* Modules: normally python files.
* Packages: Represented as directories
* Packages have \_\_path\_\_ property where as modules don’t
  + Urllib.\_\_path\_\_ = ‘/opt/local/anaconda3/lib/python3.3/urllib’
* Modules are imported as objects in python.

**How python searches for modules?**

* Uses sys.path. Starts with first dir in sys.path and checks for the imported module’s name in the file system.

**What is PYTHONPATH?**

* Same as PATH on OS
* List of directories
* PYTHONPATH sets the default sys.path directories in sys.path list

**Tip**: \_\_init\_\_.py is called as “dunder init.py” file, where dunder stands for double-underscore

**How to create package structure?**

* Create package root along one of paths in sys.path
* Python source file should be under directories in sys.path
* To turn directory to a python package, add \_\_init\_\_.py file

**What is REPL?**

* Read – Eval – Print Loop
* Example: Python shell, java shell, DB shell, scala shell (REPL technologies)
* Shell based programming language, ex python supports REPL, we can interact with python through CLI
* REPL based DB – Influx DB, SQL Developer,
* REPL technologies = [python, scala, java, influx db, etc]

**What is namespace package?**

* The package doesn’t have \_\_init\_\_.py file
* By default python imports modules by looking at init file
* If init file Is not present then it is import as namespace package
* Namespace packages are packages defined across different separate directories

**Eg:**

**Farm > birds > eagle.py, vulture.py**

**Farm > animal > dog.py, lion.py**

**Sys.path.extend([‘Farm/birds’,’Farm/animal’])**

**Import farm.birds.eagle**

**Import farm.animal.dog**

Above case will be import as a namespace package. Note that the package has same name but underlying modules are different

**What are Executable directories?**

* Python can execute directories
* Python can also execute zip files
* Adding \_\_main\_\_.py makes the directory executable
* If \_\_main\_\_.py is not added it will raise exception saying \_\_main\_\_.py file is not found

**What is recommended python project structure?**

* Project\_name
  + \_\_main\_\_.py (if you want project to be executable)
  + Package\_dir
    - \_\_init\_\_.py
    - More\_source\_code.py
  + Test
    - \_\_init\_\_.py
    - Test\_more\_course\_code.py

**What are singletons in python?**

* Singletons = Only one object of its kind exists
  + Example: Class test
  + A = test.func()
  + B = test.func()
  + Without singleton : A is B == False
  + If test is a singleton then; A is B == True
  + Thus only object of type test can exist
* Singletons breaks modularity of code
* Application? 🡪 Balance update class for transaction applications. We would like to have only one object that can update balance, and this ensures proper control/authority to just one class to do this task
* Singleton modules
  + \_<variable\_name> ensures its not accessible directly (implementation detail that we don’t have client to use/access)
  + Eg:

|  |  |
| --- | --- |
| Registry.py  \_registry = [] (is not accessible)  Def add\_names(name):  \_registry.append(name)  Def read\_registry():  Print(iter(\_registry)) | \_registry is not accessible out of registry.py |