認められ、空のイテレータを返すようになりました。(Fixed by Raymond Hettinger; issue 4816.)

- 更新されたモジュール: `json` モジュールが、エンコーディングやデコーディングを高速化する C 拡張を含んだ、 simplejson パッケージのバージョン 2.0.9 にアップグレードされました。(Contributed by Bob Ippolito; issue 4136.)

  新しい `collections.OrderedDict` 型をサポートするために、`json.load()` はオプションとして、任意のオブジェクトリテラルに対して呼び出され、ペアのリストにデコードする *object_pairs_hook* パラメタを追加しました。(Contributed by Raymond Hettinger; issue 5381.)

- `mailbox` モジュールの `Maildir` クラスは、読み込むディレクトリのタイムスタンプを記録し、その後に更新時刻が変わった場合にのみ再読み込みするようになりました。これは不必要なディレクトリ走査を避けることでパフォーマンスを向上させます。A.M. Kuchling and Antoine Pitrou; issue 1607951, issue 6896.)

- 新しい関数: `math` モジュールは、誤差関数と相補誤差関数 `erf()` および `erfc()` 、`e**x - 1` を `exp()` から 1 を引くのよりも高い精度で計算する `expm1()` 、ガンマ関数 `gamma()` 、そしてガンマ関数の自然対数 `lgamma()` を追加しました。(Contributed by Mark Dickinson and nirinA raseliarison; issue 3366.)

- `multiprocessing` モジュールの `Manager*` クラスに、サブプロセスの開始時に呼び出される呼び出し可能オブジェクトと、それに渡される引数群を渡すことができるようになりました。(Contributed by lekma; issue 5585.)

  ワーカプロセスのプールを制御する `Pool` クラスに、オプションの *maxtasksperchild* パラメタが追加されました。ワーカプロセスはこれで指定された数のタスクを処理したら、退出して `Pool` に新しいワーカーを開始させます。これは、タスクがメモリその他のリソースをリークし得るときや、ワーカをとても大きくしてしまうようなタスクがあるときに便利です。(Contributed by Charles Cazabon; issue 6963.)

- `nntplib` モジュールは、IPv6 アドレスをサポートするようになりました。(Contributed by Derek Morr; issue 1664.)

- 新しい関数: `os` モジュールは、以下の POSIX システムコールをラップします: real, effective, および saved GID と UID を返す、`getresgid()` と `getresuid()` 、real, effective, および saved GID と

UID を新しく設定する、`setresgid()` と `setresuid()` 、現在のプロセスにグループアクセスリストを初期化する `initgroups()` 。(GID/UID functions contributed by Travis H.; issue 6508. Support for initgroups added by Jean-Paul Calderone; issue 7333.)

`os.fork()` 関数は、子プロセス中で import ロックを再初期化するようになりました。これは、Solaris における `fork()` がスレッドから呼び出された時の問題を修正します。(Fixed by Zsolt Cserna; issue 7242.)

- `os.path` モジュールの、 `normpath()` および `abspath()` 関数は、Unicode を保存するようになりました。入力パスが Unicode なら、戻り値も Unicode 文字列になります。(`normpath()` fixed by Matt Giuca in issue 5827; `abspath()` fixed by Ezio Melotti in issue 3426.)

- `pydoc` モジュールは、Python が使う様々なシンボルのヘルプを追加しました。例えば、`help('<<')` や `help('@')` とできるようになりました。(Contributed by David Laban; issue 4739.)

- `re` モジュールの `split()`, `sub()`, および `subn()` は、モジュールの他の関数との一貫性のため、オプションの *flags* 引数を受け付けるようになりました。(Added by Gregory P. Smith.)

- 新しい関数: `runpy` モジュールの `run_path()` は、与えられた *path* 引数のコードを実行します。 *path* は Python ソースファイルのパス (`example.py`)、コンパイル済みバイトコードファイル (`example.pyc`)、ディレクトリ (`./package/`)、または zip アーカイブ (`example.zip`) にできます。ディレクトリか zip パスが与えられると、それが `sys.path` の最初に加えられ、モジュール `__main__` が import されます。これは、ディレクトリや zip が `__main__.py` を含むことを期待します。なければ、他の `__main__.py` が `sys.path` の後の部分から import されることがあります。これにより、Python のコマンドラインが明示的なパス名を処理する方法を真似たいようなスクリプトに、 `runpy` のより多くの機構が利用できるようになります。(Added by Nick Coghlan; issue 6816.)

- 新しい関数: `shutil` モジュールの `make_archive()` は、ファイル名、アーカイブ型 (zip または tar-format)、およびディレクトリパスを取って、そのディレクトリの内容を含むアーカイブを生成します。(Added by Tarek Ziad .)

  `shutil`'s の `copyfile()` および `copytree()` 関数は、名前付きパイプのコピーを求められたときに `SpecialFileError` 例外を送出する様になりました。以前は、コードは、名前付きパイプを、通常のファイルのように読み込み用に開いて扱い、いつまでもブロックしていました。(Fixed by Antoine Pitrou; issue 3002.)

- `signal` モジュールは、シグナルハンドラを本当に必要でない限り再インストールしなくなりました。これで、EINTR シグナルを強く捕えられないバグが修正されました。(Fixed by Charles-Francois Natali; issue 8354.)

- 新しい関数: `site` モジュールの、新しい関数群は様々なサイトおよびユーザに特有のパスを返します。`getsitepackages()` は、全てのグローバル site-packages ディレクトリを含むリストを返します。`getusersitepackages()` は、ユーザの site-packages ディレクトリのパスを返します。そして、`getuserbase()` は、データの保存に使えるディレクトリへのパスを与える、 `USER_BASE` 環境変数の値を返します。(Contributed by Tarek Ziad ; issue 6693.)

  `site` モジュールは `sitecustomize` モジュールが import されたときに例外を報告するようにな

り、 KeyboardInterrupt 例外が捕捉されて飲み込まれることはなくなりました。(Fixed by Victor Stinner; issue 3137.)

- create_connection() 関数は、接続に使われるソースアドレスを与える (host, port) の 2-タプル、 *source_address* パラメタを追加しました。(Contributed by Eldon Ziegler; issue 3972.)

  recv_into() および recvfrom_into() メソッドは、バッファ API をサポートするオブジェクト、最も便利なのは bytearray や memoryview 、に書きこむようになりました。(Implemented by Antoine Pitrou; issue 8104.)

