Python Programming

Day 6: Modules and Packages

Modules and Packages

Script	Namespace
Modules	Packages

Color and symbol meaning

Hint

Preferred

Student's activity

Code to run

	Keyword
	In-built modules
	Strings
	Output

Variable Scope and Namespace

Namespace is a conceptual space that groups classes, identifiers etc. to avoid conflicts with items in unrelated code that have the same names

If a local and a global variable have the same name, the local variable shadows the global variable

Variable Scope and Namespace

Python assumes that any variable assigned to a value in a function is local.

Therefore, in order to assign a value to a global variable within a function, you must first use the **global statement**.

Variable Scope and Namespace

The statement **global VarName** tells Python that VarName is a global variable. Python stops searching the local namespace for the variable.

```
count = 2000

def AddCount():
    count +=1
    print (count)

AddCount()
print (count)
```

we define a variable count in the global namespace. Within the function AddCount, we assign count a value, therefore Python assumes count as a local variable.

Variable Scope and Namespace

However, we accessed the value of the local variable count before setting it, so an *UnboundLocalError* is the result.

```
count = 2000
def AddCount():
  global count
   count +=1
   print (count)
AddCount()
print (count)
Traceback (most recent call last):
 File "C:\Python33\test\all.py", line 228, in <module>
  AddCount()
 File "C:\Python33\test\all.py", line 225, in AddCount
  count +=1
UnboundLocalError: local variable 'count' referenced
before assignment
```

Module

A module is a collection of classes, functions and variables that is designed to be used inside another program.

Grouping related code into a module makes the code easier to understand and use.

Module

Benefits of Module

- Modules save time.
- You can write code once and use it in many programs.
- Modules hide complexity.
- Each module creates its own namespace, so its names won't conflict with names defined elsewhere.
- Modules make debugging easier.

Module

Example is a module named display.py

Within a module, the module's name (as a string) is stored in the global name __name__.

```
def print_func(var):
    print("Hello : ", var)
    return
```

Every module has a name stored in __name__.

Module and Script

A python file can be used as a module by executing an import statement in some other python source file or as a script.

A script is a python file which contains python statements meant to be executed from the command line or from within a Python interactive shell to perform a specific task.

Module and Script

Whenever a module is imported, any statements that are in the main body of the module are executed when it is imported.

Run well as a script without problem

```
def write_code():
    #This function writes code

write_code()

>>> python important.py
```

Module and __main__

The Problem comes up when we have multiple functions which we don't want to run at the same time

```
def write_code():
    #This function writes code
def delete_code():
    #This function deletes code

write_code()
delete_code()

>>> python important.py
```

Module and __main__

Here is the *Pythonic* way of writing a module and not as a script

```
def write_code( ):
   #This function writes code
def delete_code( ):
   #This function deletes code
def main( ):
   #business logic!
   if 'id'=1:
     write_code( )
  else:
     delete_code( )
if __name__ == '__main__':
   main( )
```

Import statement

A python file can be used as a module by executing an import statement in some other python source file or as a script.

There are two ways to import modules: You can import the module name or you can import one or more of the names defined inside the module.

Import statement

To import a module by name, type import followed by the name of the module without the .py suffix, like this:

```
import display
```

To access an attribute of a module, type the module name, a dot, and the attribute name. Importing the module gives access to its objects.

```
import inventory
inventory.add_item('laptops',
'compaq', 10)
```

Import statement

Importing by name

Importing a module by name stores the module name in the namespace. But it does not directly import any names that are defined in the module.

```
print(dir())
```

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', 'inventory', 'os', 'sys']
```

Import statement

To import a module from a different directory the path do the following

1. Add a new python file labelled ‘inventory’ to your folder
2. Add the following code to inventory.py

```
store = {}
laptops =
{'dell':10,'hp':15,'lenovo':20,'acer':5}
phones = {'samsung':50, 'iphone':40,
'tecno':100,‘ sony':25}
store['laptops'] = laptops
store['phones'] = phones
```

Import statement

```
def add_item(cat,item,qty):
    # function adds item to the store category
    if not isinstance(cat, str) or not isinstance(item, str)
or not isinstance(qty, int):
        print('Enter str category, str item and int quantity')
    elif cat not in store:
        print('Create category first')
    elif item not in store[cat]:
        store[cat][item] = qty
    else:
        count = store[cat][item]
        store[cat][item] = count + qty
```

Import statement

