pathlib

Elegant filesystem interactions in Python

Johan Herland
@jherland
<jherland@cisco.com>
<johan@herland.net>

Prerequisites

- Basic Python familiarity:
- **pathlib** included (provisionally) since Python v3.4
  - Post-v3.4 features are noted
  - Available as a separate module for Python v2.6 → v3.3: https://pypi.python.org/pypi/pathlib/

Outline

- What?
- Why?
- Example
- API, a quick overview
- Issues

What?

What?

- Pathnames in Python are traditionally *strings*
- **pathlib** represents filesystem paths as proper objects
    - Immutable/value objects
    - Handle Unix/Windows differences transparently
        - Write simple code that works everywhere!
    - Rich API for querying and manipulating paths
    - Replaces/supersedes:
        - Most things from `os.path.*`
        - Many things from `os.*`
        - `fnmatch.fnmatch()`
        - `glob.glob()`
        - Maybe even `open()`?

What?

- What is it **not**?
    - Only handles *paths*, not file objects
        - Don't care what you do to a file after you have opened it
    - Does not handle URIs or other *path-like* things

Why?

Why?

- Easier and safer handling of pathnames
- More readable code
    - Less `os.path.*` noise in your code
    - Moves your path names (variables or constants) towards the *left* (i.e. into focus):

```python
# Before
if os.path.isdir(path):
    os.rmdir(path)

# After:
if path.is_dir():
    path.rmdir()
```

Why?

Typical scenario:

- Python often used in build/packaging system
- "Glue" code

- Lots of path manipulation:
  - Creating/removing directories
  - Moving files around
  - Looking for specific (types of) files
  - Constructing/manipulating *relative* and *absolute* paths

Example

Example

```
import os

outpath = os.path.join(os.getcwd(), 'my_output')
outpath_tmp = os.path.join(os.getcwd(), 'my_output.tmp')

maybe_generate_data(outpath_tmp)

if os.path.getsize(outpath_tmp):
    os.rename(outpath_tmp, outpath)
else:  # Nothing produced
    os.remove(outpath_tmp)
```

becomes

```
from pathlib import Path

outpath = Path.cwd() / 'my_output'
outpath_tmp = Path.cwd() / 'my_output.tmp'

maybe_generate_data(outpath_tmp)


if outpath_tmp.p.stat().st_size:
    outpath_tmp.rename(outpath)
else:  # Nothing produced
    outpath_tmp.unlink()
```

API

a quick overview

API

Creating a path object

```
# string → path object
devnull = Path('/dev/null')

# class methods
current_dir = Path.cwd()
home_dir = Path.home()  # (since Python v3.5)
```

API

Joining paths

The / operator replaces `os.path.join()`:

```
doc_dir = Path('documents')
doc_file = Path('my_doc.pdf')

# path / path
my_doc = doc_dir / doc_file

# path / str
my_doc = doc_dir / 'my_doc.pdf'

# str / path
my_doc = 'documents' / doc_file

# NOT: str / str
my_doc = 'documents' / 'my_doc.pdf'
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Returns a *new* path object (remember: immutable/value objects)

API

Simple attributes

```
>>> my_doc = Path.cwd() / 'documents' / 'my_doc.pdf'
PosixPath('/home/jherland/documents/my_doc.pdf')

>>> my_doc.parts
('/', 'home', 'jherland', 'documents', 'my_doc.pdf')

>>> list(my_doc.parents)
[PosixPath('/home/jherland/documents'),
 PosixPath('/home/jherland'),
 PosixPath('/home'),
 PosixPath('/')]

>>> my_doc.parent  # ← os.path.dirname()
PosixPath('/home/jherland/documents')

>>> my_doc.name  # ← os.path.basename()
'my_doc.pdf'

>>> my_doc.suffix  # also: .suffixes and .stem

'.pdf'
```

API

Absolute and relative paths

- Relative → absolute: `.resolve()`
- Absolute → relative: `.relative_to(other_dir)`