- SocketServer モジュールの TCPServer クラスは、ソケットタイムアウトと Nagle アルゴリズムの無効化をサポートするようになりました。 disable_nagle_algorithm クラス属性は、デフォルトで False です。上書きされて True になると、新しいリクエスト接続は、TCP_NODELAY オプションを設定され、一つの TCP パケットにたくさんの小さな送信がバッファリングされることを防ぎます。timeout クラス属性は、リクエストソケットに適用されるタイムアウトを秒で保持できます。その時間内にリクエストが受け付けられなければ、handle_timeout() が呼び出され、handle_request() が返されます。(Contributed by Kristj n Valur J nsson; issue 6192 and issue 6267.)

- 更新されたモジュール: sqlite3 モジュールが pysqlite package のバージョン 2.6.0 にアップデートされました。バージョン 2.6.0 はいくつかのバグ修正を含み、共有ライブラリから SQLite 拡張をロードできるようになりました。拡張を有効にするには enable_load_extension(True) メソッドを呼び出し、そして特定の共有ライブラリをロードするには enable_load_extension(True) を呼び出してください。(Updated by Gerhard H ring.)

- ssl モジュールの SSLSocket オブジェクトは、バッファ API をサポートするようになりました。これは、テストスイートの障害を修正します (fix by Antoine Pitrou; issue 7133)。また、OpenSSL の SSL_MODE_AUTO_RETRY を自動的に設定することで、recv() 演算によって SSL 再ネゴシエーションを引き起こすエラーコードが返されるのを防ぎます (fix by Antoine Pitrou; issue 8222)。

  ssl.wrap_socket() コンストラクタ関数は、許可される暗号化アルゴリズムを列挙する文字列である *ciphers* 引数を取るようになりました。この文字列の書式は、 OpenSSL ドキュメントで. 解説されています。(Added by Antoine Pitrou; issue 8322.)

  その他の変更により、拡張は OpenSSL の全ての暗号をロードし、それらすべてが利用できるようアルゴリズムを消化します。SSL 暗号化の中には検証できないものもあり、”unknown algorithm” エラーを報告します。(Reported by Beda Kosata, and fixed by Antoine Pitrou; issue 8484.)

  OpenSSL の、使われるバージョンは、モジュール属性 ssl.OPENSSL_VERSION (文字列)、ssl.OPENSSL_VERSION_INFO (5-タプル)、および ssl.OPENSSL_VERSION_NUMBER (整数) として利用できるようになりました。(Added by Antoin Pitrou; issue 8321.)

- struct モジュールは、値が特定の整数フォーマットコード (bBhHiIlLqQ のいずれか) に対して大きすぎるときに、オーバーフローエラーを静かに無視しなくなりました。これは必ず struct.error 例外を送出するようになりました。(Changed by Mark Dickinson; issue 1523.) pack() 関数は、非整数を変換してパックするのに、__int__() メソッドを試したりエラーを報告したりする前に、

__index__() の使用も試すようになりました。(Changed by Mark Dickinson; issue 8300.)

- 新しい関数: subprocess モジュールの check_output() は、指定された引数群でコマンドを実行し、コマンドがエラーを起こさずに実行したらコマンドの出力を文字列として返し、そうでなければ CalledProcessError 例外を送出します。

```
>>> subprocess.check_output(['df', '-h', '.'])
'Filesystem     Size   Used  Avail Capacity  Mounted on\n
/dev/disk0s2    52G    49G   3.0G    94%    /\n'

>>> subprocess.check_output(['df', '-h', '/bogus'])
  ...
subprocess.CalledProcessError: Command '['df', '-h', '/bogus']' returned non-zero exit statu
```

(Contributed by Gregory P. Smith.)

subprocess モジュールは、 EINTR シグナルを受け取り次第内部システムコールを最施行するようになりました。(Reported by several people; final patch by Gregory P. Smith in issue 1068268.)

- 新しい関数: symtable モジュールの is_declared_global() は、明示的に global と宣言された引数には真、暗示的に global である引数には偽を返します。(Contributed by Jeremy Hylton.)

- syslog モジュールは、識別子として、以前デフォルトであった'python' の値ではなく、sys.argv[0] の値を使うようになりました。(Changed by Sean Reifschneider; issue 8451.)

- sys.version_info の値は、属性名 major, minor, micro, releaselevel, および serial を持つ名前付きタプルになりました。

  sys.getwindowsversion() もまた、属性名 major, minor, build, platform, service_pack, service_pack_major, service_pack_minor, suite_mask, および product_type を持つ名前付きタプルを返します。(Contributed by Brian Curtin; issue 7766.)

- tarfile モジュールのデフォルトエラー処理が変更され、致命的なエラーを抑制しないようになりました。デフォルトのエラーレベルは以前は 0 で、エラーはメッセージとしてデバッグログに書き込まれるだけでしたが、デバッグログはデフォルトでは活性化されていないため、エラーは顧みられませんでした。デフォルトのエラーレベルは 1 になり、エラーがあれば例外が送出されます。(Changed by Lars Gust bel; issue 7357.)

  tarfile は、tar ファイルに追加される TarInfo オブジェクトのフィルタリングをサポートするようになりました。add() を呼び出したとき、オプションの呼び出し可能オブジェクトである *filter* 引数を与えることができます。*filter* オブジェクトには、ファイルが追加される度に TarInfo を渡され、これを変形して返せます。このオブジェクトが None を返したら、そのファイルは結果のアーカイブから除かれます。これは既存の *exclude* 引数より強力で、それゆえこれは非推奨となります。(Added by Lars Gust bel; issue 6856.) TarFile クラスはまた、コンテキストマネージャプロトコルもサポートするようになりました。(Added by Lars Gust bel; issue 7232.)

- threading.Event クラスの wait() メソッドは、終了時に内部フラグを返すようになりました。wait() は内部フラグが真になるまでブロックすることを想定されているため、この関数は通常真を返

すことになります。この戻り値は、タイムアウトが与えられ、オペレーションがタイムアウトした時のみ偽になります。(Contributed by Tim Lesher; issue 1674032.)

