Functions, Packages, Modules, Classes, and Methods in Python

Session 04

Session Overview

In this session, you will be able to:

- Define functions
- Explain how to create user-defined functions
- Explain packages and modules
- Explain namespaces, instance methods, and static methods
- Define class and class methods

Functions

Function is a set of related statements used to execute an explicit task.

Python built-in functions such as `print()`, and `mylist.append()`.

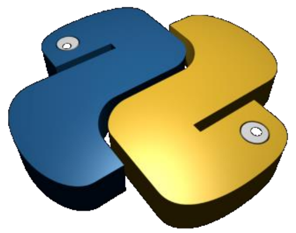
Parameters are the variables in the function.

Arguments are the values pass at the time of function call.

Defining a Function

Facts about Python function:

- Keyword `def` indicates the start of function header.
- Developer requires to provide a function name.
- Parameters (arguments) are optional.
- Colon (:) indicates the end of function header.
- `docstring` is optional.
- Function body requires one or more Python statements.
- `return` statement is optional.



Pass by Reference versus Value (1-2)

How a function that appends numeric ‘5’ to the list is passed?

Code Snippet :

```
>>> def changelist (mylist):
    "this function appends 5 to list"
    mylist.append(5)
    print ("Values inside function", mylist)
    return mylist
```

Pass by Reference versus Value (2-2)

```
>>> mylist=[3,4,5]
>>> changelist(mylist)
>>> print("Values outside function ", mylist)
```

Output

```
Values inside function [3, 4, 5, 6]
Values outside function [3, 4, 5, 6]
```

Docstring

How to print the _ _ doc _ _ string of the function changelist?

Code Snippet:

```
>>> print(changelist.__doc__)
#this function changes the passed list
```

The Return Statement

Function with a return statement:

Code Snippet :

```
>>> def sum_maker (number_list):
      "Finds and returns sum using list"
      list_sum = 0
      for num in number_list:
            list_sum += num
      return list_sum

>>> print ("sum of [4,6,7] is ",
sum_maker([4,5,6]))
```

```
Output:
sum of [4,6,7]
is 15
```

Required Arguments

Function with argument and return value:

Code Snippet:

```
>>> def sum_of_two (num1, num2):
     "Function returns sum of num1 and num2"
     return (num1+num2)
```

Default Arguments

How default arguments are used?

Code Snippet :

```
>>> def greet(name, msg = "Good morning!"):
    """
    This function greets to
    the person in ‘name’ with the
    provided message ‘msg’.

    If message is not provided,
    it defaults to "Good
    morning!"
    """
    print("Hello",name + ', ' + msg)
```

Keyword Arguments

Calling the function by passing the labels of the argument:

Code Snippet :

```
>>> greet(name="sam", msg = "Welcome")
```

Output:

```
Hello sam, Welcome
```

Variable Length Arguments

Variable length arguments:

Code Snippet :

