Diving deep on how imports work in Python

presented by Tasdik Rahman (@tasdikrahman)

PyCon Taiwan, 2017

But Why?

Modules

Terminology

- Loader

  Loads a module

- Finder

  Finds a module

Module search path

`sys.path`

```
>>> import sys
>>>
>>> pprint(sys.path)
['',
 '/usr/local/Cellar/python3/3.6.0/Frameworks/
Python.framework/Versions/3.6/lib/python36.zip',
 '/usr/local/Cellar/python3/3.6.0/Frameworks/
Python.framework/Versions/3.6/lib/python3.6',
 '/usr/local/Cellar/python3/3.6.0/Frameworks/
Python.framework/Versions/3.6/lib/python3.6/lib-
dynload',
 '/usr/local/lib/python3.6/site-packages']
>>>
```

Compiled Python files

- __pycache__ under a name like module.*version*.pyc
- __pycache__/spam.cpython-33.pyc
- platform independent
- regular lookup with source checking for modification.

```
import foo
```

2 step process

- find a module, loading and initialising it if necessary
- define a name or names in the local namespace for the scope where the "import" statement occurs.

If module is retrieved successfully

```
import foo    # foo imported and bound locally

import foo.bar.baz
# foo.bar.baz imported, foo bound locally

import foo.bar.baz as fbb
# foo.bar.baz imported and bound as fbb
```

```
from foo.bar import  baz
```

- find the module specified in the "from" clause, loading and initialising if necessary
- for each of the identifiers specified in the "import" clauses:

1. check if the imported module has an attribute by that name
2. attempt to import a submodule with that name and check the imported module again for that attribute
3. if the attribute is not found, "ImportError" is raised.

```python
from foo.bar import baz
# foo.bar.baz imported and bound as baz

from foo import attr
# foo imported and foo.attr bound as attr
```

from foo **import** *

Packages

Advantages?

```
car/                              Top-level package
      __init__.py                 Initialize the car package
      engine/                     Subpackage for engine behaviour
              __init__.py
              rev.py
               temperature.py
               fuel.py
               coolant.py
              ...
      transmission/               Subpackage for transmission
              __init__.py
              forward.py
              reverse.py
              ...
      infotainment/               Subpackage for infotainment system
              __init__.py
              music.py
              reverseparking.py
              chilledbeer.py
              ...
```

__init__.py

```
import car.engine.rev

car.engine.rev.spin(…)

# OR
from car.engine import rev

rev.spin(...)

# OR
from car.engine.rev import spin

spin(...)
```

__all__

```python
from car.engine import *
```

```python
# car/engine/__init__.py

__all__ = ["temperature", "fuel", "coolant"]
```

What if there is no

___all___ ?

Intra-package reference

```
# car/engine/temperature

from car.transmission import forward

from . import forward

from .. import transmission

from ..infotainment import music
```

Some takeaways

Questions?

Would be happy to take them :)

http://tasdikrahman.me/
@tasdikrahman