- unicodedata モジュールに提供される Unicode データベースは、どの文字が数字や空白文字か、または改行を表すかを決定するために、内部で使われるようになりました。このデータベースはまた、Unihan.txt データファイルからの情報も含みます (patch by Anders Chrigstr m and Amaury Forgeot d'Arc; issue 1571184)。また、バージョン 5.2.0 にアップデートされました (updated by Florent Xicluna; issue 8024)。

- urlparse モジュールの urlsplit() は、未知の URL スキームを **RFC 3986** に応じた方法で処理します。URL が "<something>://..." の形式なら、 :// の前のテキストを、それがモジュールの知らない作り物のスキームであってさえ、スキームとして扱います。この変更は、古い働きを使って動作していたコードを破壊することがあります。例えば、Python 2.6.4 や 2.5 は以下を返します:

```
>>> import urlparse
>>> urlparse.urlsplit('invented://host/filename?query')
('invented', '', '//host/filename?query', '', '')
```

Python 2.7 (や Python 2.6.5) は以下を返します:

```
>>> import urlparse
>>> urlparse.urlsplit('invented://host/filename?query')
('invented', 'host', '/filename?query', '', '')
```

(Python 2.7 では、これは普通のタプルではなく名前付きタプルを返すので、実際は微妙に異なる出力をします。)

urlparse モジュールは、 **RFC 2732** で定義された IPv6 リテラルアドレスもサポートします (contributed by Senthil Kumaran; issue 2987)。:

```
>>> urlparse.urlparse('http://[1080::8:800:200C:417A]/foo')
ParseResult(scheme='http', netloc='[1080::8:800:200C:417A]',
            path='/foo', params='', query='', fragment='')
```

- 新しいクラス: weakref モジュールの WeakSet クラスは、その要素の弱参照だけを保持する集合です。要素は、それを指す参照がなくなり次第除去されます。(Originally implemented in Python 3.x by Raymond Hettinger, and backported to 2.7 by Michael Foord.)

- エレメントツリーライブラリ、xml.etree は、(<?xml-stylesheet href="#style1"?> のような) 命令や (<!-- comment --> のような) コメントを処理する XML を出力するとき、アンパサンドや山括弧をエスケープしなくなりました。(Patch by Neil Muller; issue 2746.)

- xmlrpclib と SimpleXMLRPCServer によって提供される XML-RPC クライアントおよびサーバは、HTTP/1.1 keep-alive をサポートし、場合によっては gzip エンコーディングを使って XML の送受を圧縮することで、パフォーマンスが向上しました。gzip 圧縮は SimpleXMLRPCRequestHandler の encode_threshold 属性によって制御されます。これはバイト数で、これより大きな応答は圧縮されます。(Contributed by Kristj n Valur J nsson; issue 6267.)

- zipfile モジュールの ZipFile は、コンテキストマネジメントプロトコルをサポートし、 with zipfile.ZipFile(...) as f: と書けるようになりました。(Contributed by Brian Curtin; issue 5511.)

  zipfile は、空のディレクトリのアーカイブ化をサポートするようになり、これを正しく解凍します。(Fixed by Kuba Wieczorek; issue 4710.) アーカイブ内のファイルの読み込みが速くなり、 read() および readline() の割り込みが正しく働くようになりました。(Fixed by Kuba Wieczorek; issue 4710.)

  is_zipfile() 関数は、以前受け付けられていたパス名に加え、ファイルオブジェクトも受け付けるようになりました。(Contributed by Gabriel Genellina; issue 4756.)

  writestr() メソッドは、 ZipFile コンストラクタで指定されたデフォルトの圧縮メソッドをオーバライドする、オプションの *compress_type* パラメタを追加しました。(Contributed by Ronald Oussoren; issue 6003.)

10.1 新しいモジュール: importlib

Python 3.1 は、根底の Python の import 文のロジックの最実装である importlib パッケージを含みます。importlib は、Python インタプリタの実装者や、import プロセスに関与する新しいインポータを書きたいと願うユーザにとって便利です。Python 2.7 は importlib パッケージを完全には含みませんが、代わりに 1 つの関数 import_module() を含む小さなサブセットがあります。

import_module(name, package=None) はモジュールを import します。*name* はモジュールまたはパッケージの名前を含む文字列です。..utils.errors のように . 文字で始まる文字列を与えることで、相対 import も可能です。相対 import では、 *package* 引数は必ず与えられなければならず、相対 import のアンカーとして使われるパッケージの名前でなければなりません。 import_module() は、import されたモジュールを sys.modules に挿入し、モジュールオブジェクトを返すこともします。

ここに例があります:

```
>>> from importlib import import_module
>>> anydbm = import_module('anydbm')  # Standard absolute import
>>> anydbm
<module 'anydbm' from '/p/python/Lib/anydbm.py'>
>>> # Relative import
>>> file_util = import_module('..file_util', 'distutils.command')
>>> file_util
<module 'distutils.file_util' from '/python/Lib/distutils/file_util.pyc'>
```

importlib は、Brett Cannon によって実装され、 Python 3.1 に導入されました。

10.2 新しいモジュール: sysconfig

sysconfig モジュールが Distutils パッケージから引き抜かれ、新しいトップレベルモジュールになりました。sysconfig は、Python のビルドプロセスについての以下の情報を得るための関数群を提供しています。

コンパイラスイッチ、インストレーションパス、プラットフォーム名、そして Python がソースディレクトリから実行されているかどうかです。

モジュールの関数のいくつかを紹介すると:

- `get_config_var()` は、Python の Makefile と `pyconfig.h` ファイルから変数を返します。
- `get_config_vars()` は、すべての環境設定変数を含む辞書を返します。
- `get_path()` は、特定のタイプのモジュール、つまり標準ライブラリ、サイト特有のモジュール、プラットフォーム特有のライブフラリなど、への設定されたパスを返します。
- `is_python_build()` は、Python ソースツリーからバイナリを起動していれば真を、そうでなければ偽を返します。

詳細と関数の完全な一覧は `sysconfig` ドキュメントを参照してください。

Distutils パッケージと `sysconfig` は Tarek Ziad　によってメンテナンスされていて、彼は Distutils の新世代版を開発するために Distutils2 パッケージ (source repository at http://hg.python.org/distutils2/) も開始しました。

10.3 ttk: Tk のテーマ付きウィジェット

Tcl/Tk 8.5 は、基本的な Tk ウィジェットを最実装しながらより自由な外観のカスタマイズが広がり、よりネイティブなプラットフォームのウィジェットに似たテーマ付きウィジェット群を含みます。このウィジェット群は、元は Tile と呼ばれていましたが、Tcl/Tck リリース 8.5 への追加の際に (“themed Tk” を略して) Ttk に改名されました。