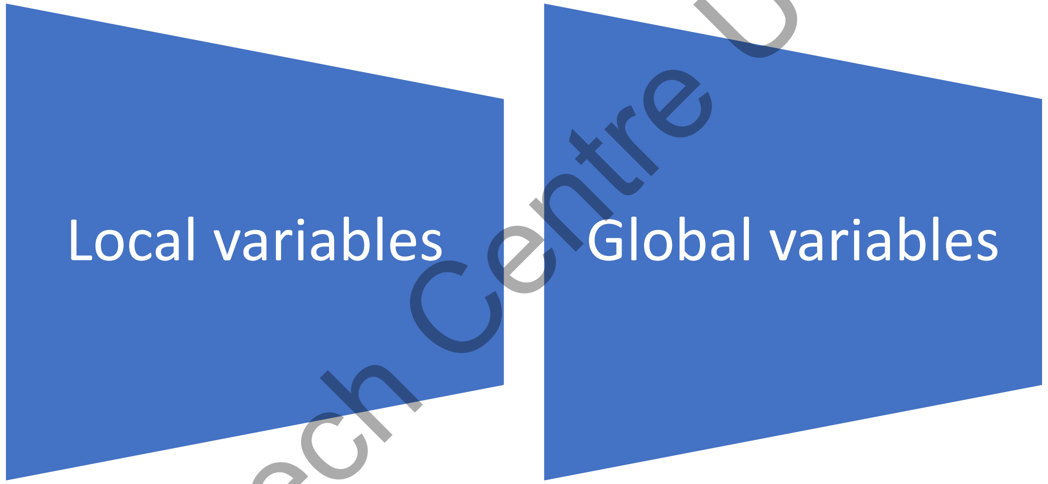
```
def functionname([formal_args,]
*var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

Anonymous Functions

- Functions are termed as anonymous, if they are not user define, which means by using the `def` keyword.
- Using `lambda` keyword to generate small anonymous functions.
- Lambda forms take any number of arguments but return only one value in the form of an expression.
- They do not contain commands or multiple expressions.

Scope of Variables

The scope of a variable defines the slice of the program where a developer can access a specific identifier.



Variables in Python

Global versus Local Variables

Variables which are declared inside a function body have a local scope and those described outside have a global scope.

Local variables can be accessed only within the function in which they have been stated.

Global variables can be accessed all over the program body by all functions.

When a developer calls a function, the variables stated inside the function are carried into scope.

Modules

- Modules offer an easy method to consolidate components into a system by helping as self-contained packages of variables termed as namespaces.
- All names defined at the top level of a module file turn into attributes of the imported module object.
- Modules refer to a file containing Python definitions and statements.
- File containing Python code, for example, `example.py`, is called a module and its module name is example.

Packages (1-2)

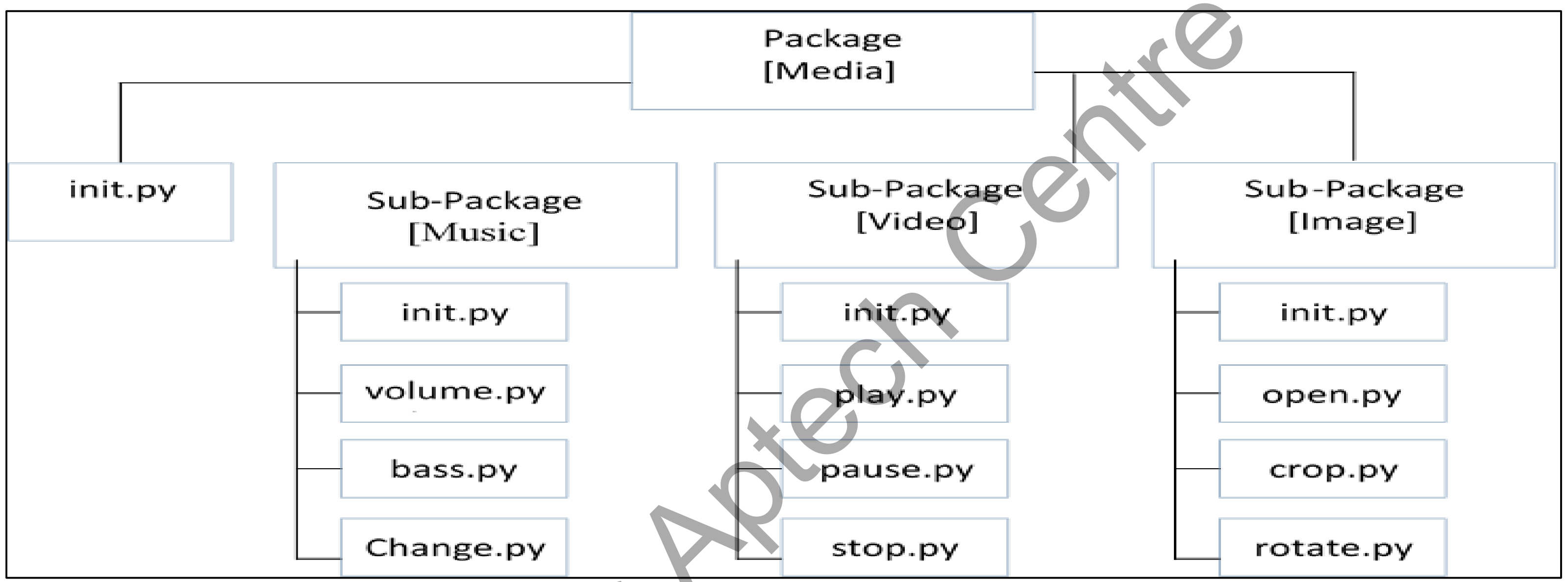
As the application program grows larger in size with various modules, a developer can place similar modules in one package.

This makes a project (program) simple to manage and practically clear.

Directory can contain files and sub-directories, similarly a Python package can have modules and sub-packages.

Packages (2-2)

Organization of packages and modules in a game being developed:



Package Folder Structure

Importing Modules from Packages

Developers can import modules from packages using the `dot (.)` operator.

How to import the crop module from the image?

Code Snippet:

