Entering the world of entry_points

Poruri Sai Rahul,
Enthought Inc.

Before we start

- How many of you have used entry_points?
- How many of you are familiar with entry_points but haven't used them?
- How many of you don't know anything about entry_points?
- How many of you have created a package or written a setup.py file?
- How many of you have never looked at a setup.py file or created a package?

Before we start

- How many of you have used entry_points?
- How many of you are familiar with entry_points but haven't used them?
- How many of you don't know anything about entry_points?
- How many of you have created a package or written a setup.py file?
- How many of you have never looked at a setup.py file or created a package?

Before we start

- How many of you have used entry_points?
- How many of you are familiar with entry_points but haven't used them?
- How many of you don't know anything about entry_points?
- How many of you have created a package or written a setup.py file?
- How many of you have never looked at a setup.py file or created a package?

Before we start

- How many of you have used entry_points?
- How many of you are familiar with entry_points but haven't used them?
- How many of you don't know anything about entry_points?
- How many of you have created a package or written a setup.py file?
- How many of you have never looked at a setup.py file or created a package?

Before we start

- How many of you have used entry_points?
- How many of you are familiar with entry_points but haven't used them?
- How many of you don't know anything about entry_points?
- How many of you have created a package or written a setup.py file?
- How many of you have never looked at a setup.py file or created a package?

Entering the world of entry_points

Specifically, `entry_point` is a keyword argument to the `setup` function from the `setuptools` package

Generally, entry points can be used to

- make a Python package more accessible and
- extending a Python package's functionality

Making a package more accessible

If you didn't know, `pip` is a python package.

If the `pip` package didn't use entry points, you'd have to do

- `python -m pip install numpy`

Instead of

- `pip install numpy`

Making `pip` more accessible

```
entry_points={
    "console_scripts": [
        "pip=pip:main",
        "pip%s=pip:main" % sys.version[:1],
        "pip%s=pip:main" % sys.version[:3],
    ],
},
```

Making `pip` more accessible

- `entry_points` is a dictionary
- `console_scripts` is a recognized keyword
- “`pip=pip:main`” means that the function `main` in the `pip` package can be accessed from the command line using `pip`

Setting console_scripts

```
entry_points = {
    ‘console_scripts’ = [
        “command=package.submodule:main”,
        ...,
    ],
}
```

Setting console_scripts

Now, instead of

- `$ python -c “from package.submodule import main; main()”`

we can do

- `$ command`

Making `pip` more accessible

```
entry_points={
    "console_scripts": [
        "pip=pip:main",
        "pip%s=pip:main" % sys.version[:1],
        "pip%s=pip:main" % sys.version[:3],
    ],
},
```

Making flask more accessible

```
entry_points='''
    [console_scripts]
    flask=flask.cli:main
'''
```

Extending a package

Twine is a package/utility to interact with PyPI. It's used to register and upload packages/updates.

```
$ twine -h

usage: twine [-h] [--version] {register,upload}


positional arguments:

  {register,upload}
```

Extending Twine

The Twine developers are nice, they want to make it easy for you to add new positional arguments, on top of the existing `register/upload`.

```
def _registered_commands(group='twine.registered_commands'):

        registered_commands =
pkg_resources.iter_entry_points(group=group)

        return dict((c.name, c) for c in registered_commands)
```

Extending Twine

The Twine developers are nice, they want to make it easy for you to add new positional arguments, on top of the existing `register`/`upload` .

```
def _registered_commands(group='twine.registered_commands'):

    registered_commands = pkg_resources.iter_entry_points(group=group)

    return dict((c.name, c) for c in registered_commands)
```

Extending Twine

```
entry_points={
     "twine.registered_commands": [
          "upload = twine.commands.upload:main",
          "register = twine.commands.register:main",
     ],
     "console_scripts": [
          "twine = twine.__main__:main",
     ],
},
```

How

pkg_resources.WorkingSet.iter_entry_points

pkg_resources.Distribution.get_entry_map

https://github.com/pypa/pkg_resources/blob/master/pkg_resources/__init__.py

“… , while others (such as plugin discovery) will work correctly so long as "egg-info" metadata directories are available for relevant distributions.”

https://github.com/pypa/setuptools/blob/90419ce42bc728e2ad4f773dd3e1346c263ad207/docs/pkg_resources.txt#L25-L27

Ref :

https://setuptools.readthedocs.io/en/latest/pkg_resources.html#entry-points

https://setuptools.readthedocs.io/en/latest/formats.html#entry-points-txt-entry-point-plugin-metadata

https://setuptools.readthedocs.io/en/latest/setuptools.html

http://python-packaging.readthedocs.io/en/latest/command-line-scripts.html#the-scripts-keyword-argument

Appendix

Extending a package

- ***`Sphinx`*** is the default tool/package to build documentation in Python.
- A number of themes are builtin.
- What if you wanted to create a theme of your own?

- How many of you have come across ***Read the Docs***?
- What if you wanted to create ***Read the Docs***-styled docs using Sphinx?

Extending Sphinx

The package `sphinx_rtd_theme` contributes a theme using

```
entry_points = {
    'sphinx.html_themes': [
        'sphinx_rtd_theme = sphinx_rtd_theme',
    ]
}
```

Extending Sphinx

Sphinx looks for items contributed to this entry point :

```
entry_points = pkg_resources.iter_entry_points('sphinx.html_themes', name)
try:
    entry_point = next(entry_points)
    self.app.registry.load_extension(self.app, entry_point.module_name)
    return
except StopIteration:
    pass
```

Extending a package

```
from pkg_resources import iter_entry_points

for entry_point in iter_entry_points(“str”):

    do_something_with(entry_point.load())
```

Extending functionality using an entry point

```
from setuptools import setup

setup(
    entry_points={
        “str” = [“str = module.submodule:main”],
    },
)
```

entry_points = (('nose.plugins.0.10', None),

('nose.plugins', ZeroNinePlugin))

https://github.com/nose-devs/nose/blob/7c26ad1e6b7d308cafa328ad34736d34028c122a/nose/plugins/manager.py#L368-L402

```
for ep in
pkg_resources.iter_entry_points('flask.commands'):

    self.add_command(ep.load(), ep.name)
```

https://github.com/pallets/flask/blob/851eaa4db7cef513dae35286d816867d68a72049/flask/cli.py#L435-L446

entry_points = iter_entry_points('sphinx.builders', name)

https://github.com/sphinx-doc/sphinx/blob/007593fa810918f243223080598d7ef74fd48fe9/sphinx/registry.py#L68

entry_points =
pkg_resources.iter_entry_points('sphinx.html_themes', name)

for entry_point in
pkg_resources.iter_entry_points('sphinx_themes'):

https://github.com/sphinx-doc/sphinx/blob/007593fa810918f243223080598d7ef74fd48fe9/sphinx/theming.py#L217-L249

for ep in iter_entry_points('distutils.commands',
'build_sphinx'):

https://github.com/pypa/setuptools/blob/995d309317c6895a123c03df28bc8f51f6ead5f5/setuptools/command/upload_docs.py#L47-L50

https://github.com/pypa/setuptools/blob/a9d902519680132493176106fe46aa9d77eafd10/setuptools/command/sdist.py#L17-L21

```
      entry_points={
      'console_scripts': [
            'sphinx-build = sphinx:main',
            'sphinx-quickstart = sphinx.quickstart:main',
            'sphinx-apidoc = sphinx.apidoc:main',
            'sphinx-autogen = sphinx.ext.autosummary.generate:main',
      ],
      'distutils.commands': [
            'build_sphinx = sphinx.setup_command:BuildDoc',
      ],
},
```