seven Documentation

Release 1.0.0

Attila Oláh

Sep 27, 2017

Contents

`seven` is a Python 2.6+ compatibility layer for Python 2.5. It allows you to import Python 2.6 or 2.7 code in a 2.5 interpreter.

It was mainly intended to be used on Google App Engine, before the guys at Google added Python 2.7 support.

The project development is now discontinued, but you are welcome to use it for whatever you might need it.

It is licensed under the MIT licence, except for the lib2to3 code it contains, which in made available under the PSF licence. Those files contain individual licence information, so check the source code for more info.

CHAPTER 1

Basic usage

Before importing non-compatible code, you need to start the `seven` import hook by calling `seven.start()`:

```
>>> import seven
>>> seven.start()

>>> import incompatible.modules

>>> seven.stop()  # optional
```

The above will install an import hook and preprocess all modules before importing them using `lib2to3` from the `Python 2.7 stdlib`. The installed hook will process all imported python modules until you call `seven.stop()`. If you don't stop it, all imports will be processed.

CHAPTER 2

Import whitelists

Preprocessing all imported modules might not always be a good idea, mainly for two reasons. Firs, it comes with a slight performance penalty. Second, more important reason is that exceptions raised in modules that have been processed by the import hook can be really hard to debug. Line numbers are usually wrong, and the filename is not displayed correctly. These things might be improved in the future, but for now it is advised that only the necessary modules be processed.

The following example will only preprocess the modules `foo`, `spam.eggs` and their submodules, `foo.*` and `spam.eggs.*`:

```
>>> seven.start(['foo', 'spam.eggs'])
```

CHAPTER 3

What gets fixed?

Of course, not all fixers have been implemented yet. The following features are available (without using `from __future__` imports):

- `with` statement (`from __future__ import with_statement`)
- absolute imports (`from __future__ import absolute_imports`)
- integer division (`from __future__ import division`)
- class decorators

Class decorators are converted like this:

```
@decorated
class C:
    pass
```

becomes:

```
class C:
    pass
C = decorated(C)
```

CHAPTER 4

How to disable logging and pickling?

On certain production environments, logging might not be very useful, or writing files might not be possible (e.g. App Engine). To disable logging or writing pickle dumps, create a module named `sevenconfig` and add the `log = False` or `speedups = False` globals (or both). Make sure the module is importable before importing `seven`.

CHAPTER 5

What needs to be implemented?

The following features do not have fixers yet:

- indented `class` decorators
- `except Exception as e:`
- advanced string formatting
- `print` as a function
- `set` literals
- `dict` and `set` comprehensions
- multiple context managers in one `with` statement
- etc.

CHAPTER 6

How to write fixers?

By subclassing `seven.lib2to3.fixer_base.BaseFix`. You can also fork this project on GitHub, have a look at the existing fixers in `src/seven/fixes` and add your own, thend send me a pull request.