isort Documentation

Release latest

September 07, 2015

Contents

isort your python imports for you so you don't have to.

isort is a Python utility / library to sort imports alphabetically, and automatically separated into sections. It provides a command line utility, Python library and plugins for various editors to quickly sort all your imports. It currently cleanly supports Python 2.6 - 3.5 using pies to achieve this without ugly hacks and/or py2to3.

Before isort:

```
from my_lib import Object

print("Hey")

import os

from my_lib import Object3

from my_lib import Object2

import sys

from third_party import lib15, lib1, lib2, lib3, lib4, lib5, lib6, lib7, lib8, lib9, lib10, lib11, li

import sys

from __future__ import absolute_import

from third_party import lib3

print("yo")
```

After isort:

```
from __future__ import absolute_import

import os
import sys

from third_party import (lib1, lib2, lib3, lib4, lib5, lib6, lib7, lib8,
                         lib9, lib10, lib11, lib12, lib13, lib14, lib15)

from my_lib import Object, Object2, Object3

print("Hey")
print("yo")
```

Installing isort is as simple as:

```
pip install isort
```

or if you prefer

```
easy_install isort
```

CHAPTER 1

Using isort

From the command line:

```
isort mypythonfile.py mypythonfile2.py
```

or recursively:

```
isort -rc .
```

which is equivalent to:

```
isort **/*.py
```

or to see the proposed changes without applying them:

```
isort mypythonfile.py --diff
```

Finally, to atomically run isort against a project, only applying changes if they don't introduce syntax errors do:

```
isort -rc --atomic .
```

(Note: this is disabled by default as it keeps isort from being able to run against code written using a different version of Python)

From within Python:

```
from isort import SortImports

SortImports("pythonfile.py")
```

or:

```
from isort import SortImports

new_contents = SortImports(file_contents=old_contents).output
```

From within Kate:

```
ctrl+[
```

or:

```
menu > Python > Sort Imports
```

CHAPTER 2

Installing isort's Kate plugin

For KDE 4.13+ / Pate 2.0+:

```
wget https://raw.github.com/timothycrosley/isort/master/kate_plugin/isort_plugin.py --output-document
wget https://raw.github.com/timothycrosley/isort/master/kate_plugin/isort_plugin_ui.rc --output-docur
wget https://raw.github.com/timothycrosley/isort/master/kate_plugin/katepart_isort.desktop --output-d
```

For all older versions:

```
wget https://raw.github.com/timothycrosley/isort/master/kate_plugin/isort_plugin_old.py --output-docu
```

You will then need to restart kate and enable Python Plugins as well as the isort plugin itself.

CHAPTER 3

Installing isort's for your preferred text editor

Several plugins have been written that enable to use isort from within a variety of text-editors. You can find a full list of them on the isort wiki. Additionally, I will enthusiastically accept pull requests that include plugins for other text editors and add documentation for them as I am notified.

CHAPTER 4

How does isort work?

isort parses specified files for global level import lines (imports outside of try / except blocks, functions, etc..) and puts them all at the top of the file grouped together by the type of import:

- Future
- Python Standard Library
- Third Party
- Current Python Project
- Explicitly Local (. before import, as in: `from . import x`)
- Custom Separate Sections (Defined by forced_separate list in configuration file)
- Custom Sections (Defined by sections list in configuration file)

Inside of each section the imports are sorted alphabetically. isort automatically removes duplicate python imports, and wraps long from imports to the specified line length (defaults to 80).

CHAPTER 5

When will isort not work?

If you ever have the situation where you need to have a try / except block in the middle of top-level imports or if your import order is directly linked to precedence.

For example: a common practice in Django settings files is importing * from various settings files to form a new settings file. In this case if any of the imports change order you are changing the settings definition itself.

However, you can configure isort to skip over just these files - or even to force certain imports to the top.

