Demystifying how imports work in Python

Tasdik Rahman (@tasdikrahman)

Presented @ ChennaiPy, October'16 meetup

Requirements

- Python 3.4 or newer.
- Material @ https://github.com/prodicus/talks
- No extra 3rd party extensions
- Coming over for this meetup!

Modules

- Any `python` source file would be counted as a module.
- You import a module to execute and access its classes/definitions/variables.

```
>>> import os
>>> os.path.abspath('.')
'/home/tasdik/Dropbox/talks/chennaipy/october/samplecode'
>>>
```

- `posixpath` would be the module name where the method `abspath()` resides.

What happens when you import a module?

- It being a python script, the statements start getting executed from top to bottom of the source file.
- If there are any tasks in the statements (eg: a `print()` statement), then they get executed when the module is being imported.

```
# 'samplecode/basicpackage/'
>>> import basicpackage.bar
inside basicpackage/__init__.py
inside 'basicpackage/bar'
>>>
```

Different styles for importing modules

from module import foo

- This essentially imports the module first then picks up specific parts from the module to be available locally.

```
>>> from basicpackage import foo
inside basicpackage/__init__.py
inside 'basicpackage/foo.py' with a variable in it
>>>
```

- allows using the parts of the module without giving the full prefix before it.

from module import *

- Brings out all the symbols from the module and makes them available in the namespace.

```
>>> from basicpackage_all import *
inside basicpackage_all/__init__.py
inside 'basicpackage_all/foo.py' with a variable in it
inside 'basicpackage_all/bar.py'
>>>
```

- You can use `__all__` inside your `__init__.py` module to import the modules which you need to import.
- **Generally not a good idea!**

Takeaways so far

- The way you import a module doesn't actually change the working of the module.
- Difference between `import foo.bar` and `from foo import bar` ?
  - the difference is subjective. Pick one style and be consistent with it.
  - doing a `from foo import bar` is more efficient.
    - `python` imports the whole file! period.

Module names

- naming modules follow the general variable naming convention.

```
# Bad choices
$ touch 2foo.py MyAwesomeFoo.py os.py

# Good choices
$ touch foo.py a_large_module_name.py
```

- Don't use Non-ASCII characters while doing so.
- Avoid creating module names which conflict with the standard library modules.

Module lookup

- If it's not in the python path, it just won't import.

```
>>> pprint(sys.path)
['',
 '/usr/lib/python35.zip',
 ...
 '/usr/lib/python3/dist-packages']
```

- Explicitly bring a module inside your path

```
>>> import sys
>>> sys.path.append('/absoule/path/to/module')
```