詳しく知るには、`ttk` モジュールのドキュメントをお読みください。http://www.tcl.tk/man/tcl8.5/TkCmd/ttk_intro.htm にある、Ttk テーマエンジンを解説した Tcl/Tk マニュアルページを読むのもいいでしょう。Python/Ttk コードを動かしているスクリーンショットは http://code.google.com/p/python-ttk/wiki/Screenshots にあります。

`ttk` モジュールは Guilherme Polo によって書かれ、 issue 2983 に追加されました。 `Tile.py` と呼ばれる別バージョン `Tile.py` は、Martin Franklin によって書かれ、Kevin Walzer によってメンテナンスされているもので、組み込みが issue 2618 で提案されましたが、著者は Guilherme Polo の作品のほうがより包括的であると主張しました。

10.4 更新されたモジュール: unittest

`unittest` モジュールが大幅に改善されました。多くの新機能が追加されました。それらの機能のほとんどは、特に注釈のない限り、Michael Foord によって実装されました。このモジュールの改善された版は、Python バージョン 2.4 から 2.6 で使うために、http://pypi.python.org/pypi/unittest2 から `unittest2` パッケージとして別にダウンロードできます。

コマンドラインから使われるとき、このモジュールはテストを自動的に発見します。これは py.test や nose ほど派手ではありませんが、テストをパッケージディレクトリに保持されたテストを簡潔に実行する方法を提供します。例えば、以下のコマンドは `test/` サブディレクトリから `test*.py` という名前の import できるテストファイル検索します:

```
python -m unittest discover -s test
```

詳細は、 `unittest` モジュールのドキュメントを参照してください。(Developed in issue 6001.)

`main()` 関数は、その他幾つかの新しいオプションを提供します:

- *-b* や *--buffer* は、標準出力や標準エラーストリームをそれぞれのテストの間バッファに入れます。テストが通れば、結果の出力は全て捨てられます。失敗したら、バッファに入れられた出力が表示されます。

- *-c* や *--catch* は、control-C による中断をより上品にします。テストプロセスを即座に中断するのではなく、現在実行中のテストは完了させ、中断までの部分的な結果は報告されます。待ちきれなければ、control-C をもう一度押せば、即座に中断されます。

  この control-C ハンドラは、コードがテストされているときや、実行されているテストが独自のシグナルハンドラを定義したとき、シグナルハンドラがすでに設定されていてそして呼び出されたということを知らせることで、問題を起こさないようにします。これがまずいなら、`removeHandler()` デコレータを使って、テストが control-C ハンドリングを無効にするべきであると示せます。

- *-f* や *--failfast* は、テストが失敗したとき、他のテストを続けるのではなく、テストの実行を即座に停止します。(Suggested by Cliff Dyer and implemented by Michael Foord; issue 8074.)

進捗メッセージは、verbose モードで実行したとき、期待された失敗に 'x' を、期待されない成功に 'u' を表示するようになりました。(Contributed by Benjamin Peterson.)

テストケースは、テストをスキップするために `SkipTest` 例外を送出します。(issue 1034053).

`assertEqual()`, `assertTrue()`, および `assertFalse()` 失敗のメッセージは、提供する情報量が増えました。 `TestCase` クラスの `longMessage` 属性を True に設定すると、失敗時に、標準のエラーメッセージと追加して提供したメッセージの両方が表示されます。(Added by Michael Foord; issue 5663.)

`assertRaises()` メソッドは、実行する呼び出し可能オブジェクトを与えずに呼び出されたとき、コンテキストハンドラを返すようになりました。例えば、こう書くことができます:

```
with self.assertRaises(KeyError):
    {}['foo']
```

(Implemented by Antoine Pitrou; issue 4444.)

モジュールレベルおよびクラスレベルの設定と、ティアダウンフィクスチャがサポートされました。モジュールは、 `setUpModule()` および `tearDownModule()` 関数を含むことができます。クラスは `setUpClass()` および `tearDownClass()` メソッドを含むことができ、これらは (`@classmethod` や透過なものを使って) クラスメソッドとして定義しなければなりません。これらの関数とメソッドは、テストランナーが別のモジュールやクラスのテストケースに切り替えるときに呼び出されます。

メソッド `addCleanup()` および `doCleanups()` が追加されました。`addCleanup()` で、無条件に呼び出されるクリーンアップ関数を (`setUp()` が失敗したら `setUp()` の後に、そうでなければ `tearDown()` の後に) 追加できるようにします。これにより、テスト中のリソースの配置と開放が簡潔になります。(issue 5679).

より特化したテストを提供するメソッドがいくつか追加されました。これらのメソッドの多くは、Google のエンジニアたちによってそのテストスイートのために書かれました。Gregory P. Smith, Michael Foord, and GvR が、これを `unittest` の Python 版にマージしました。

- `assertIsNone()` および `assertIsNotNone()` はひとつの式を取り、その結果が `NONE` であるか、またはないかを確かめます。
- `assertIs()` および `assertIsNot()` は、2 つの値を取り、その評価が同一のオブジェクトであるか、またはないかを確かめます。(Added by Michael Foord; issue 2578.)
- `assertIsInstance()` および `assertNotIsInstance()` は、結果のオブジェクトが特定のクラス、またはタプルにあるクラスのいずれかのインスタンスであるかを調べます。(Added by Georg Brandl; issue 7031.)
- `assertGreater()`, `assertGreaterEqual()`, `assertLess()`, および `assertLessEqual()` は、二つの量を比較します。
- `assertMultiLineEqual()` は 2 つの文字列を比較し、等しくなければ、2 文字列の差分をハイライトする有益な比較を表示します。この比較は、Unicode 文字列が `assertEqual()` で比較されるときのデフォルトに使われるようになりました。
- `assertRegexpMatches()` および `assertNotRegexpMatches()` は、第一引数が第二引数として与えられた正規表現にマッチする、またはしない文字列であるかどうかを調べます。
- `assertRaisesRegexp()` は、特定の式が送出されるかを調べ、そしてまたその式の文字列表現が与えられた正規表現にマッチするかを調べます。
- `assertIn()` および `assertNotIn()` は、 *first* が *second* に属するか、または属さないかを調べます。
- `assertItemsEqual()` は、2 つの与えられたシーケンスが同じ要素を含むかを調べます。
- `assertSetEqual()` は、2 つの集合が等しいか比較し、エラーの場合のみ、集合間の差分を報告します。
- 同様に、 `assertListEqual()` および `assertTupleEqual()` は、指定された型を比較し、差分を説明しますが、完全な値を表示するとはかぎりません。これらのメソッドは、リストやタプルを `assertEqual()` で比較するときにデフォルトで使われるようになりました。より一般的には、`assertSequenceEqual()` は 2 つのシーケンスを比較し、必要なら両方のシーケンスが特定の型であるかを調べます。
- `assertDictEqual()` は、二つの辞書を比較し、差分を報告します。これは、辞書を `assertEqual()` で比較するときにデフォルトで使われるようになりました。