CHAPTER 6

Configuring isort

If you find the default isort settings do not work well for your project, isort provides several ways to adjust the behavior.

To configure isort for a single user create a `~/.isort.cfg` file:

```
[settings]
line_length=120
force_to_top=file1.py,file2.py
skip=file3.py,file4.py
known_future_library=future,pies
known_standard_library=std,std2
known_third_party=randomthirdparty
known_first_party=mylib1,mylib2
indent='    '
multi_line_output=3
length_sort=1
forced_separate=django.contrib,django.utils
default_section=FIRSTPARTY
```

Additionally, you can specify project level configuration simply by placing a `.isort.cfg` file at the root of your project. isort will look up to 25 directories up, from the the file it is ran against, to find a project specific configuration.

Or, if you prefer, you can add an isort section to your project's `setup.cfg` with any desired settings.

You can then override any of these settings by using command line arguments, or by passing in override values to the SortImports class.

Finally, as of version 3.0 isort supports editorconfig files using the standard syntax defined here: http://editorconfig.org/

Meaning you place any standard isort configuration parameters within a .editorconfig file under the `*.py` section and they will be honored.

For a full list of isort settings and their meanings take a look at the isort wiki.

CHAPTER 7

Multi line output modes

You will notice above the "multi_line_output" setting. This setting defines how from imports wrap when they extend past the line_length limit and has 6 possible settings:

0 - Grid

```
from third_party import (lib1, lib2, lib3,
                         lib4, lib5, ...)
```

1 - Vertical

```
from third_party import (lib1,
                         lib2,
                         lib3
                         lib4,
                         lib5,
                         ...)
```

2 - Hanging Indent

```
from third_party import \
    lib1, lib2, lib3, \
    lib4, lib5, lib6
```

3 - Vertical Hanging Indent

```
from third_party import (
    lib1,
    lib2,
    lib3,
    lib4,
)
```

4 - Hanging Grid

```
from third_party import (
    lib1, lib2, lib3, lib4,
    lib5, ...)
```

5 - Hanging Grid Grouped

```
from third_party import (
    lib1, lib2, lib3, lib4,
    lib5, ...
)
```

Alternatively, you can set `force_single_line` to `True` (`-sl` on the command line) and every import will appear on its own line:

```
from third_party import lib1
from third_party import lib2
from third_party import lib3
...
```

Note: to change the how constant indents appear - simply change the indent property with the following accepted formats: * Number of spaces you would like. For example: 4 would cause standard 4 space indentation. * Tab * A verbatim string with quotes around it.

For example:

```
"    "
```

is equivalent to 4.

For the import styles that use parentheses, you can control whether or not to include a trailing comma after the last import with the `include_trailing_comma` option (defaults to `False`).

CHAPTER 8

Intelligently Balanced Multi-line Imports

As of isort 3.1.0 support for balanced multi-line imports has been added. With this enabled isort will dynamically change the import length to the one that produces the most balanced grid, while staying below the maximum import length defined.

Example:

```
from __future__ import (absolute_import, division,
                        print_function, unicode_literals)
```

Will be produced instead of:

```
from __future__ import (absolute_import, division, print_function,
                        unicode_literals)
```

To enable this set `balanced_wrapping` to `True` in your config or pass the `-e` option into the command line utility.

CHAPTER 9

Custom Sections and Ordering

You can change the section order with `sections` option from the default of:

```
FUTURE,STDLIB,THIRDPARTY,FIRSTPARTY,LOCALFOLDER
```

to your preference:

```
sections=FUTURE,STDLIB,FIRSTPARTY,THIRDPARTY,LOCALFOLDER
```

You also can define your own sections and thier order.

Example:

```
known_django=django
known_pandas=pandas,numpy
sections=FUTURE,STDLIB,DJANGO,THIRDPARTY,PANDAS,FIRSTPARTY,LOCALFOLDER
```

would create two new sections with the specified known modules.

