import import

- dive into Python import statement

Konrad Hałas (@konradhalas)

Few words about me

- PyWaw co-organizer
- developer in SUNSCRAPERS
- MutPy author

Agenda

- presentation purpose
- basic info
- syntax history
- inside machinery
- best practices
- common problems

Presentation purpose

- understand how import statement works
- learn how to override standard import behavior

- prepare to meet with "magic" import code in other projects
- do it well and avoid problems

What is purpose of import statement?

- it allows to use code from other modules
- it allows to split code into multiple modules

What does import statement do?

- find a module
- initialize it if necessary
- define a name or names in the local namespace

Syntax history

Python 1.0.0 - 1994

```
    import_stmt:  "import" identifier ("," identifier)*
                | "from" identifier "import" identifier
("," identifier)*
                | "from" identifier "import" "*"
```

Python 1.5.0 - 1997

```
    import_stmt:  "import" module ("," module)*
                  | "from" module "import" identifier ("," 
identifier)*
                  | "from" module "import" "*"
    module:       (identifier ".")* identifier
```

Python 2.0.0 - 2000

```
    import_stmt:  "import" module ["as" name] ("," module 
["as" name] )*
                  | "from" module "import" identifier ["as" 
name]
                  ("," identifier ["as" name] )*
                  | "from" module "import" "*"
    module:       (identifier ".")* identifier
```

Python 2.5.0 - 2006

```
    import_stmt:       "import" module ["as" name] ( "," mo
dule ["as" name] )*
                       | "from" relative_module "import" id
entifier ["as" name]
                       ( "," identifier ["as" name] )*
                       | "from" relative_module "import" "
(" identifier ["as" name]
                       ( "," identifier ["as" name] )*
[","] ")"
                       | "from" module "import" "*"
    module :           (identifier ".")* identifier
    relative_module :  "."* module | "."+
    name :             identifier
```

import statement machinery

- defined in **PEP 302** ("New Import Hooks")
- introduced in **Python 2.3** (2003)
- complete integration in **Python 3.3** (2012)

Let's start...

Machinery structures

1. `sys.modules`
2. `sys.meta_path`
3. `sys.path`
4. `sys.path_hooks`
5. `sys.path_importer_cache`

`sys.modules`

This is a dictionary that maps module names to modules which have already been loaded.

In [1]:

```
import sys

sys.modules
```

Out[1]:

```
{'IPython': <module 'IPython' from '/Users/
konradhalas/dev/virtualenvs/import_import/l
ib/python3.3/site-packages/IPython/__init_
_.py'>,
 'IPython.config': <module 'IPython.config'
from '/Users/konradhalas/dev/virtualenvs/im
port_import/lib/python3.3/site-packages/IPy
thon/config/__init__.py'>,
 'IPython.config.application': <module 'IPy
thon.config.application' from '/Users/konra
dhalas/dev/virtualenvs/import_import/lib/py
thon3.3/site-packages/IPython/config/applic
ation.py'>,
 'IPython.config.configurable': <module 'IP
ython.config.configurable' from '/Users/kon
radhalas/dev/virtualenvs/import_import/lib/
python3.3/site-packages/IPython/config/conf
igurable.py'>,
 'IPython.config.loader': <module 'IPython.
config.loader' from '/Users/konradhalas/de
v/virtualenvs/import_import/lib/python3.3/s
ite-packages/IPython/config/loader.py'>,
 'IPython.core': <module 'IPython.core' fro
m '/Users/konradhalas/dev/virtualenvs/impor
t_import/lib/python3.3/site-packages/IPytho
n/core/__init__.py'>,
 'IPython.core.alias': <module 'IPython.cor
e.alias' from '/Users/konradhalas/dev/virtu
```

```
 'zmq.utils': <module 'zmq.utils' from '/Us
ers/konradhalas/dev/virtualenvs/import_impo
rt/lib/python3.3/site-packages/zmq/utils/__
init__.py'>,
 'zmq.utils.initthreads': <module 'zmq.util
s.initthreads' from '/Users/konradhalas/de
v/virtualenvs/import_import/lib/python3.3/s
ite-packages/zmq/utils/initthreads.so'>,
 'zmq.utils.jsonapi': <module 'zmq.utils.js
onapi' from '/Users/konradhalas/dev/virtual
envs/import_import/lib/python3.3/site-packa
ges/zmq/utils/jsonapi.py'>,
 'zmq.utils.strtypes': <module 'zmq.utils.s
trtypes' from '/Users/konradhalas/dev/virtu
alenvs/import_import/lib/python3.3/site-pac
kages/zmq/utils/strtypes.py'>}
```