`assertDictContainsSubset()` は、*first* に属するキー/値の対の全てが *second* に現れるかを調べます。

- `assertAlmostEqual()` および `assertNotAlmostEqual()` は、*first* と *second* がほぼ等しいかを判定します。このメソッドは、オプションで指定された *places* (デフォルトは 7) の数に差を丸めてそれをゼロと比べるか、差が与えられた *delta* の値より小さいことを要求します。

- `loadTestsFromName()` は、`TestLoader` の `suiteClass` 属性を適切に受け入れます。(Fixed by Mark Roddy; issue 6866.)

- 新しいフックにより、 `assertEqual()` メソッドを拡張して新しいデータ型を扱わせられます。`addTypeEqualityFunc()` メソッドは型オブジェクトと関数を取ります。この関数は、比較される両方のオブジェクトが特定の型であるときに使われます。この関数は、2 つのオブジェクトを比較し、マッチしなければ例外を送出するべきです。この関数が、新しいシーケンス比較メソッドがするように、2 つのオブジェクトがなぜマッチしないのかについて追加の情報を提供するのがいいアイデアです。

`unittest.main()` はオプションの `exit` 引数を取るようになりました。False なら、 `main()` は `sys.exit()` を呼び出さず、これにより対話型インタプリタから `main()` が使えるようになります。(Contributed by J. Pablo Fern ndez; issue 3379.)

`TestResult` に、テストランの直前と直後に呼び出される `startTestRun()` および `stopTestRun()` が追加されました。(Contributed by Robert Collins; issue 5728.)

これら一連の変更により、`unittest.py` は無様に大きくなったので、モジュールはパッケージとなり、コードは複数のファイルに分割されました (by Benjamin Peterson)。これは、モジュールをインポートして使う方法には影響しません。

参考:

http://www.voidspace.org.uk/python/articles/unittest2.shtml Describes the new features, how to use them, and the rationale for various design decisions. (By Michael Foord.)

10.5 更新されたモジュール: ElementTree 1.3

Python に同梱される ElementTree ライブラリのバージョンが、バージョン 1.3 にアップデートされました。新機能の一部は:

- 様々な解析関数が、使われる `XMLParser` インスタンスを与える *parser* キーワード引数を取るようになりました。これにより、ファイルの内部エンコーディングをオーバーライドできるようになりました:

```
p = ET.XMLParser(encoding='utf-8')
t = ET.XML("""<root/>""", parser=p)
```

  XML を解析するときのエラーは、 `ParseError` 例外を送出するようになりました。この例外のインスタンスは、問題の位置を与える (*line*, *column*) タプルを含む `position` 属性を持ちます。

- ツリーを文字列に変換する ElementTree のコードが大幅に改善され、多くの場合でおおよそ 2 倍速く

なりました。 ElementTree `write()` および Element `write()` メソッドは、 *method* パラメタを追加し、これは “xml” (デフォルト), “html”, または “text” にできます。HTML モードは、空の要素を `<empty/>` ではなく `<empty></empty>` として出力し、text モードは要素を無視し、テキストのチャンクのみ出力します。要素の `tag` 属性を `None` に設定してその子はそのままにすると、ツリーが書き出されるとき、その要素は省かれるので、それ以上再編成することなく要素を一つ取り除けます。

名前空間の操作も改善されました。すべての `xmlns:<whatever>` 宣言は、結果の XML に散らばるのではなく、ルート要素に出力されます。`default_namespace` 属性を設定することでデフォルトの名前空間を設定でき、 `register_namespace()` で新しい接頭辞を登録できます。XML モードでは、XML 宣言を抑制するために真/偽の *xml_declaration* パラメタを使えます。

- 新しい `Element` メソッド: `extend()` はシーケンスから項目の子に要素を追加します。要素それ自体はシーケンスのように振る舞うので、子をある要素から別の要素に移すのが簡単です:

```
from xml.etree import ElementTree as ET

t = ET.XML("""<list>
  <item>1</item> <item>2</item>  <item>3</item>
</list>""")
new = ET.XML('<root/>')
new.extend(t)

# Outputs <root><item>1</item>...</root>
print ET.tostring(new)
```

- 新しい `Element` メソッド: `iter()` は、要素の子をジェネレータとして与えます。また、`for child in elem:` と書いて要素の子に渡ってループできます。既存のメソッド `getiterator()` と、子のリストを構成して返す `getchildren()` は、非推奨になりました。

- 新しい `Element` メソッド: `itertext()` は、その要素の子孫であるテキストのチャンクを全て与えます。例えば:

```
t = ET.XML("""<list>
  <item>1</item> <item>2</item>  <item>3</item>
</list>""")

# Outputs ['\n  ', '1', ' ', '2', '  ', '3', '\n']
print list(t.itertext())
```

- 非推奨: 要素をブール値として使う (すなわち、`if elem:`) と、要素が子を持てば真を、子を持たなければ偽を返します。この振る舞いは紛らわしいです – ですから、 `FutureWarning` を引き起こすようになりました。コードでは、明示するべきです。子の数に興味があるなら `len(elem) != 0` と、または `elem is not None` と書いてください。

Fredrik Lundh は ElementTree を開発し、1.3 バージョンを作成しました。1.3 を解説した彼の記事を http://effbot.org/zone/elementtree-13-intro.htm で読めます。Florent Xicluna は、python-dev および issue 6472 での議論の後、Python に含まれるバージョンをアップデートしました。

11 ビルドと C API の変更

パイソンのビルド過程と C API に以下のような変更がなされました:

- GNU デバッガ、GDB 7、の最新のリリースは Python を使って書けます実行可能なプログラム P のデバッグを始めるとき、GDB は `P-gdb.py` という名前のファイルをロックし、それを自動的に読み込みます。Dave Malcolm は Python 自体をデバッグするときに便利ないくつかのコマンドを加える `python-gdb.py` に貢献しました。例えば、 `py-up` および `py-down` は、通常いくつかの C スタックフレームに対応する、Python のスタックフレームを上がったり下がったりします。`py-print` は、Python 変数の値を表示し、 `py-bt` は、Python スタックトレースを表示します。(Added as a result of issue 8032.)

