CH 8: Python Modules

**8.1. Introduction to python module**

Python module is very useful concept which allows coder to organize their code based on relevance. For example, if you are developing a shopping site, you might want to put all payment related API functions together, or wants to organize search related API functions together. It helps to organize the code-base in a structure manner. Thus, a module is a Python object with arbitrarily named attributes that you can bind and reference.

To make it clear, let’s have some example. Suppose we have 4 functions, which generates fruits, like apple, banana, coconut, watermelon. We can put all these function in a garder.py file, as they are related file.

def banana():

print 'You have a banana'

def coconut():

print 'You have a coconut'

def apple():

print 'You have an apple'

def watermelon():

print 'You have a watermelon'

Now, when we want to access any of these functions, you have to import the garden module first and access them using garden.

Example:

>>> import garden

>>> garden.banana()

You have a banana

>>> garden.coconut()

You have a coconut

>>> garden.apple()

You have an apple

>>> garden.watermelon()

You have a watermelon

>>>

If you get an error like this:

>>> import garden

Traceback (most recent call last):

File "<pyshell#68>", line 1, in <module>

import garden

ImportError: No module named garden

>>>

That means, you didn’t add the path of the garden.py location in python path. Please add it in sys.path variable

When we do an import, it will import all the functions defined in that module. We can also import individual function from module as below:

>>> from garden import apple

>>> apple()

You have an apple

>>>

We can also list of available function in a module using dir()

>>> import garden

>>> dir(garden)

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', '\_\_package\_\_', 'apple', 'banana', 'coconut', 'watermelon']

How a python interpreter locates a module to load? When you import a module, the Python interpreter searches for the module in the following sequences:

1. The current directory.
2. If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
3. If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

The module search path is stored in the system module sys as the sys.path variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default.

**8.2. Introduction to python Package**

As we discuss in previous section, module is a container of different related functions/codes. Python package is a collection of related modules. For example we have two modules: fruits.py and flowers.py. We want to put both this module in a garden package. For doing that, we can create a directory called “garden” and place this two file inside garden folder.

To treat “garden” directory as a python package, we need to add another file called \_\_init\_\_.py in the directory. Now garden contains three files.

We can import the python module from package in similar way.

>>> from garden.fruits import apple

>>> apple()

You have an apple

>>>

Still u cannot directly access apple() using garden. To do that, we need to add some statement in \_\_init\_\_.py file

from fruits import \*

from flower import \*

Now you can directly access the function using module, like this:

>>> from garden import apple

>>> apple()

You have an apple

>>>

**8.3. Common Python module**

In this section, we will discuss some useful module like sys, os and date-time module.

**8.3.1 Sys Module:** sys module is very important module in python. The sys module provides information about constants, functions and methods of the Python interpreter. Below are the example shows, how sys module works:

Example 1: Print some System Specific value using sys module:

>>> import sys

>>> sys.version # print the python version

'2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]'

>>> sys.byteorder # Print the Byte order to store the variable

'little'

>>> sys.copyright # Print Python Copy write information

'Copyright (c) 2001-2013 Python Software Foundation.\nAll Rights Reserved.\n\nCopyright (c) 2000 BeOpen.com.\nAll Rights Reserved.\n\nCopyright (c) 1995-2001 Corporation for National Research Initiatives.\nAll Rights Reserved.\n\nCopyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.\nAll Rights Reserved.'

>>> sys.getrecursionlimit() # Find the maximum recursion Depth is allowed by the system =>1000

>>> x = 10 # Decleare some variable and find out reference count.

>>> y = x

>>> sys.getrefcount(x)

54

>>> sys.getsizeof(x) # Find the size of a integer variable

12

>>> sys.getsizeof('python') # Find the how many bye is used to store a string

27

>>> sys.getwindowsversion() # Find out the windows version in which python is running

sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service\_pack='Service Pack 1')

>>> sys.path # Find out the python system path

['', 'C:\\Python27\\Lib\\idlelib', 'C:\\Windows\\system32\\python27.zip', 'C:\\Python27\\DLLs', 'C:\\Python27\\lib', 'C:\\Python27\\lib\\plat-win', 'C:\\Python27\\lib\\lib-tk', 'C:\\Python27', 'C:\\Python27\\lib\\site-packages']

>>> sys.platform # find out the System platform like windows or linux etc.