sys.meta_path

A list of *finder* objects that have their `find_module()` methods called to see if one of the objects can find the module to be imported.

In [2]:

```
sys.meta_path
```

Out[2]:

```
[_frozen_importlib.BuiltinImporter,
 _frozen_importlib.FrozenImporter,
 _frozen_importlib.PathFinder]
```

Module finder

- it must implement `find_module(fullname, paths=None)` method
  - `fullname` - fully qualified name of the module
  - `paths` - list of paths
- it should return a *loader* object if the module was found, or `None` if it wasn't

`DummyFinder` - example

In [3]:

```
class DummyFinder:

    def find_module(self, fullname, paths=None):
        print('find_module - fullname: {}, paths: {}'.form
at(fullname, paths))
        return None

sys.meta_path.insert(0, DummyFinder())
```

In [5]:

```
import eggs
```

```
find_module - fullname: eggs, paths: None
```

In [6]:

```
from spam import bacon
```

```
find_module - fullname: spam, paths: None
find_module - fullname: spam.bacon, paths:
['./workspace/spam']
```

In [8]:

```
import spam.bacon
```

```
find_module - fullname: spam, paths: None
find_module - fullname: spam.bacon, paths:
['./workspace/spam']
```

In [9]:

```
'eggs' in sys.modules
```

Out[9]:

```
True
```

In [10]:

```
'spam' in sys.modules
```

Out[10]:

```
True
```

In [11]:

```
'spam.bacon' in sys.modules
```

Out[11]:

```
True
```

Module finder should return a module loader object

Module loader

- loader is typically returned by a finder
- it must define `load_module(fullname)` method
- this method should return the loaded module or raise `ImportError` exception

`DummyLoader` - example

In [13]:

```
class DummyFinder:

    def find_module(self, fullname, paths=None):
        print('find_module - fullname: {}, paths: {}'.form
at(fullname, paths))
        if fullname == 'eggs':
            return DummyLoader()
        else:
            return None

class DummyLoader:

    def load_module(self, fullname):
        print('load_module - fullname: {}'.format(fullname
))
        raise ImportError('spam spam spam')

sys.meta_path.insert(0, DummyFinder())
```

In [14]:

```
import eggs
```

```
---------------------------------------
-------------------------------
ImportError                               T
raceback (most recent call last)
<ipython-input-14-5de045e19dc5> in <module>
()
----> 1 import eggs

<ipython-input-13-96b35398c59f> in load_mod
ule(self, fullname)
     12     def load_module(self, fullname)
:
     13         print('load_module - fullna
me: {}'.format(fullname))
---> 14         raise ImportError('spam spa
m spam')
     15
     16 sys.meta_path.insert(0, DummyFinder
())

ImportError: spam spam spam

find_module - fullname: eggs, paths: None
load_module - fullname: eggs
```

finder + loader == importer

In [16]:

```
class DummyImporter:

    def find_module(self, fullname, paths=None):
        print('find_module - fullname: {}, paths: {}'.form
at(fullname, paths))
        if fullname == 'eggs':
            return self
        else:
            return None

    def load_module(self, fullname):
        print('load_module - fullname: {}'.format(fullname
))
        raise ImportError('spam spam spam')

sys.meta_path.insert(0, DummyImporter())
```

