Which path module or class do Python folks use instead of os.path?

Asked 15 years, 4 months ago Modified 13 years, 8 months ago Viewed 6k times

9

Just wondering how many people use a path module in Python such as Jason Orendorff's one, instead of using `os.path` for joining and splitting paths? Have you used:

- Jason's path module (updated for PEP 355)
- Mike Orr's Unipath, basically a more modern version of the above
- Noam Raphael's alternative path module that subclasses tuple instead of str

I know Jason's path module was made into PEP 355 and rejected by the BDFL. This seems like it was mainly because it tried to do everything in one class.

Our use case is mainly to simplify joining and splitting components of paths, so we'd be quite happy if such a path class only implemented the split/join type of operations. Who wouldn't want to do this:

```
path(build_dir, path(source_file).name)
```

or this:

```
build_dir / path(source_file).name
```

instead of this:

```
os.path.join(build_dir, os.path.basename(source_file))
```

python path

Share

Improve this question

Follow

edited Aug 10, 2009 at 0:15

asked Aug 10, 2009 at 0:03

Ben Hoyt

11k ● 6 ● 64 ● 87

looks like Python 3 has pathlib and there's a backport for Python 2. – Jason S Apr 25, 2017 at 17:44

3 Answers

Sorted by: Highest score (default)

11

I can pick up a Python program and interpret the current standard method without hesitation - it's explicit and there's no ambiguity:

```
os.path.join(build_dir, os.path.basename(source_file))
```

Python's dynamic typing makes the first method rather difficult to comprehend when reading:

```
build_dir / path(source_file).name
```

Plus it's not common to divide strings, which brings up more confusion. How do I know that those two aren't integers? Or floats? You won't get a TypeError at runtime if both end up as non-string types.

Finally,

```
path(build_dir, path(source_file).name)
```

How is that any better than the os.path method?

While they may "simplify" coding (ie, make it easier to write), you're going to run into strife if someone else who is unfamiliar with the alternative modules needs to maintain the code.

So I guess my answer is: I don't use an alternative path module. os.path has everything I need already, and it's interface isn't half bad.

Share Improve this answer Follow

answered Aug 10, 2009 at 0:10

Matthew Iselin
10.7k ● 4 ● 53 ● 62

2 Hear hear! There's nothing so horribly wrong with the standard os.path module to warrant adding more dependencies to your project. If you have a particularly hairy path construction problem, like constructing a path out of an object hierarchy, then why not wrap that in a function? The next programmer will thank you for encapsulating it, and for not making him learn and debug a whole other module. – Theran Aug 10, 2009 at 0:16

Thanks for the answer, Matthew. And good comment, Theran: you're right about factorizing things into functions -- this has simplified my script already. In fact, I think the two of you have convinced us. Plus, I guess there's always "from os import path" if we just want a bit more brevity. – Ben Hoyt Aug 10, 2009 at 2:41

No ambiguity, correct, but it's a pain to write and a pain to read. I'd rather deal with an immutable Path object like Java's `java.nio.file.Path` than a String. Example: `with build_dir.child('config.yaml').open() as f` vs `with open(os.path.join(build_dir, 'config.yaml')) as f` -- the logical flow is more consistently left-to-right, and with fewer nested parentheses, so easier to read and understand. – Jason S Apr 25, 2017 at 17:32

2

A simple but useful trick is this:

> import os
>
> Path = os.path.join

Then, instead of this:

> os.path.join(build_dir, os.path.basename(source_file))

You can do this:

> Path(build_dir, Path(source_file))

Share Improve this answer Follow

answered Apr 26, 2011 at 15:44

Steve Ferg
21 ● 1

1 I do this all the time, but I use `pj` instead of `Path`. :-) – docwhat Mar 2, 2012 at 6:20

Would you care to explain why you can replace `os.path.basename` by another instance of `Path = os.path.join`? Because for me, `os.path.join(r"C:\Build", os.path.basename(r"D:\Source\Code.py"))` returns `'C:\\Build\\Code.py'`, while `Path(r"C:\Build", Path(r"D:\Source\Code.py"))` returns `'D:\\Source\\Code.py'`. – bers Mar 12, 2019 at 9:29

-1

Dividing strings to join paths may seem like a "neat trick" but it's precisely that kind of thing that Python programmers like to avoid (and btw, programmers in most other langauges.) The os.path module is widely used and easily understood by all. Doing funky things with overloaded operators on the other hand is confusing, it impairs the readability of your code, which is meant to be one of Python's strong points.

C++ programmers, on the other hand, love that kind of thing. Perhaps that's one of the reasons C++ code can be so difficult to read.

Share Improve this answer Follow

answered Aug 10, 2009 at 3:17

Eloff
21.6k ● 18 ● 89 ● 120