- Python に備え付けの `.gdbinit` ファイルを使うと、デバッグされているスレッドが GIL をほじしていないとき、2.7 バージョンの “pyo” マクロが正常に動くようになりました。このマクロは、表示の前にこれを取得するようになりました。(Contributed by Victor Stinner; issue 3632.)

- `Py_AddPendingCall()` はスレッドセーフになり、ワーカスレッドがメイン Python スレッドに通知を投入することができるようになりました。これは特に、非同時性の IO 処理に便利です。(Contributed by Kristj n Valur J nsson; issue 4293.)

- 新しい関数: `PyCode_NewEmpty()` は、空のコードオブジェクトを生成します。要求されるのは、ファイル名、関数名、そして最初の行番号だけです。これは、より便利なトレースバックスタックを構成しようとする拡張モジュールに便利です。以前は、このような拡張はより多くの引数を必要とする `PyCode_New()` を呼び出す必要がありました。(Added by Jeffrey Yasskin.)

- 新しい関数: `PyErr_NewExceptionWithDoc()` は、既存の `PyErr_NewException()` と同じように新しい例外クラスを生成しますが、新しい例外クラスの docstring を含む、追加の `char *` 引数を取ります。(Added by ‘lekma’ on the Python bug tracker; issue 7033.)

- 新しい関数: `PyFrame_GetLineNumber()` はフレームオブジェクトを取り、そのフレームが現在実行している行番号を返します。以前は、コードは現在実行しているバイトコード命令のインデクスを得て、それからそのアドレスに対応する行番号を探索する必要がありました。(Added by Jeffrey Yasskin.)

- 新しい関数: `PyLong_AsLongAndOverflow()` および `PyLong_AsLongLongAndOverflow()` は、C の `long` や `long long` のような Python の長整数に近いです。この数が出力型に適合するには大きすぎるなら、 *overflow* フラグが設定され、呼び出し元に返されます。(Contributed by Case Van Horsen; issue 7528 and issue 7767.)

- 新しい関数: 文字列から浮動小数点数への変換から由来する、新しい `PyOS_string_to_double()` 関数が追加されました。古い `PyOS_ascii_strtod()` および `PyOS_ascii_atof()` は、非推奨となりました。

- 新しい関数: `PySys_SetArgvEx()` は、 `sys.argv` の値を設定し、*updatepath* に依ってオプションで `sys.path` を更新して `sys.argv[0]` で指名されたスクリプトを含むディレクトリを含めることができます。

この関数は、Python を埋め込んだアプリケーションのセキュリティホールを塞ぐために追加されました。古い関数 `PySys_SetArgv()` は、必ず `sys.path` を更新し、これはカレントディレクトリを追加することがあります。これにより、Python を埋め込んだアプリケーションを別の誰かが制御するディレクトリで実行すると、攻撃者がディレクトリにトロイの木馬モジュール (例えば、 `os.py` という名前のファイル) を仕込み、インポートさせ実行させる事ができました。

Python を埋め込んだアプリケーションを保守しているなら、`PySys_SetArgv()` を呼び出していないか調べ、 *updatepath* を偽に設定した `PySys_SetArgvEx()` を使うべきかよく考えてください。

issue 5753 で CVE-2008-5983 として報告され、Antoine Pitrou によって修正されたたセキュリティ問題です。

- 新しいマクロ: Python ヘッダファイルは、以下のマクロを定義しました: `Py_ISALNUM`, `Py_ISALPHA`, `Py_ISDIGIT`, `Py_ISLOWER`, `Py_ISSPACE`, `Py_ISUPPER`, `Py_ISXDIGIT`, `Py_TOLOWER`, および `Py_TOUPPER`。これらすべての関数は、文字を分類する C の標準マクロと類似ですが、Python は文字をロケールに依らず文字を分析したいときがあるため、現在のロケール設定を無視します。(Added by Eric Smith; issue 5793.)

- 取り除かれた関数: `PyEval_CallObject` は、マクロとしてのみ利用できるようになりました。関数版は、ABI リンク互換性を保つために残されましたが、それは 1997 年のことです。今となっては削除するべきです。(Removed by Antoine Pitrou; issue 8276.)

- 新しいフォーマットコード: `PyFormat_FromString()`, `PyFormat_FromStringV()`, および `PyErr_Format()` 関数は、C の `long long` 型を表示する `%lld` および `%llu` フォーマットを受け付けるようになりました。(Contributed by Mark Dickinson; issue 7228.)

- スレッドとプロセスのフォークの間の複雑な相互関係が変更されました。以前は、`os.fork()` によって生成された子プロセスは失敗していました。これは、子は動作しているスレッド 1 つだけで生成され、そのスレッドが `os.fork()` を処理するからです。他のスレッドが、Python のインポートロックのような、ロックを保持していたら、フォークが実行されたとき、ロックは新しいプロセスでも “held” としてマークされたままです。しかし、子プロセスでは、他のスレッドは複製されないため、ロックを開放するものは何もないので、子プロセスは import の実行を続けることができません。

  Python 2.7 は、 `os.fork()` を実行する前に import ロックを取得し、また `threading` モジュールを使って生成された全てのロックをクリーンアップします。内部ロックを持っていたり、自身で `fork()` を呼び出したりするような C 拡張モジュールは、このクリーンアップの恩恵を受けません。

  (Fixed by Thomas Wouters; issue 1590864.)

- `Py_Finalize()` 関数は、内部の `threading._shutdown()` 関数を呼び出します。これにより、インタプリタがシャトダウンするときに送出が防がれる例外があります。(Patch by Adam Olsen; issue 1722344.)

- `PyMemberDef` 構造を使って型の属性を定義するとき、それ以上 `T_STRING_INPLACE` 属性を削除または設定できなくなります。

- `ctypes` によって定義されたグローバルシンボルには、 `Py` または`_ctypes` が接頭されるようになり

ました。(Implemented by Thomas Heller; issue 3102.)