CHAPTER 10

Auto-comment import sections

Some projects prefer to have import sections uniquely titled to aid in identifying the sections quickly when visually scanning. isort can automate this as well. To do this simply set the `import_heading_{section_name}` setting for each section you wish to have auto commented - to the desired comment.

For Example:

```
import_heading_stdlib=Standard Library
import_heading_firstparty=My Stuff
```

Would lead to output looking like the following:

```
# Standard Library
import os
import sys

import django.settings

# My Stuff
import myproject.test
```

CHAPTER 11

Ordering by import length

isort also makes it easy to sort your imports by length, simply by setting the `length_sort` option to `True`. This will result in the following output style:

```
from evn.util import (
    Pool,
    Dict,
    Options,
    Constant,
    DecayDict,
    UnexpectedCodePath,
)
```

CHAPTER 12

Skip processing of imports (outside of configuration)

To make isort ignore a single import simply add a comment at the end of the import line containing the text `isort:skip`:

```
import module  # isort:skip
```

or:

```
from xyz import (abc,  # isort:skip
                 yo,
                 hey)
```

To make isort skip an entire file simply add `isort:skip_file` to the module's doc string:

```
""" my_module.py
    Best module ever

   isort:skip_file
"""

import b
import a
```

CHAPTER 13

Adding an import to multiple files

isort makes it easy to add an import statement across multiple files, while being assured it's correctly placed.

From the command line:

```
isort -a "from __future__ import print_function" *.py
```

from within Kate:

or:

CHAPTER 14

Removing an import from multiple files

isort also makes it easy to remove an import from multiple files, without having to be concerned with how it was originally formatted.

From the command line:

```
isort -r "os.system" *.py
```

from within Kate:

or:

CHAPTER 15

Using isort to verify code

15.1 The `--check-only` option

isort can also be used to used to verify that code is correctly formatted by running it with `-c`. Any files that contain incorrectly sorted and/or formatted imports will be outputted to `stderr`.

```
isort **/*.py -c -vb

SUCCESS: /home/timothy/Projects/Open_Source/isort/isort_kate_plugin.py Everything Looks Good!
ERROR: /home/timothy/Projects/Open_Source/isort/isort/isort.py Imports are incorrectly sorted.
```

One great place this can be used is with a pre-commit git hook, such as this one by @acdha:

https://gist.github.com/acdha/8717683

This can help to ensure a certain level of code quality throughout a project.

15.2 Git hook

isort provides a hook function that can be integrated into your Git pre-commit script to check Python code before committing.

To cause the commit to fail if there are isort errors (strict mode), include the following in `.git/hooks/pre-commit`:

```
from isort.hooks import git_hook

if __name__ == '__main__':
    sys.exit(git_hook(strict=True))
```

If you just want to display warnings, but allow the commit to happen anyway, call `git_hook` without the *strict* parameter.

15.3 Setuptools integration

Upon installation, isort enables a `setuptools` command that checks Python files declared by your project.

Running `python setup.py isort` on the command line will check the files listed in your `py_modules` and `packages`. If any warning is found, the command will exit with an error code:

```
$ python setup.py isort
```

Also, to allow users to be able to use the command without having to install isort themselves, add isort to the setup_requires of your `setup()` like so:

```
setup(
    name="project",
    packages=["project"],

    setup_requires=[
        "isort"
    ]
)
```

CHAPTER 16

Why isort?

isort simply stands for import sort. It was originally called "sortImports" however I got tired of typing the extra characters and came to the realization camelCase is not pythonic.

I wrote isort because in an organization I used to work in the manager came in one day and decided all code must have alphabetically sorted imports. The code base was huge - and he meant for us to do it by hand. However, being a programmer - I'm too lazy to spend 8 hours mindlessly performing a function, but not too lazy to spend 16 hours automating it. I was given permission to open source sortImports and here we are :)

Thanks and I hope you find isort useful!

~Timothy Crosley