**Modules**

A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example of a simple module, support.py −

def print\_func( par ):

print "Hello: ", par

return

**Note:** Within a module, the module’s name (as a string) is available as the value of the global variable \_\_name\_\_.

# The import Statement

Use any Python source file as a module by executing an import statement in some other Python source file.

**Syntax:**

import module1[, module2[,... moduleN]

A module is loaded only once, regardless of the number of times it is imported.

How?

#!/usr/bin/python3

import support # Import module support

# Now you can call defined function that module as follows

support.print\_func("Zara")

Output:

Hello: Zara

# The from...import Statement

To import specific attributes from a module into the current namespace.

**Syntax:**

from modname import name1[, name2[, ... nameN]]

#!/usr/bin/python3

# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n

result = []

a, b = 0, 1

while b < n:

result.append(b)

a, b = b, a + b

return result

>>> from fib import fib

>>> fib(100)

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

# The from...import \* Statement

To import all the names from a module into the current namespace

**Syntax:**

from modname import \*

# Executing Modules as Scripts

Within a module, the module’s name (as a string) is available as the value of the global variable \_\_name\_\_.

The code in the module will be executed, just as if you imported it, but with the \_\_name\_\_ set to "\_\_main\_\_".

#!/usr/bin/python3

# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n

result = []

a, b = 0, 1

while b < n:

result.append(b)

a, b = b, a + b

return result

if \_\_name\_\_ == "\_\_main\_\_":

f = fib(100)

print(f)

Output:

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

# Locating Modules

When you import a module, the Python interpreter searches for the module in the following sequences:

1. The current directory.
2. Python then searches each directory in the shell variable PYTHONPATH.
3. Default path. On UNIX, this default path is normally /usr/local/lib/python3/.

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.

# The PYTHONPATH Variable

The PYTHONPATH is an environment variable, consisting of a list of directories.

Windows system:

set PYTHONPATH = c:\python34\lib;

UNIX system:

set PYTHONPATH = /usr/local/lib/python

# Namespaces and Scoping

Variables are names (identifiers) that map to objects. A namespace is a dictionary of variable names (keys) and their corresponding objects (values).

1. A Python statement can access variables in a local namespace and in the global namespace. If a local and a global variable have the same name, the local variable shadows the global variable.
2. Each function has its own local namespace. Class methods follow the same scoping rule as ordinary functions.
3. Python makes educated guesses on whether variables are local or global. It assumes that any variable assigned a value in a function is local.
4. Therefore, in order to assign a value to a global variable within a function, you must first use the global statement.
5. The statement global VarName tells Python that VarName is a global variable. Python stops searching the local namespace for the variable.

#!/usr/bin/python3

Money = 2000

def AddMoney():

# Uncomment the following line to fix the code:

# global Money

Money = Money + 1

print(Money)

AddMoney()

print(Money)

Output:

2000

UnboundLocalError: local variable 'Money' referenced before assignment

#!/usr/bin/python3

Money = 2000

def AddMoney():

# Uncomment the following line to fix the code:

global Money

Money = Money + 1

print(Money)

AddMoney()

print(Money)

Output:

2000

2001

# The dir( ) Function

Returns a sorted list of strings containing the names defined by a module.

#!/usr/bin/python3

# Import built-in module math

import math

content = dir(math)

print(content)

Output:

['\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

# The globals() and locals() Functions

The globals() and locals() functions can be used to return the names in the global and local namespaces depending on the location from where they are called.

The return type of both these functions is dictionary. Therefore, names can be extracted using the keys() function.

# The reload() Function

if you want to reexecute the top-level code in a module, you can use the reload() function. The reload() function imports a previously imported module again.

**Syntax:**

reload(module\_name)

# Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Consider a Python files available in Phone directory.

Phone/G3.py file having function G3()

Phone/Isdn.py file having function Isdn()

Phone/Pots.py file having function Pots()

Now, create one more file \_\_init\_\_.py in the Phone directory −

Phone/\_\_init\_\_.py

To make all of your functions available when you have imported Phone, you need to put explicit import statements in \_\_init\_\_.py as follows:

from G3 import G3

from Isdn import Isdn

from Pots import Pots

After you add these lines to \_\_init\_\_.py, you have all of these classes available when you import the Phone package.

#!/usr/bin/python3

# Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

Output:

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

# END