from * import python
...

Łukasz Taczuk
35C3

Plan

What problem are we trying to solve?

How does the import mechanism work

How to write your own import hook

Demo 

How to use import hooks for the greater good (or bad, while we're at it)

My own motivation

Creating a new game mode for Frontline in 500 lines of code

An Arma 3 mod success story

Summary

NumPy - Map representation

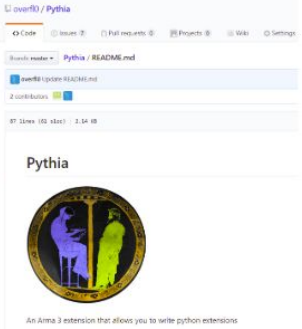
Scikit-image - Polygons, circles

plotlib - Isobar computation

y - Smoothing the frontline

Pythia

- Lets you write python Arma 3 extensions
- Still in development
- Fork it on github!



Best way to import custom code

```
def custom_import(module_name):
    contents = get_custom_module(module_name)
    eval(contents)
```

Not really!

please, don't leave, yet

That’s all folks!

•••

Thank you for staying until the end!
Do you have any questions?

Best way to import custom code

```
def custom_import(module_name):
    contents = get_custom_module(module_name)
    eval(contents)
```

What's wrong with using eval?

Custom way of importing

```
custom_import('stuff')
```

Can't do:

```
import stuff
```

Can't do:

```
from stuff import other_stuff
```

Can't easily import modules from **inside** the custom code

The code imported must be aware that it's imported in a custom way

Importing primer

```
parent/
    __init__.py
    one/
        __init__.py
    two/
        __init__.py

        and_a_half.py
```

```
import parent.one
import parent.two
import parent.two.and_a_half
```

Importing primer

```
>>> import parent.one

parent
one

>>> import parent.two

two

>>> import parent.two.and_a_half

and a half
```

```
parent/
    __init__.py
    one/
        __init__.py
    two/
        __init__.py

        and_a_half.py
```

finder

An object that tries to find the loader (spec) for a module that is being imported.

loader

An object that loads a module. It must define a method named load_module(). A loader is typically returned by a finder. See PEP 302 for details and importlib.abc.Loader for an abstract base class.

Importer

An object that both finds and loads a module; both a finder and loader object.

Importing

- The module name is converted to its fully qualified name:

  `parent.package.subpackage.module`

- Check `sys.modules` cache

- Find the module spec using *finders* (or *importers*)
  (note: older code may return loaders directly)

- Load the module using the *loader* (or *importers*)

Finding the module

1. Go through all registered finders, one by one (`sys.meta_path`)
2. Call the find_spec() method
3. Finder may return:
   a. `None` - Continue with the next finder
   b. Raise an exception - the exception is propagated
   c. return a `spec` object

   Note: returning a spec object doesn't load the module.
   It merely means "yes, I CAN load that module if you ask me to"

4. If all finders return None, raise `ModuleNotFoundError`
5. Otherwise take the spec and move to the loading part

Loading the module

1. The module is created either using the `create_module` method
2. If `create_method` returns `None` or doesn't exist, the import machinery creates the module itself
3. The module is executed using `exec_module`
4. If `exec_module` doesn't exist, `load_module` is called (deprecated)

Lots of things to do, lots of things to remember, but...

importlib.abc

ABC hierarchy

```
object
 +-- Finder (deprecated)
 |    +-- MetaPathFinder
 |    +-- PathEntryFinder
 +-- Loader
      +-- ResourceLoader --------+
      +-- InspectLoader          |
           +-- ExecutionLoader --+
                                 +-- FileLoader
                                 +-- SourceLoader
```

ABC hierarchy

```
object
 +-- Finder (deprecated)
 |    +-- MetaPathFinder
 |    +-- PathEntryFinder
 +-- Loader
      +-- ResourceLoader --------+
      +-- InspectLoader          |
           +-- ExecutionLoader --+
                                 +-- FileLoader
                                 +-- SourceLoader
```

MetaPathFinder

Finder stored in `sys.meta_path` - duh!

Contains the abstract method:

- `find_spec(fullname, path, target=None)`

MetaPathFinder

```
class MyModuleFinder(importlib.abc.MetaPathFinder):
   def find_spec(self, name, path, target=None):
       print('MyModuleFinder: Trying to load: {}'.format(name))

       if not can_import(name):
           return None

       return importlib.machinery.ModuleSpec(name, MyLoader(),
                   is_package=is_package(name))
```

MetaPathFinder

```
class MyModuleFinder(importlib.abc.MetaPathFinder):
    def find_spec(self, name, path, target=None):
        print('MyModuleFinder: Trying to load: {}'.format(name))

        if not can_import(name):
            return None

        return importlib.machinery.ModuleSpec(name, MyLoader(),
                    is_package=is_package(name))
```

ABC hierarchy

```
object
 +-- Finder (deprecated)
 |    +-- MetaPathFinder
 |    +-- PathEntryFinder
 +-- Loader
      +-- ResourceLoader --------+
      +-- InspectLoader          |
           +-- ExecutionLoader --+
                                 +-- FileLoader
                                 +-- SourceLoader
```

SourceLoader

An abstract base class for implementing source (and optionally bytecode) file loading.

Contains the abstract methods:

- `ResourceLoader.get_data(path)`
- `ExecutionLoader.get_filename(fullname)`

Note: you can also have a sourceless loader and one that loads .pyd files

SourceLoader

```
class MyLoader(importlib.abc.SourceLoader):
    def get_filename(self, fullname):
        print('MyLoader: Requesting filename for ', fullname)
        return get_mapped_filename(fullname)

    def get_data(self, filename):
        print('MyLoader: Fetching {} from our virtual'
              'filesystem'.format(filename))
        return data[filename]
```

MyImporter

```
class MyImporter(importlib.abc.SourceLoader,
                 importlib.abc.MetaPathFinder):

    def find_spec(self, name, path, target=None): ...

    def get_filename(self, fullname): ...

    def get_data(self, filename): ...
```

Installing the hooks

```
def install():
    sys.meta_path.insert(0, MyFinder())
    sys.meta_path.insert(0, MyImporter())
```

Demo

Import all the things!

Possible uses

Storing python code in a database (?)

Generating code on the fly (per requester)

Generating code based on the time (tokens)

Your own py2exe or PyInstaller

Code in proprietary format for python embedded in applications

DRM

CTFs :)

Links

https://docs.python.org/3/reference/import.html

https://docs.python.org/3/library/importlib.html

https://github.com/pyinstaller/pyinstaller/blob/develop/PyInstaller/loader/pyimod03_importers.py

Other links:

https://github.com/overfl0/stack_overflow_import :)

https://github.com/ivellios/py_github_import :) :)

That’s all folks!

•••

Thank you for staying until the end! Go crazy now!
Do you have any questions?

Reach me at: ***taczuk gmail***