- 新しい設定オプション: *--with-system-expat* スイッチにより、pyexpat モジュールをビルドして、システム Expat ライブラリを使えます。(Contributed by Arfrever Frehtes Taifersar Arahesis; issue 7609.)
- 新しい設定オプション: *--with-valgrind* オプションは、Valgrind メモリエラー検出器が正しく分析するのが難しい pymalloc アロケータを無効にするようになりました。ですから Valgrind は、メモリリークやオーバーランをより検出できます。(Contributed by James Henstridge; issue 2422.)
- 新しい設定オプション: *--with-dbmliborder=* に空の文字列を与えて様々な DBM モジュールを全て無効にできます。(Added by Arfrever Frehtes Taifersar Arahesis; issue 6491.)
- **configure** スクリプトは、ある種の 32-bit Intel チップ上の浮動小数点丸めバグを調べ、X87_DOUBLE_ROUNDING プリプロセッサ定義を定義します。現在この定義を使うコードはありませんが、使うことを望む人がいたら使えます。(Added by Mark Dickinson; issue 2937.)

  configure はまた、C++ リンクをサポートする LDCXXSHARED Makefile を設定するようになりました。(Contributed by Arfrever Frehtes Taifersar Arahesis; issue 1222585.)
- ビルドプロセスは、pkg-config サポートに必要なファイルを生成するようになりました。(Contributed by Clinton Roy; issue 3585.)
- ビルドプロセスは、Subversion 1.7 をサポートするようになりました。(Contributed by Arfrever Frehtes Taifersar Arahesis; issue 6094.)

11.1 カプセル

Python 3.1 は、拡張モジュールに C API を提供する新しい C データ型 PyCapsule を追加しました。カプセルは、本質的には C の void * ポインタのホルダであり、モジュール属性として利用できるようになります。例えば、 socket モジュールの API は socket.CAPI として公開されていて、 unicodedata は ucnhash_CAPI を公開しています。その他の拡張はモジュールを import し、辞書にアクセスしてカプセルオブジェクトを取得し、そしてモジュールの様々な API 関数へのポインタの配列を指す void * ポインタを取得できます。

これに使われる既存のデータ型 PyCObject がありますが、これは型安全性を提供しません。pure Python で書かれた邪悪なコードは、PyCObject をモジュール A から取り、どうにかしてそれをモジュール B の PyCObject で代えることで、セグメンテーション違反を起こし得ます。カプセルは自身の名前を知っていて、ポインタを得るには名前を提供しなければなりません:

```
void *vtable;

if (!PyCapsule_IsValid(capsule, "mymodule.CAPI") {
        PyErr_SetString(PyExc_ValueError, "argument type invalid");
        return NULL;
}
```

```
vtable = PyCapsule_GetPointer(capsule, "mymodule.CAPI");
```

vtable が期待どおりのものを指すことが保証されます。異なるカプセルが渡されたら、PyCapsule_IsValid() は不適合な名前を検知し、偽を返します。これらのオブジェクトの使用について、より詳しい情報は、 *using-capsules* を参照してください。

Python 2.7 は、様々な拡張モジュール API を提供するためにカプセルを内部的に使うようになりましたが、カプセルを処理するために PyCObject_AsVoidPtr() が変更され、 CObject インタフェースとのコンパイル時互換性は保たれます。 PyCObject_AsVoidPtr() を使うと PendingDeprecationWarning が合図されますが、デフォルトでは言及されません。

Implemented in Python 3.1 and backported to 2.7 by Larry Hastings; discussed in issue 5630.

11.2 ポート特有の変更: Windows

- msvcrt モジュールは、 crtassem.h からのいくつかの定数を追加しました 。CRT_ASSEMBLY_VERSION, VC_ASSEMBLY_PUBLICKEYTOKEN, および LIBRARIES_ASSEMBLY_NAME_PREFIX. (Contributed by David Cournapeau; issue 4365.)

- レジストリへのアクセスのための _winreg モジュールは、以前サポートされていた関数群の引数を増やした拡張版である CreateKeyEx() および DeleteKeyEx() を実装しました。DisableReflectionKey(), EnableReflectionKey(), および QueryReflectionKey() もテストされ、ドキュメント化されました。(Implemented by Brian Curtin: issue 7347.)

- 新しい _beginthreadex() API がスレッドの開始に使われ、ネイティブのスレッドローカルストレージ関数が使われるようになりました。(Contributed by Kristj n Valur J nsson; issue 3582.)

- os.kill() 関数が Windows で働くようになりました。このシグナル値は定数 CTRL_C_EVENT, CTRL_BREAK_EVENT, その他の整数にできます。最初の 2 定数は Control-C および Control-Break 打鍵イベントをサブプロセスに送ります。その他の値は、 TerminateProcess() API を使います。(Contributed by Miki Tebeka; issue 1220212.)

- os.listdir() 関数は、空のパスに対してちゃんと失敗するようになりました。(Fixed by Hirokazu Yamamoto; issue 5913.)

- mimelib モジュールは、初期化の際に Windows レジストリから MIME データベースを正しく読みこむようになりました。(Patch by Gabriel Genellina; issue 4969.)

11.3 ポート特有の変更: Mac OS X

- 追加されたパッケージを、システムインストレーションとユーザがインストールした同じバージョンのコピーで共有するために、 sys.path に/Library/Python/2.7/site-packages が追加されました。(Changed by Ronald Oussoren; issue 4865.)

11.4 ポート特有の変更: FreeBSD

- FreeBSD 7.1 の、他のルーティングテーブルを選択するための `getsockopt()`/`setsockopt()` で使われている `SO_SETFIB` 定数が、 `socket` モジュールで使えるようになりました。(Added by Kyle VanderBeek; issue 8235.)

12 その他の変更と修正

- ベンチマークスクリプトの 2 つ、 `iobench` および `ccbench`, が `Tools` ディレクトリに追加されました。 `iobench` は、様々な演算を実行している間 `open()` によって返された組み込みファイル I/O オブジェクトの速度を計測し、 `ccbench` は、変化する数のスレッドを使っていくつかのタスクを処理するときの計算スループット、スレッド切り替えレイテンシ、IO 処理バンド幅の計測を試みる同時実行ベンチマークです。

- `Tools/i18n/msgfmt.py` スクリプトは、複数の形式の `.po` ファイルを理解できるようになりました。(Fixed by Martin von L wis; issue 5464.)

- 既存の対応する `.py` が存在する `.pyc` または `.pyo` ファイルからモジュールを import するとき、結果のコードオブジェクトの `co_filename` 属性は、元のファイル名が廃止されたとき、上書きされます。これは、ファイルがリネーム、移動、異なるパスを通したアクセスをされたときに起こりえます。(Patch by Ziga Seilnacht and Jean-Paul Calderone; issue 1180193.)