```
def del_item(cat, item, qty):
    #function decrements the qty of the given item
    if not isinstance(cat, str) or not isinstance(item, str) or
not isinstance(qty, int):
        print('Enter str category, str item and int quantity')
    elif cat not in store:
        print('Category does not exist')
    elif item not in store[cat]:
        store[cat][item] = qty
    else:
        count = store[cat][item]
        store[cat][item] = count + qty
```

Import statement

```
def add_cat(cat):
    #function adds category to store
    if len(cat) > 1 and isinstance(cat, str):
        store[cat]= {}

def view_store():
    print(store)
if __name__ == '__main__':
    add_item('laptop', 'compaq', 10)
    view_store()
```

Import statement

3. Add a new python file labelled sales
4. Add the following code to sales.py

```
import sys

sys.path.append('/path/to/your_folder_name/')

import inventory
inventory.view_store()
```

Import statement

The items inside modules are collectively called **attributes** of the module. To make sure that the module's name has been imported, type dir(), this will display a list with all the attributes present in the module.

print dir()
['__builtins__', '__doc__', '__name__', 'inventory']

Import statement

Importing items from inside a module

To import the function that's inside inventory, type this:

```
from inventory import add_item
add_item('laptops', 'compaq', 10)
```

It is important to note this approach can cause problems if the items being imported have the same names as other items you're working with

Import statement

Importing items from inside a module

Importing items from inside a module stores their names in the namespace, which gives you direct access to the items—you don't have to type the module name to use them and this can be useful in some situations

```
from inventory import add_item
add_item('laptops', 'compaq', 10)

print(dir())
['__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__',
'__package__', 'add_item', 'os', 'sys']
```

Import statement

Importing all items inside a module

It is also possible to import all names from a module into the current namespace:

```
from inventory import *
print(dir( ))
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', 'add_cat', 'add_item', 'del_item', 'laptops', 'os', 'phones', 'store', 'sys', 'view_store']
```

However, this statement should be used only if you're sure that it won't cause namespace conflicts.

Import statement

Importing a module using a different name

To import a module using a different name, type *import*, the *module name*, as, and the *name you want* to use.

import inventory as *inv*

Tip: You can also combine the from import and import as syntax, as follows:

from inventory import *add_item* as *a_i*

Package

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

A package has its own namespace which ensures that names defined within a package including module names won't conflict with names defined elsewhere.

Packages are stored in the lib/site-packages/directory

Package

admin/			Top Level Package				
	__init__.py		Initialize the admin package				
	Register/		Subpackage for registration				
		__init__.py					
		staff.py					
		students.py					
		Faculty.py					

	Finance/		Subpackage for Financials				
		__init__.py					
		salary.py					
		tuition.py					
		levy.py					

	Results/		Subpackage for results				
		__init__.py					
		alpha.py					
		omega.py					

Example structure for a package that handles school activities

Class Activity 1

Create a directory named calculator that has a module named average. Create a script named util that will import module average.
Module average contains a function that accepts 3 input from user and return the average of the 3 input values.

Note: The output should use string formatting to describe the average result.

Python Programming

Tutorials

Exercise 1:

Create ‘myFunctions’ module to contain all the functions for this tutorial, add a function that calculates the root of a quadratic equation and call the function from a script named ‘mod’.

Exercise 2:

Add another function into ‘myFunctions’ module that calculates compound interest, and call the function from ‘mod’ script created earlier.

$$A = P(1+r)^t$$

Exercise 3:

Add another function into ‘myFunctions’ module that calculates restaurant bill, and call the function from ‘mod’ script created earlier.

Use a dictionary to hold the food item and there cost
Sample; menu={‘rice’:300, ‘beans’:200, ‘eba’:200, ‘meat’:100}
Hint: the function will have 2 parameters (item, quantity)

Exercise 4:

Add another function into ‘myFunctions’ module that calculates dental payment of a patient. The patient will be requested to enter either or all of the services below

1. Cleaning at #1500,
2. Cavity filling at #8000,
3. x_ray at #3500

and call the function from ‘mod’ script created earlier to output the name and total cost of services requested by the user.

Exercise 5:

Create a new directory named ‘temp’ and place inside a script named ‘outfile’. Add another function ‘simple interest’ into ‘myFunctions’ module in a different directory then call this function from outfile.py

Next Lecture ...

Python Programming

Day 7: Control Statement