```
>>> Path.cwd()
PosixPath('/home/jherland')

>>> docs = Path('documents')
PosixPath('documents')
>>> docs.is_absolute()  # ← os.path.isabs()
False

>>> docs_abs = docs.resolve()  # ← os.path.abspath()
PosixPath('/home/jherland/documents')
>>> docs_abs.is_absolute()
True

>>> docs_abs.relative_to('/home')  # ← os.path.relpath()
PosixPath('jherland/documents')
```

API

More `os.path.*` replacements

```
p.exists()  # ← os.path.exists()

p.is_dir()  # ← os.path.isdir()
p.is_file()  # ← os.path.isfile()
p.is_symlink()  # ← os.path.islink()

p.expanduser()  # ← os.path.expanduser() (since Python v3.5)
p.samefile(other_path)  # ← os.path.samefile() (since Python v3.5)
```

API

More `os.*` replacements

```
p.chmod(mode)  # ← os.chmod()

p.iterdir()  # ← os.listdir()

p.mkdir(...)  # ← os.mkdir() and os.makedirs()

p.rename(target)  # ← os.rename()

p.rmdir()  # ← os.rmdir()

p.stat()  # ← os.stat()

p.symlink_to(target)  # ← os.symlink()

p.unlink()  # ← os.unlink()
```

API

Other replacements

```
p.match(pattern)  # ← fnmatch.fnmatch()

p.glob(pattern)  # ← glob.glob() (rooted at p)

p.open(...)  # ← open()

# For example:
with in_path.open() as inf:
    with out_path.open('w') as outf:
        for line in inf:
            outf.write(process(line))
```

API

Even more convenience

```python
# Read/write (short) file contents in a single method call
p.read_bytes()
p.read_text(encoding=None, errors=None)
p.write_bytes(data)
p.write_text(data, encoding=None, errors=None)
```

Note: Only since Python v3.5

Issues

Issues

Still included on a *provisional basis*. There may be backwards-incompatible changes.

Missing integration with other stdlib modules and functions.

Path-related APIs use strings rather than `Path` objects:

```
open()
shutil.*  # .copy(), .rmtree(), etc.
subprocess.*  # argv[0], executable=, or cwd=
codecs.open() # (et al)
io.*
os.path.*
os.*
tempfile.*
shelve.*
csv.*
...
```

Issues

Techniques for combining pathlib and legacy code:

```
# Passing paths to an API that expects a string:
shutil.rmtree(str(path_object))

# Or (since Python v3.4.5 and Python v3.5.2):
shutil.rmtree(path_object.path)

# Converting Path-or-string into Path:
path = Path(path_or_string)

# Converting Path-or-string into string:
path_str = getattr(path_or_string, 'path', path_or_string)
```

Issues

Stuff missing from pathlib?

- Replacement for `os.walk()`?
- Replacement/support for `os.scandir()`? (new in Python v3.5)
    - Integration with its `DirEntry` objects to cache stat info is not trivial

(That said, it's fairly straightforward to write your own `Path`-aware wrapper around the above)

- Niggle: Appending a suffix to a path object (`pathstr += '.foo'`):
    - **FAILS**: `pathobj += '.foo'`
    - **UGLY?**: `pathobj = Path(str(pathobj) + '.foo')`
    - **UGLY?**: `pathobj = pathobj.with_suffix(pathobj.suffix + '.foo')`
    - **UGLY?**: `pathobj = pathobj.parent / (pathobj.name + '.foo')`
- More?

Epilogue

More details

- Docs: https://docs.python.org/3/library/pathlib.html
- History: https://www.python.org/dev/peps/pep-0428/
- For Python < v3.4: https://pypi.python.org/pypi/pathlib/

Thanks!

Questions?

Johan Herland
@jherland
<jherland@cisco.com>
<johan@herland.net>

Slideshow created with remark.