What's New in Python 3.1

Some Notes

27th June 2009

New features

- PEP372: Ordered Dictionaries (collections.OrderedDict)
- collections.Counter - count unique items in a sequence
- (7).bit_length()
- Support for multiple context managers when using *with* statements
  - contextlib.nested deprecated
- itertools gains combinations_with_replacement, compress
  - itertools.count now has optional step parameter
  - itertools.count takes any type supporting PyNumber_Add, including Fractions and Decimals
- logging has a NullHandler for discarding logged messages
- importlib - pure python implementation of import and __import__
- unittest supports skipping tests
  - also allows tests to be marked as expected to fail

General improvements

- io module replaced with version written in C - greatly improves performance from 3.0
  - python io module still in standard library if needed
- "–with-computed-gotos" configure option - uses tweaked dispatch mechanism for compilers which support computed gotos - 20% faster on some benchmarks
- PEP383: Non-decodable Bytes in System Character Interfaces
- collections.namedtuple supports
- Significant speedups to UTF-8, UTF-16 & LATIN-1 decoding
- C extension added json module - significantly faster
  - json module no longer supports bytes - closer match to json specification
- format improvements
  - PEP378: Format specifier for thousands separator
  - Automatic numbering of fields

- round no longer silently converts integers to floats
- repr(float) uses shortest available representation
- Decimal's can be created from floats
  - differs from repr - aims to be exact decimal representation of actual floating point number
- gzip.GzipFile, bz2.BZ2File support context manager protocol
- regex flags can be passed to more re functions - re.sub, re.split, re.subn
- pdb now supports zipimport
- better support for creating pickles for python 2 (pickle protocol 2)
  - breaks interaction with python 3.0, but that should be using pickle protocol 3 anyway

Deprecated features

- PyCObject deprecated in favour of new PyCapsule
  - Adds additional type information - used to avoid avoid issues with passing a object from one module to another.
  - Destruct or takes object as a PyObject, rather than PyCObject's (void*) or (void*, void*) logic
- string.maketrans() deprecated
  - bytes, bytearry & str classes all have suitable maketrans methods now

PEP 372: Order Dictionaries

- Preserves insertion order.
- Intended for use in json, ElementTree, etc.
- Standard library now uses OrderedDict by default in ConfigParser & collections.namedtuple._asdict()
- True subclass of dict
- Nifty implementation - retains the O(1) amortised cost for dict operations
- Isn't a true sequence, but support reversed

PEP 383: The Problem

- File system names on POSIX system are byte-strings
  - encoding of the filename isn't specified anywhere, taken from environment
  - Entirely possible to have mixed encodings on the file system
  - All OS fs operations just use bytes, encoding and decoding sensibly is a application problem
- Windows (NT onwards)
  - NTFS names use UCS-2 sequences
    - it's possible to have non-Unicode sequences, though those meant to be non-trivial to create through the standard API's
  - VFAT can mix UCS-2 and byte-string encoding - windows API will do the guessing and conversion "somehow"
  - Windows doesn't have a good byte-level API
- MACOS has a couple of extra wrinkles as HFS+ restricts names to UTF-8, NFD normalised, strings

PEP383: So What?

- In py3k, default os operations (os.listdir, etc) return unicode strings (although byte-strings can be asked for)
- In v3.0, On POSIX, os.listdir(".") will throw errors if not all names can be converted, while os.listdir(b".") will work
- On windows, os.listdir(".") will return all the filenames, but os.listdir(b".") won't
- Using os.listdir(b".") and encoding to unicode via standard error handlers isn't reversible (and may not be unique)

PEP383: surrogateescape, the error handler formally known as "UTF8b"

- The sequence <valid unicode> DCxx CANNOT occur in unicode strings
  - DCxx DCxx is also obviously illegal
- Problematic strings are byte strings which cannot be decoded to unicode - escape the sequences as DC + problematic byte for all bytes.
- Reversible - illegal sequence DC + xx should be converted back to byte xx.
- Null operation for valid unicode, so only arises when encountering undecodable strings (wrong fsencoding, multiple encodings in a single dir, etc).
- Implemented as the "surrogateescape" error handler

PEP383: Notes

- For most setups, this should never be triggered - only an issue when fsencoding is wrong for the file name in question
- Does allow converting arbitrary byte string input to required form by explicitly using the error handler
- Security issues - Under this scheme, DC2F => "/", etc.
  - Sanitising input a lot harder if this is allowed.
  - Avoided by not allowing ASCII characters to be escaped
  - Scheme thus cannot be used on data from several non ISO encodings - SHIFT JIS variants being the most prominent.
- Can test if a string can be passed through strict encoding to see if escaped characters are present.
- Can't avoid the use of this when calling os functions with unicode strings.

PEP383: Benefits

- Arbitrary fs names can be read into python strings, although the strings are no longer valid unicode
- Round tripping works - encoding the invalid unicode to the SAME fs encoding gives the same byte sequence
- Trans-coding - All bets are off. No magic solution for converting invalid encodings to valid encodings
- Doesn't eliminate byte interfaces - only aims to ensure certain classes of application only need to worry about about dealing with unicode strings, and thus avoid portability issues