```
import Media.Image.crop
```

Namespaces

Name is a term given to objects.

Everything in Python is an object.

Name is a method for accessing the underlying object.

Code Snippet:

```
>>> a = 2    # create a variable a with value 2
>>> print(id(a))   # print id of a
>>> print(id(2))   # print id of 2
```

Classes

Class is a blueprint for the object.

Example of a class definition:

Code Snippet:

```
class ClassDemo:
'''This is a docstring. This Class does nothing'''
 pass # this class does nothing just an empty class
```

Creating an Object

Class object is used to access diverse attributes.

Also be utilized to create new object instances (instantiation) of that specific class.

Process of creating an object is same as creating a function call.

Code Snippet:
```
>>> object_name = ClassName()
```

Constructors

Developer typically uses a constructor to initialize all the variables:

Code Snippet:

```
    >>> class ComplexNumber:
           def __init__(self,r = 0,i = 0): #this is
constructor
            self.real = r
           self.imag = i

           def getData(self):
            print("%d+%dj" %(self.real, self.imag))
```

Deleting Attributes and Objects (1-2)

Delete an attribute using the `del` statement:

Code Snippet:

```
>>> c1 = ComplexNumber(2,8)
>>> c1.attr = 10
>>> print (c1.attr)  # prior to deleting the attr
>>> del c1.attr
>>> print (c1.attr)  # post attr deletion
```

Deleting Attributes and Objects (2-2)

```
Traceback (most recent call last):
File "<pyshell#432>", line 1, in <module>
print (c1.attr)
AttributeError: 'ComplexNumber' object has no
attribute 'attr’

>>> print(c1)    #prior to deleting the object
<__main__.ComplexNumber object at 0x1013a7c18>

>>> del c1
>>> print (c1)  #post deleting the object

Traceback (most recent call last):
File "<pyshell#436>", line 1, in <module>
print (c1)
NameError: name 'c1' is not defined
```

Instance, Class, and Static Methods

Three different method types:

Code Snippet:

```
class MyClass:
        def method(self):
         return 'instance method called', self

        @classmethod
        def classmethod(cls):
   return 'class method called', cls

        @staticmethod
     def staticmethod():
   return 'static method called'
```

Instance Methods

Instance methods can access the class through the `self.__class__` attribute.

Calling an instance method:

Code Snippet:

```
>>> obj = MyClass()
>>> obj.method()
('instance method called', <MyClass instance at
0x101a2f4c8>)
```

Class Methods

Following Code Snippet illustrates the class method:

Code Snippet:

```
>>> obj.classmethod()
('class method called', <class
MyClass at 0x101a2f4c8>)
```

Static Methods

Static methods are predominantly a way to namespace the methods used in a code.

Static method:

Code Snippet:

```
>>> obj.staticmethod()
'static method called'
```

Summary (1-3)

- A function is a set of related statements used to execute an explicit task. Functions help to break down a program into minor and segmented chunks.
- A developer can define functions to provide the required functionality.
- Functions are termed anonymous if they are not declared in the standard way, which is by using the `def` keyword.

Summary (2-3)

- A developer can call a function through the following formal arguments:
  - Required arguments
  - Default arguments
  - Keyword arguments
  - Variable-length arguments
- Two basic scopes of variables in Python are local variables and global variables.

Summary (3-3)

- Modules offer an easy method to consolidate components into a system by helping as self-contained packages of variables termed as namespaces.
- A developer can place similar modules in one package and different modules in different packages. This makes a project (program) simple to manage.
- An object is a collection of methods (functions) and data (variables) that act on that data. Three method types available are the instance method, class method and static method.