- `regrtest.py` スクリプトは、テストを乱順に実行する *`-r`* オプションの、ランダムシードに使われる整数を取る *`--randseed=`* スイッチを取るようになりました。 *`-r`* オプションは、使われたシードも報告します。(Added by Collin Winter.)

- もう一つの `regrtest.py` スイッチは *`-j`* で、いくつのテストを並行して実行するかを指定する整数です。これにより、マルチコア機での合計実行時間を短縮できます。このオプションは、実行時間を長くすることで知られている *`-R`* スイッチなど、いくつかの他のオプションと互換です。(Added by Antoine Pitrou, issue 6152.) これはまた、選ばれたテストを失敗するまでループして実行する *`-F`* スイッチとも同時に使えます。(Added by Antoine Pitrou; issue 7312.)

- スクリプトとして実行されるとき、 `py_compile.py` モジュールは標準入力を読み込んでコンパイルするファイル名のリストとする `'-'` を引数として受け付けるようになりました。(Contributed by Piotr O arowski; issue 8233.)

13 Python 2.7 への移植

この説では、先に説明した変更その他のバグ修正の中で、コードの変更を必要とするかもしれないものを列挙します:

- `range()` 関数は、引数をより一貫して処理するようになりました。浮動小数点数でもなく整数でもない引数が与えられたら、その `__int__()` を呼び出すようになりました。(Fixed by Alexander

Belopolsky; issue 1533.)

- 文字列 `format()` メソッドは浮動小数点数および複素数に使われるデフォルトの精度を、 `str()` で使われる精度に合わせ、小数第 6 位から 12 位に変更しました。(Changed by Eric Smith; issue 5920.)

- `with` 文の最適化のため、特殊メソッド `__enter__()` および `__exit__()` はオブジェクトの型に属さなければならず、オブジェクトのインスタンスには直接取り付けられません。これは (`object` から導出された) 新スタイルクラスと C 拡張型に影響します。(issue 6101.)

- Python 2.6 におけるバグのため、 `__exit__()` の *exc_value* パラメタははよく、例外のインスタンスではなく文字列表現になっていました。これは 2.7 では修正され、 *exc_value* は期待通りインスタンスになります。(Fixed by Florent Xicluna; issue 7853.)

- `__slots__` を使って属性の制限が設定されたとき、設定されていない属性を削除しても `AttributeError` が送出されないように修正されました。(Fixed by Benjamin Peterson; issue 7604.)

標準ライブラリでは:

- 年がサポートされている範囲外にはみ出るような `datetime` インスタンスの演算が、`OverflowError` を送出しないことがありました。このようなエラーはより注意深く調べられ、例外を送出するようになりました。(Reported by Mark Leander, patch by Anand B. Pillai and Alexander Belopolsky; issue 7150.)

- `Decimal` インスタンスを文字列の `format()` メソッドに使うとき、デフォルトの揃えは以前は左揃えでした。これは右揃えに変更され、プログラムの出力が変わるかもしれません。(Changed by Mark Dickinson; issue 6857.)

  signaling NaN 値 (または `sNAN`) を含む比較は、比較演算子に依って静かに真や偽となるのではなく、`InvalidOperation` を合図するようになりました。quiet NaN 値 (または `NaN`) はハッシュ可能になりました。(Fixed by Mark Dickinson; issue 7279.)

- ElementTree ライブラリ `xml.etree` は、(<*?xml-stylesheet href="#style1"?*> のような) 命令や (<*!–comment –*> のような) コメントを処理する XML を出力するときにアンパサンドおよび角括弧をエスケープしなくなりました。(Patch by Neil Muller; issue 2746.)

- `StringIO` オブジェクトの `readline()` メソッドは、ファイル風のオブジェクトのように、負の長さが要求されたとき、何もしなくなりました。(issue 7348)

- `syslog` モジュールは、識別子として以前のデフォルト値である `'python'` ではなく、`sys.argv[0]` の値を使うようになりました。(Changed by Sean Reifschneider; issue 8451.)

- `tarfile` モジュールのデフォルトエラー処理が変更され、致命的なエラーを抑制しないようになりました。デフォルトのエラーレベルは以前は 0 で、エラーはメッセージとしてデバッグログに書き込まれるだけでしたが、デバッグログはデフォルトでは活性化されていないため、エラーは顧みられませんでした。デフォルトのエラーレベルは 1 になり、エラーがあれば例外が送出されます。(Changed by Lars Gust bel; issue 7357.)

- `urlparse` モジュールの `urlsplit()` は、未知の URL スキームを **RFC 3986** に応じた方法で処理

します。URL が "<something>://..." の形式なら、 :// の前のテキストを、それがモジュールの知らない作り物のスキームであってさえ、スキームとして扱います。この変更は、古い働きを使って動作していたコードを破壊することがあります。例えば、Python 2.6.4 や 2.5 は以下を返します:

```
>>> import urlparse
>>> urlparse.urlsplit('invented://host/filename?query')
('invented', '', '//host/filename?query', '', '')
```

Python 2.7 (や Python 2.6.5) は以下を返します:

```
>>> import urlparse
>>> urlparse.urlsplit('invented://host/filename?query')
('invented', 'host', '/filename?query', '', '')
```

(Python 2.7 では、これは普通のタプルではなく名前付きタプルを返すので、実際は微妙に異なる出力をします。)

C 拡張では:

- PyArg_Parse* 系統の関数で整数フォーマットコードを使う C 拡張は、DeprecationWarning を引き起こすのではなく、 TypeError 例外を送出するようになりました。(issue 5080)
- 非推奨となった古い PyOS_ascii_strtod() および PyOS_ascii_atof() 関数の代わりに、新しい PyOS_string_to_double() 関数を使ってください。

Python を埋め込んだアプリケーションでは:

- PySys_SetArgvEx() 関数が追加され、既存の PySys_SetArgv() 関数が使われた時のセキュリティホールをを塞ぐことができるようになりました。PySys_SetArgv() を呼び出していないか調べ、 *updatepath* を偽に設定した PySys_SetArgvEx() を使うべきかよく考えてください。

14 謝辞

著者は、以下の方々がこの記事の多くの草稿に提案、修正、そして助力を差し伸べてくださったことに感謝の意を表したいと思います: Nick Coghlan, Philip Jenvey, Ryan Lovett, R. David Murray, Hugh Secker-Walker.

索引