4 Using Python Libraries

In This Chapter

4.1 Introduction
4.2 What is a Library ?
4.3 Importing Modules in a Python Program
4.4 Using Python Standard Library's Functions and Modules
4.5 Creating a Python Library

4.1 INTRODUCTION

You all must have read and enjoyed a lot of books – story books, novels, course-books, referenc books etc. And, if you recall, all these book types have one thing in common. Confused ? Don't be – all these book types are further divided into chapters. Can you tell, why is this done Yeah, you are right. Putting all the pages of one book or novel together, with no chapters, w make the book boring and difficult to comprehend. So, dividing a bigger unit into smal manageable units is a good strategy.

Similarly, in programming, if we create smaller handleable units, these are called *modules*. A related term is *library* here. A library refers to a collection of modules that together cater to specific type of needs or applications *e.g., NumPy* library of Python caters to scientific computing needs. In this chapter, we shall talk about using some Python libraries as well as creating own libraries/modules.

LIBRARY

A **library** refers to a collection modules that together cater specific type of needs applications

2 WHAT IS A LIBRARY ?

As mentioned above that a library is a collection of modules (and packages) that together cater to a specific type of applications or requirements. A library can have multiple modules in it. This will become clearer to you when we talk about 'what modules are' in coming lines.[1]

Some commonly used Python libraries are as listed below :

(*i*) **Python standard library.** This library is distributed with Python that contains modules for various types of functionalities. Some commonly used modules of Python standard library are :

- **math module**, which provides mathematical functions to support different types of calculations.
- **cmath module**, which provides mathematical functions for complex numbers.
- **random module**, which provides functions for generating pseudo-random numbers.
- **statistics module**, which provides mathematical statistics functions.
- **Urllib module**, which provides URL handling functions so that you can access websites from within your program.

(*ii*) **NumPy library.** This library provides some advance math functionalities along with tools to create and manipulate numeric arrays.

(*iii*) **SciPy library.** This is another useful library that offers algorithmic and mathematical tools for scientific calculations.

(*iv*) **tkinter library.** This library provides traditional Python user interface toolkit and helps you to create userfriendly GUI interface for different types of applications.

(*v*) **Malplotlib library.** This library offers many functions and tools to produce quality output in variety of formats such as plots, charts, graphs etc.

A library can have multiple modules in it. Let us know what a module means.

2.1 What is a Module ?

The act of partitioning a program into individual components (known as *modules*) is called **modularity**. A module is a separate unit in itself. The justification for partitioning a program is that

- it reduces its complexity to some degree and
- it creates a number of well-defined, documented boundaries within the program.

Another useful feature of having modules, is that its contents can be reused in other programs, without having to rewrite or recreate them.

For example, if someone has created a module say '**PlayAudio**' to play different audio formats, coming from different sources, *e.g.*, *mp3player*, *fm-radio player*, *dvd player* etc. Now, while writing a different program, if someone wants to incorporate *fm-radio* into it, he needs not re-write the code for it. Rather, he can use the *fm-radio functionality* from **PlayAudio** module. Isn't that amazing ? Re-usage without any re-work – well, that's the beauty of modules.

[1] ackage will become more clear to you in section 4.5.

4.2.1A Structure of a Python Module

A Python module can contain much more than just *functions*. A Python module is a normal Python file (*.py* file) containing *one* or *more* of the following objects related to a particular task :

- *docstrings* — triple quoted comments ; useful for documentation purposes. For documentation, the docstrings should be the first string stored inside a module/function-body/class.
- *variables and constants* — labels for data.
- *classes* — templates/blueprint to create objects of a certain kind.
- *objects* — instances of classes. In general, objects are representation of some real or abstract entity.
- *statements* — instructions.
- *functions* — named group of instructions.

So, we can say that the module '*XYZ*' means it is file '*XYZ. py*'.

Python comes loaded with some predefined modules that you can use and you can even create *your own modules*. The Python modules that come preloaded with Python are called ***standard library modules***. You have worked with one standard library module **math** earlier and in this chapter, you shall also learn to work with another standard library module : **random** module.

PYTHON MODULE

A **Python module** is a file (*.py* file) containing *variables, class definitions, statements* and *functions* related to a particular task.