`load_module` method responsibilities

1. it must create empty module and add it to `sys.modules` or use existing one
2. it may set the `__file__` attribute of the module
3. it may set the `__name__` attribute of the module
4. if the module is a package, it must set the module `__path__` attribute
5. it must set the module `__loader__` attribute
6. it may set the `__package__` attribute of the module

7. if the module is a Python module, it should execute the module's code
8. it must return created module

`DynamicImporter` - example

In [18]:

```
import types

class DynamicImporter:

    def __init__(self, name, code):
        self.name = name
        self.code = code

    def find_module(self, name, path):
        if name == self.name:
            return self
        else:
            return None

    def load_module(self, name):
        if name in sys.modules:
            module = sys.modules[name]
        else:
            module = types.ModuleType(name)
            sys.modules[name] = module
        module.__loader__ = self
        exec(self.code, module.__dict__)
        return module
```

In [19]:

```
sys.meta_path.insert(0, DynamicImporter(name='spam', code=
'eggs = True'))

import spam

print(spam.eggs)
```

```
True
```

Circular dependency example

```
# a.py

import b

X = 1

# b.py

import a

print(a.X)
```

In [21]:

```
import a
```

```
---------------------------------------
-------------------------------
AttributeError                            T
raceback (most recent call last)
<ipython-input-21-370c047e3c7f> in <module>
()
----> 1 import a

/Users/konradhalas/dev/workspace/personal/i
mport_import_presentation/workspace/a.py in
 <module>()
----> 1 import b
      2
      3 X = 1

/Users/konradhalas/dev/workspace/personal/i
mport_import_presentation/workspace/b.py in
 <module>()
      1 import a
      2
----> 3 print(a.X)

AttributeError: 'module' object has no attr
ibute 'X'
```

Circular dependency solution

- merge modules
- extract common artefacts
- defer `import` (put it into method body)

How "normal" `import` works?

In [22]:

```
sys.meta_path
```

Out[22]:

```
[_frozen_importlib.BuiltinImporter,
 _frozen_importlib.FrozenImporter,
 _frozen_importlib.PathFinder]
```

Machinery structures

1. `sys.modules`
2. `sys.meta_path`
3. **`sys.path`**
4. **`sys.path_hooks`**
5. **`sys.path_importer_cache`**

`PathFinder` - path based finder

1. iterates over `sys.path`
2. for each *path entry* in `sys.path` iterates over every callable in `sys.path_hooks`
3. each of the *path entry hooks* in `sys.path_hooks` is called with a single argument - path

4. path entry hook should return *path entry finder* or raise `ImportError`
5. returend path entry finder is cached in `sys.path_importer_cache`
6. call `find_loader` method on returned path entry finder with a single argument - fully qualified module name
7. this method should return loader (and namespace portion) or raise `ImportError`

In [23]:

```
sys.path_hooks
```

Out[23]:

```
[zipimport.zipimporter, <function _frozen_i
mportlib.path_hook_for_FileFinder>]
```

