OstrichLib Documentation

Release 0.0.0

Itamar Ostricher

Apr 28, 2016

Contents

Mostly, just a bunch of useful Python functions.

Come visit an Ostrich on GitHub!

CHAPTER 1

utils package

1.1 collections utils module

collections utils module

A bunch of collections-related utility functions. Hazza!

ostrich.utils.collections.**listify**(*args*)

Return args as a list.

If already a list - return as is.

```
>>> listify([1, 2, 3])
[1, 2, 3]
```

If a set - return as a list.

```
>>> listify(set([1, 2, 3]))
[1, 2, 3]
```

If a tuple - return as a list.

```
>>> listify(tuple([1, 2, 3]))
[1, 2, 3]
```

If a generator (also range / xrange) - return as a list.

```
>>> listify(x + 1 for x in range(3))
[1, 2, 3]
>>> from past.builtins import xrange
>>> from builtins import range
>>> listify(xrange(1, 4))
[1, 2, 3]
>>> listify(range(1, 4))
[1, 2, 3]
```

If a single instance of something that isn't any of the above - put as a single element of the returned list.

```
>>> listify(1)
[1]
```

If "empty" (None or False or '' or anything else that evaluates to False), return an empty list ([]).

```
>>> listify(None)
[]
```

```
>>> listify(False)
[]
>>> listify('')
[]
>>> listify(0)
[]
>>> listify([])
[]
```

1.2 path utils module

path utils module

ostrich.utils.path.**commonpath**(*paths*)

Return the longest common sub-path of each pathname in paths sequence.

Raise ValueError if paths contains both absolute and relative pathnames, or if paths is empty.

Unlike `os.path.commonprefix()`, this returns a valid path:

```
>>> print(commonpath(['foo/bar', 'foo/baz', 'foo/baaam']))
foo
>>> from os.path import commonprefix
>>> print(commonprefix(['foo/bar', 'foo/baz', 'foo/baaam']))
foo/ba
```

Adapted from the source code of Python 3.5.1.

1.3 proc utils module

proc utils module

An adaptation of Python 3.5 subprocess.run function

source: https://github.com/python/cpython/blob/3.5/Lib/subprocess.py

exception ostrich.utils.proc.**CalledProcessError**(*returncode*, *cmd*, *output=None*, *stderr=None*)

This exception is raised when a process run by run() with check=True returns a non-zero exit status.

The exit status will be stored in the returncode attribute; The cmd (run args) will be stored in the cmd attribute; The output will be stored in output / stdout attribute; The stderr will be stored in stderr attribute.

stdout

Alias for output attribute, to match stderr

class ostrich.utils.proc.**CompletedProcess**(*args*, *returncode*, *stdout=None*, *stderr=None*)

A process that has finished running.

This is returned by run().

Attributes:

- args: The list or str args passed to run().
- returncode: The exit code of the process, negative for signals.
- stdout: The standard output (None if not captured).

- stderr: The standard error (None if not captured).

check_returncode()
Raise CalledProcessError if the exit code is non-zero.

ostrich.utils.proc.**run**(**popenargs*, ***kwargs*)
Run command with arguments and return a *CompletedProcess* instance.

The returned instance will have attributes args, returncode, stdout and stderr.

By default, stdout and stderr are not captured, and those attributes will be None. Pass stdout=PIPE and/or stderr=PIPE in order to capture them.

If *check* is True and the exit code was non-zero, it raises a CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute, and output & stderr attributes if those streams were captured.

If *timeout* is given, and the process takes too long, a TimeoutExpired exception will be raised, if timeout is supported in the underlying Popen implementation (e.g. Python >= 3.2, or an available subprocess32 package).

There is an optional argument *input*, allowing you to pass a string to the subprocess's stdin. If you use this argument you may not also use the Popen constructor's *stdin* argument, as it will be used internally.

The other arguments are the same as for the Popen constructor.

If universal_newlines=True is passed, the *input* argument must be a string and stdout/stderr in the returned object will be strings rather than bytes.

1.4 text utils module

text utils module

A collection of text-related utility functions. Hurray!

ostrich.utils.text.**as_text**(*str_or_bytes*, *encoding=u'utf-8'*, *errors=u'strict'*)
Return input string as a text string.

Should work for input string that's unicode or bytes, given proper encoding.

```
>>> print(as_text(b'foo'))
foo
>>> b'foo'.decode('utf-8') == u'foo'
True
```

ostrich.utils.text.**get_safe_path**(*in_str*)

Return *in_str* converted to a string that can be be safely used as a path (either filename, or directory name).

```
>>> print(get_safe_path("hello world, what's up?.txt"))
hello_world__what_s_up_.txt
```

Surrounding spaces are removed, and other "bad characters" are replaced with an underscore. The function attempts to replace unicode symbols (outside ASCII range) with ASCII equivalents (NFKD normalization). Everything else is also replaced with underscore.

Warning The function just returns a safe string to be used as a path. It DOES NOT promise anything about the existence or uniqueness of the path in the filesystem! Because of the conversions, many input strings will result the same output string, so it is the responsibility of the caller to decide how to handle this!

```
>>> get_safe_path(' foo/bar.baz') == get_safe_path('foo$bar.baz ')
True
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

O

A

C

G

L

O

R

S