'win32'

*Example 2: Command line Argument:*

Lots of scripts need access to the arguments, passed to the script, when the script was started. sys.argv is a list, which generally stores the command-line arguments passed to the script. The first item of this list contains the name of the script itself, next are the passing arguments.

import sys

# it's easy to print this list of course:

print sys.argv

# or it can be iterated via a for loop:

for i in range(len(sys.argv)):

if i == 0:

print "Function name: %s" % sys.argv[0]

else:

print "%d. argument: %s" % (i,sys.argv[i])

print 'Total number of Argument passed: ',len(sys.argv)-1

Output :

C:\Users\dipankar.dutta\Documents>python test.py

['test.py']

Function name: test.py

Total number of Argument passed: 0

C:\Users\dipankar.dutta\Documents>python test.py a b c 10

['test.py', 'a', 'b', 'c', '10']

Function name: test.py

1. argument: a

2. argument: b

3. argument: c

4. argument: 10

Total number of Argument passed: 4

Example 3: Changing the output behavior of the interactive Python shell

Python's interactive mode is one of the things which make Python special among other programming languages like Perl or Java. We have already seen that, it's enough to write an expression on the command line and get back a meaningful output. However some users might prefer different output behavior. To change this behavior, we can rebind sys.displayhook to a callable object.

# This is an Default behaviors

>>> x = 42

>>> x

42

# Define a custom\_display function

>>> def custom\_display(x):

print 'Here is the output: ',x

# bind it with displayhook

>>> sys.displayhook = custom\_display

# Now print x again

>>> x

Here is the output: 42

# print an Expression

>>> 2+3+5+7

Here is the output: 17

>>> import sys

>>> sys.maxint

Here is the output: 2147483647

>>> sys.executable

Here is the output: C:\Python27\pythonw.exe

>>>

**8.3.2. OS Module**

The OS module in Python provides a way of using operating system dependent functionality, like creating a directory or listing the file in the directory etc. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux. It also helps us to find out useful information about the process. In this section we will give some example of OS functionality:

*Example1: Tryout basic OS manipulation using OS module*

# We need to import OS module first

import os

#Executing a shell command => os.system()

>>> os.system('pwd')

/home/dipankar/myExp

>>> os.system('ls')

onwire

Proti-paksha on SJDA scam-ZIqcJA6pwKk.mp4.part

youtap.py

#Returns the current working directory.

>>> os.getcwd()

'/home/dipankar/myExp'

#Return the real group id of the current process.

# Return the current process’s user id.

# Returns the real process ID of the current process.

>>> os.getpid()

10196

>>> os.getuid()

1000

>>> os.getgid()

1000

#Return information’s, identifying the current operating system.

>>> os.uname()

('Linux', 'dipankar-OptiPlex-390', '3.2.0-29-generic', '#46-Ubuntu SMP Fri Jul 27 17:03:23 UTC 2012', 'x86\_64')

>>> os.system('uname -a')

Linux dipankar-OptiPlex-390 3.2.0-29-generic #46-Ubuntu SMP Fri Jul 27 17:03:23 UTC 2012 x86\_64 x86\_64 x86\_64 GNU/Linux

#Return a list of the entries in the directory given by path.

>>> os.listdir('/')

['bin', 'tmp', 'sys', 'VMs', 'lib', 'proc', 'mnt', 'usr', 'etc', 'initrd.img', 'sbin', 'lost+found', 'var', 'dev', 'selinux', 'cdrom', 'boot', 'opt', 'home', 'blrshare', 'run', 'vmlinuz', 'srv', 'media', 'lib64', 'root']

#Create a directory named path with numeric mode mode.

>>> os.mkdir('python1')

>>> os.listdir('.') # Check if directory is created or not

['python1', 'youtap.py', 'Proti-paksha on SJDA scam-ZIqcJA6pwKk.mp4.part', 'onwire']

#Recursive directory creation function.

>>> os.makedirs('./python2/python21/python211/')

#Rename the file or directory src to dst :os.rename(src, dst)

>>> os.rename('python1','python100')

>>> os.listdir('.')

['youtap.py', 'Proti-paksha on SJDA scam-ZIqcJA6pwKk.mp4.part', 'onwire', 'python2', 'python100']

#Remove directories recursively using os.removedirs(path) abd Remove (delete) the directory path using os.rmdir(path)

>>> os.rmdir('python100')

>>> os.removedirs('./python2/python21/python211/')

>>> os.listdir('.')

['youtap.py', 'Proti-paksha on SJDA scam-ZIqcJA6pwKk.mp4.part', 'onwire']

#Walking aDir

>>> os.walk('.') # it return a generator

<generator object walk at 0x20f8780>

>>> a =os.walk('.')

