Modules and Packages

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Script

- Script is an executable file, i.e. when given to python it executes some statements
- The programs we have been writing are scripts
- The script may have functions, classes, etc, but must have some statements outside these which are executed when the file is run

Modules

- A module is a simple Python file that contains collections of functions and global variables and with having a .py extension file
- Modules typically have no executable instructions, but mostly definitions of functions, variables, classes, ..
- A module is meant to be imported and used in some script
- Generally, module refer to a file containing Python definitions and statements. Eg: A file Algebra.py containing some functions from algebra would make up the ‘Algebra’ module.
- A module defines a namespace - i.e. all the names (functions, global vars, ..) defined in module file

Using Modules and functions in it

- To use the functions / vars defined in a module in another python program, we need to import the module into our program - then the definitions become available
- When we import a module - the module is executed - but if it has functions, class definitions, then python just records their definitions but does not execute anything
- Access module elements: <module_name>.<fn_name>
  - This allows different modules to have same names - and no name conflict in the importing module

Inbuilt and External Modules

- There are two types of modules in Python: inbuilt and external.
- Inbuilt modules are modules that are included with the Python standard library and can be used without installing any additional packages.

  Examples: math, time, random

- External modules are modules that are not included with the Python standard library and need to be installed separately.

  Example: numpy, pandas, matplotlib

Installing External Modules

External modules can be installed using the pip package manager by running the following command in the command prompt:

```
pip install <module_name>
```

```
Example:
```

```
pip install numpy
```

Importing Modules

```
# Import module and access definitions using dot notation
import math
print(math.sqrt(16))
```

```
# Import and rename module and access definitions using dot notation
import math as m
print(m.sqrt(16))
```

```
# Import required definitions from module
from math import sqrt
print(sqrt(16))
```

```
# Import required definitions from module and rename
from math import sqrt as square_root
print(square_root(16))
```

```
# Import all definitions from module - not used due to possible side effects
from math import *
```

if __name__ == “__main__”:

- When we directly run a python file, then the the built-in var "__name__" is set to "__main__"
- On import stmt, the executables of the imported module are executed (function definitions are noted), __name__ is the name of the module
- So, when we run a program file, __name__ is __main__, but when we import it, __name__ is the module name
- We can use this to define a python program which can be run as a script when needed, and imported when desired:
  - For executable statements (which we want to run as a script), first check if __name__ == “__main__” and then execute them.
- Any code placed under this is only executed when the python program is run directly and is ignored when the python file is imported as a module.

if __name__ == “__main__”:

x.py

```
def fx1(a):
    return a*a

def fx2(a):
    return a*a*a

if __name__ == "__main__":
    print(fx1(2))
    print(fx2(2))
```

y.py

```
import x

def fy1(a):
    return -a

if __name__ == "__main__":
    print(fy1(3))
    print(x.fx1(3))
```

```
~/Desktop  python3 x.py
4
8
~/Desktop  python3 y.py
-3
9
```

- When we execute x.py directly, the code stmts after if __name__ == “__main__” are executed.
- When we execute y.py, on import x, x.py will be executed; but these stmts are not executed; only function info is recorded - and functions fx1, fx2 become available in y
- Can access functions defined in x in y by x.fx1()

Quiz – Single Correct

What will be the output of code given below?

A. 4
B. 16
C. 20
D. 32

```
from math import sqrt

def my_func(x):
    return x*x

n = 4
a = my_func(n)
my_func = sqrt
b = my_func(a)

print(int(a+b))
```

Quiz – Single Correct

What will be the output of code given below?

```
from math import sqrt

def my_func(x):
    return x*x

n = 4
a = my_func(n)
my_func = sqrt
b = my_func(a)

print(int(a+b))
```

A. 4
B. 16
C. 20
D. 32

The first call of my_func used the definition given in the program. Then sqrt function of math module is assigned to my_func. Thus the second call of my_func used the definition of sqrt function.

Packages

- Collection of modules is a package
- Package - typically a directory, with sub-packages and modules in sub-directories
- Package directory must have __init.py__, which specifies a directory to be a package (typically initializes some things)
- Can import a package - all modules get imported
- Refer to a module name in a package: package.module.name
- Packages are also used through import statement
  - So, for all uses packages and modules are similar
- Python has many built-in modules (or packages) - reference readily available

Some common built-in Packages/Modules

- There are many built in modules, packages
  - os: functions relating to OS calls like mkdir, chdir, getcwd, listdir, ..
  - random: for generating random numbers: random, randomint, randrange, shuffle, choice, …
  - math: all types of math fns like sin, cos, tan, log, sqrt, .. also pi, e
  - sys: system level fns: argv, path, exit,
  - statistics: mean, median, mode, stddev, ..
  - time: time, localtime, sleep, …
- dir(module) gives the namespace of a module. Try dir('math')
- vars (module) gives a dictionary - names, and the types/values
  - vars(list)
- help(name) - provides info on name (fun, var, module…)
  - help('math'), help(list)

Libraries

- A library is a collection of related functionality of codes that allows you to perform many tasks without writing your code
- Library is just like a package (dont know if there is any difference)
- Used just like a package with the . notation
- Some people think of library as a collection of packages and modules

Summary

- Modules are python files with functions and some executables -generally in executables they will have some definitions only
- A module can be imported in another program - at import time the module is "executed" - all global vars and functions become avail
- Module names (fns, vars) are accessed as module_name.name
- Different ways to import a module in a program
- A module can act as a script and as an importable, by having statements executed only if __name__ is "__main__"
- Packages are collection of modules - they can also be imported