In [24]:

```
sys.path_importer_cache
```

Out[24]:

```
{'.': FileFinder('.'),
 './workspace/': FileFinder('./workspac
e/'),
 './workspace/spam': FileFinder('./workspa
ce/spam'),
 '/Users/konradhalas/.ipython/extensions':
FileFinder('/Users/konradhalas/.ipython/ex
tensions'),
 '/Users/konradhalas/dev/virtualenvs/impor
t_import/bin/../lib/python3.3/': FileFinde
r('/Users/konradhalas/dev/virtualenvs/impo
rt_import/bin/../lib/python3.3/'),
 '/Users/konradhalas/dev/virtualenvs/impor
t_import/bin/../lib/python3.3/collection
s': FileFinder('/Users/konradhalas/dev/vir
tualenvs/import_import/bin/../lib/python3.
3/collections'),
 '/Users/konradhalas/dev/virtualenvs/impor
t_import/bin/../lib/python3.3/encodings':
FileFinder('/Users/konradhalas/dev/virtual
envs/import_import/bin/../lib/python3.3/en
codings'),
 '/Users/konradhalas/dev/virtualenvs/impor
t_import/bin/../lib/python3.3/lib-dynloa
d': FileFinder('/Users/konradhalas/dev/vir
tualenvs/import_import/bin/../lib/python3.
3/lib-dynload'),
 '/Users/konradhalas/dev/virtualenvs/impor
t_import/bin/../lib/python3.3/plat-darwi
n': FileFinder('/Users/konradhalas/dev/vir
```

```
n3.3/plat-darwin': FileFinder('/usr/local/
Cellar/python3/3.3.0/Frameworks/Python.fra
mework/Versions/3.3/lib/python3.3/plat-dar
win'),
 '/usr/local/Cellar/python3/3.3.0/Framewor
ks/Python.framework/Versions/3.3/lib/pytho
n3.3/sqlite3': FileFinder('/usr/local/Cell
ar/python3/3.3.0/Frameworks/Python.framewo
rk/Versions/3.3/lib/python3.3/sqlite3'),
 '/usr/local/Cellar/python3/3.3.0/Framewor
ks/Python.framework/Versions/3.3/lib/pytho
n3.3/urllib': FileFinder('/usr/local/Cella
r/python3/3.3.0/Frameworks/Python.framewor
k/Versions/3.3/lib/python3.3/urllib')}
```

In [25]:

```
def http_path_entry_hook(path):
    if path.startswith('http://') or path.startswith('http
s://'):
        return HttpPathEntryFinder(path)
    else:
        raise ImportError()
```

In [26]:

```
import requests

class HttpPathEntryFinder:

    def __init__(self, path):
        self.path = path

    def find_loader(self, name):
        url = self.path + name + '.py'
        if requests.head(url).status_code == 200:
            return HttpLoader(url), None
        else:
            raise ImportError()
```

In [27]:

```
class HttpLoader:

    def __init__(self, url):
        self.url = url

    def load_module(self, name):
        if name in sys.modules:
            module = sys.modules[name]
        else:
            module = types.ModuleType(name)
            sys.modules[name] = module
        module.__loader__ = self
        module.__file__ = self.url
        response = requests.get(self.url)
        exec(response.content, module.__dict__)
        return module
```

In [28]:

```
sys.path_hooks.insert(0, http_path_entry_hook)
sys.path.append('https://raw.github.com/jcrocholl/pep8/master/')
```

In [29]:

```
import pep8
```

In [30]:

```
pep8
```

Out[30]:

```
<module 'pep8' from 'https://raw.github.com/jcrocholl/pep8/master/pep8.py'>
```

In [31]:

```
pep8.Checker(lines=['x  +  y\n']).check_all()
```

```
stdin:1:2: E221 multiple spaces before operator
stdin:1:5: E222 multiple spaces after operator
```

Out[31]:

```
2
```

Good import practices

- put all imports at the top of the file
- divide imports into 3 sections by blank lines:

    - Python modules
    - third-party modules
    - your modules
- put two blank lines after last section
- each section should be sorted alphabetically
- only one import should be on each line
- avoid *star* imports
- don't import single artefact, import whole module
- use relative imports if you can
- avoid `from-import-as`
- do not hack `__init__.py`

Bad example

```
import requests, ast
from xml import *
from json import dump
from spam import eggs  # current package
from os import path as os_path
X = 1
```

Good example

```
import ast
import json
import os.path
from xml import sax

import requests
```

```
from . import eggs


X = 1
```

Not covered topics

- `importlib` and `imp` module
- namespace packages
- `BuiltinImporter` and `FrozenImporter`

Summary

- you could do magic with `import` statement
- `import` machinery is quite complicated, but also reasonable
- follow good practices
- don't do that at home

Thank you!

Q&A

Presentation source code:
bitbucket.org/khalas/import_import_presentation
(https://bitbucket.org/khalas/import_import_presentation)

`reload` function

- <= Python 2.7 - `reload` (builtin function)
- <= Python 3.3 - `imp.reload`
- >= Python 3.4 - `importlib.reload`