Packages

date: 4/14/2020

Package — just a special type of module, it can contain other modules, even packages

```
>>> import urllib
>>> import urllib.request
>>> type(urllib)
<class 'module'>
>>> type(urllib.request)
<class 'module'>
```

- urllib.request is nested in urllib
- in this case: urllib is a package and urllib.request is a module

```
>>> urllib.__path__
['/usr/lib/python3.7/urllib']
>>> urllib.request.__path__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'urllib.request' has no attribute '__path__'
>>>
```

`__path__` is attribu list of file system path indicating where urllib lives and searches to find nested modules

This is a nature of distinction of packages and modules:

- packages are generally represented by directories.
- modules represented by single files.

How python locates packages?

When we are asking python to load a package, it looks to your file sys. and loads corresponding code.
How python knows where to look?
The answer is python checks path attribute of the standard sys librariy

```
>>> sys.path
['', '/usr/lib/python37.zip', '/usr/lib/python3.7', '/usr/lib/python3.7/lib-dynload', '/home/
```

When we are asking python to import package it starts checking through the directories in `sys.path` .

sys.path

```
>>> import sys
```

```
>>> sys.path
['', '/usr/lib/python37.zip', '/usr/lib/python3.7', '/usr/lib/python3.7/lib-dynload', '/home/
```

It can be even bigger, it depends on how and how many third party packages we are installed.

```
>>> sys.path[0]
'' #It is empty when we run python without argument $python3
>>>
```

> **sys.path[0] — instructs python to first look in the current directory**

PYTHONPATH

Environment variable listing path added to the sys.path

Format of PYTHONPATH is the PATH variable of your system:

- Windows: it is a semicolon seperated list of directories
- Linux and MacOS: it is a colon(:) seperated list of directories

```
$ export PYTHONPATH=not_searched
$ python3
>>> import sys
>>> [ p in sys.path if not_searched in p]
[path to not_searched]
```

Basic Package Usage

How are the packages implemented?

To create a normal package, you simply creating normal python source file and make sure that it is on sys.path.

sys.path/mypackage(your package dir)/ `__init__` .py

> **`__init__` file is often called package init file, is what makes the package a module.**

Let's go the our project directory:

- `mkdir reader` — make a directory
- `touch reader/__init__.py` — initialize module
- `python3` — open REPL
- `import reader`
  As we expected it will import it as module

```
```

```
>>> import reader
>>> type(reader)
<class 'module'>
>>> reader.__file__
'/home/kamil/my_projects/pluralsights/reader/__init__.py'
```

Lets create our module named reader.py in our reader package

- `touch reader.py`
- add the code below to reader.py:

```
class Reader:

  def __init__(self, filename):
    self.filename = filename
    self.f = open(filename, "rt")

  def read(self):
    return self.f.read()

  def close(self):
    return self.f.close()
```

- run REPL:

```
>>> from reader.reader import Reader
>>> Reader("reader/reader.py")
<reader.reader.Reader object at 0x7fe18e924ed0>
>>> ra = Reader("reader/reader.py")
>>> ra.read
<bound method Reader.read of <reader.reader.Reader object at 0x7fe18e92b1d0>>
>>> ra.read()
"class Reader:\n    def __init__(self,filename):\n        self.filename = filename\n        s
>>> ra.close()
```

- in order to get rid of second reader in our import
  in init file:

```
  from reader import Reader
```

- That's all.
- Packages are Modules that contains other modules.
- Packages are generally implemented as directories containing a special `__init__.py` file
- The `__init__.py` file is executed when the package is imported
- Packages contain sub packages which themselves are implemented wit `__init__.py` files in directories.