>>> a.next() # it basically give current path, list of folder and list of file

('.', ['onwire', 'python2'], ['youtap.py', 'Proti-paksha on SJDA scam-ZIqcJA6pwKk.mp4.part'])

## Creating path using Join operaton.

>>> >>> os.path.join('.','onware')

'./onware'

>>> os.path.join('.','abc','abc.txt')

'./abc/abc.txt'

#find absolute Path

>>> os.path.abspath('.')

'/home/dipankar/myExp'

*Example2: Tryout basic OS manipulation using OS module*

In the following program we like to List of all the files, total count of files and folders & Total size of files.

import os

import sys

fileList = []

fileSize = 0

folderCount = 0

rootdir = sys.argv[1]

for root, subFolders, files in os.walk(rootdir):

folderCount += len(subFolders)

for file in files:

f = os.path.join(root,file)

fileSize = fileSize + os.path.getsize(f)

#print(f)

fileList.append(f)

print("Total Size is {0} bytes".format(fileSize))

print(“Total Files “, len(fileList))

print(“Total Folders “, folderCount)

**8.3.3 Date and Time Module**

In Python, There is a concept of ticks. Ticks are the time intervals number of seconds since 12:00am, January 1, 1970(epoch). Python provides number of module to handle with timing.

**Example 1:** Here we will show how we can use time and calendar module:

>>> import time

#Print Number of Ticks

>>> print 'Ticks', time.time() # =>Ticks 1386320291.2

# Print Time tuple Its <year, month, day, hour, min, sec, weekday, yearday, tdst >

>>> print 'Timetuple is: ',time.localtime(time.time())

Timetuple is: time.struct\_time(tm\_year=2013, tm\_mon=12, tm\_mday=6, tm\_hour=14, tm\_min=30, tm\_sec=20, tm\_wday=4, tm\_yday=340, tm\_isdst=0)

#Print Human readable Time format using asctime()

>>> print 'Formated time: ',time.asctime(time.localtime(time.time()))

Formated time: Fri Dec 06 14:31:16 2013

# Print the calendar of 2013 January month

>>> import calendar

>>> print 'calender of Jan of 2013 is : ',calendar.month(2013,1)

calender of Jan of 2013 is :

January 2013

Mo Tu We Th Fr Sa Su

1 2 3 4 5 6

7 8 9 10 11 12 13

14 15 16 17 18 19 20

21 22 23 24 25 26 27

28 29 30 31

**Example2: We can also get the time information using datetime module**. It’s easier than time module and support time operation. Strftime() provides a way to display date and time in your own way.

import datetime

print 'Current time:',datetime.datetime.now()

print 'Time on format: ',datetime.datetime.now().strftime("%y-%m-%d-%H-%M")

print "Current year: ", datetime.date.today().strftime("%Y")

print "Month of year: ", datetime.date.today().strftime("%B")

print "Week number of the year: ", datetime.date.today().strftime("%W")

print "Weekday of the week: ", datetime.date.today().strftime("%w")

print "Day of year: ", datetime.date.today().strftime("%j")

print "Day of the month : ", datetime.date.today().strftime("%d")

print "Day of week: ", datetime.date.today().strftime("%A")

**Output:**

Current time: 2013-12-06 14:58:01.659000

Time on format: 13-12-06-14-58

Current year: 2013

Month of year: December

Week number of the year: 48

Weekday of the week: 5

Day of year: 340

Day of the month : 06

Day of week: Friday

**Example 3: Arithmetic operation on a date in Python?**

Python also provide a way do arithmetic operation of date time. Like: We want to add 5 days with 04/21/2013. To do that we can create a datetime object using datetime.datetime() and doing arithmetic operation on this using +, - Operation. We can’t use the date directly but we can create a timedelta() and then do the operation.

>>> import datetime

>>> s = '04/21/2013'

>>> d = datetime.datetime.strptime(s, '%m/%d/%Y') # Create date time object

>>> d = d+ datetime.timedelta(days=1)

>>> print d.strftime('%m/%d/%Y')

04/22/201

Similar way, we can find out the difference between two dates,

>>> start = datetime.datetime(year=2012, month=2, day=25, hour=9)

>>> end = datetime.datetime(year=2012, month=2, day=25, hour=18)

>>> diff = end - start

>>> diff

datetime.timedelta(0, 32400)

>>> diff.total\_seconds()

32400

>>> diff.total\_seconds() / 60 / 60

9

OR

> import datetime

> print datetime.datetime.now() - datetime.datetime(2013, 1, 1)

55 days, 14:11:06.749378