ipynb Documentation

Release 0.1

Yuvi Panda

Jul 19, 2017

Contents

This module allow you to import `ipynb` as is they were classical python modules. Simply prepend `ipynb.fs.full` to the regular import.

The Zen of Python say it well: Explicit is better than implicit. Thus wile import-hook are useful they lack explicitness. This module is meant to fix this by allowing you to explicitly requiring notebook files.

This module does install an import hook, though it will only try to load packages that explicitly start with `ipynb.fs.`.

This is still highly work in progress, any feedback and improvement welcomed.

The source code for this package can be found on GitHub.

CHAPTER 1

Installation

To install this package you can use *pip <https://pypi.python.org/pypi/pip>* that should be available with your python distribution. Use the following at command prompt:

```
$ pip install ipynb --upgrade
```

Make sure to use a recent version of pip!

`ipynb` requires python 3.4 or above to work. It is technically possible to have it work on older python versions but might require quite some work. We would welcome your contributions.

Developer install

If you are developing the `ipynb` package, we suggest you do a developer install. After cloning the source code, from the root of the newly clone directory issue a:

```
$ pip install -e .
```

CHAPTER 2

How does `ipynb` work

The `ipynb` package setup an importhook which will automatically make available as a python module any `.ipynb` files as long as the import starts with `ipynb.fs.`.

CHAPTER 3

Limitation

Although `ipynb` files are often connected to IPython kernel, `ipynb` does not (yet?) support many of IPython syntactic features like `!shell command` as well as line magics (`%magic`) and cell magics (`%%cell_magic`). While the former should be pretty easy to emulate, the two later one requires the code to be ran from with the main namespace of IPython so are unavailable.

CHAPTER 4

Import only definitions

If you would like to import only class & function definitions from a notebook (and not the top level statements), you can use `ipynb.fs.defs` instead of `ipynb.fs.full`. Full uppercase variable assignment will get evaluated as well.

CHAPTER 5

Relative imports

If you want to import notebooks from other notebooks relatively, you can easily do that with the following:

```
` import ipynb.fs from .full.notebook1 import foo from .defs.notebook2 import
bar `
```

This will do the imports from other notebooks relative to the path of the notebook in which the importing is happening. The *import ipynb.fs* is boilerplate that is required for this feature to work properly.

CHAPTER 6

Releasing a package that contains notebook files

You might have the need to release a python package with some modules written as `.ipynb` files, but you do not want to require the `ipynb` package for your users.

If you are using *setuptools*, you can import *ipynb.setup.find_packages*, which will convert `.ipynb` files to python files on before building an source distribution or a wheel. This allows others to use your package without needing to have the `ipynb` package installed.

Listing 6.1: setup.py

```
from ipynb.setup import find_packages
from setuptools import setup

setup(
    name='trombulator',
    version='4.8.15162342',
    description='Trombulate with class using trombulator',
    url='http://tronbula.tor/',
    author='Rick Sanchez',
    author_email='morty@lubalubadub.dub',
    license='BSD',
    packages=find_packages(),
    python_requires='>=3.4'
)
```

Contents:

CHAPTER 7

Indices and tables

- genindex
- modindex
- search