Figure 4.1 shows the general composition / structure of a Python module.

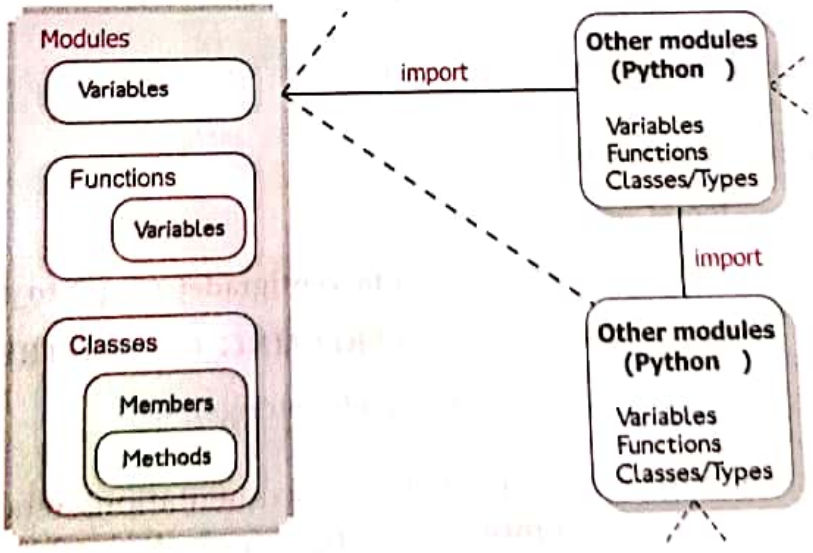

Figure 4.1 Composition/Structure of a Python Module

A module, in general :

- is independent grouping of code and data (variables, definitions, statements and functio
- can be re-used in other programs.
- can depend on other modules.

Let's have a look at an example module, namely *tempConversion* in figure 4.2.

(Please note, it is not from Python's standard library ; it is a user created module, shown here for your understanding purposes).

```
# tempConversion.py
""" "Conversion functions between fahrenheit and centigrade" """

# Functions
def to_centigrade(x):
    """ "Returns: x converted to centigrade" """
    return 5 * (x - 32) / 9.0

def to_fahrenheit(x):
    """ "Returns: x converted to fahrenheit" """
    return 9 * x / 5.0 + 32

# Constants
FREEZING_C = 0.0      # water freezing temp. (in celcius)
FREEZING_F = 32.0     # water freezing temp. (in fahrenheit)
```

A docstring : There are two more docstrings in this module – one each inside each function

Two function-definitions inside module tempConversion.py

STYLE-CONVENTION
Two blank lines between two function definitions

Two constants defined

Figure 4.2 A sample module (tempConversion.py)

The elements of module shown in figure 4.2 are :

Name of the module	**tempConversion**		
Module file name	**tempConversion.py**		
Contains	*Two Functions*	(*i*) **to_centigrade( )**	(*ii*) **to_fahrenheit( )**
	Two Constants[2]	(*i*) **FREEZING_C**	(*ii*) **FREEZING_F**
	Three docstrings	(triple quotes strings)	

The docstrings of a module are displayed as documentation, when you issue followi command on Python's Shell prompt >>>, after importing the module with imp **<module-name>** command :

```
help(<module-name>)
```

or example, after importing module given in Fig. 4.2, *i.e.*, module **tempConversion**, if we wr

```
help (tempConversion)
```

entity like constants in Python : it is just for understanding

Python will display all *docstrings* along with module name, filename, functions' name and constant as shown below :

```
>>> import temConversion
>>> help(tempConversion)
Help on module tempConversion :

NAME
    tempConversion - Conversion functions between fahrenheit and centigrade

FILE
    c:\python37\pythonwork\tempconversion.py

FUNCTIONS
    to_centigrade(x)
        Returns : x converted to centigrade
    to_fahrenheit(x)
        Returns : x converted to fahrenheit

DATA
    FREEZING_C = 0.0
    FREEZING_F = 32.0
```

The docstrings of module displayed as documentation

There is one more function dir( ) when applied to a module, gives you names of all that is defined inside the module. (see below)

```
>>> import tempConversion
>>> dir(tempConversion)
['FREEZING_C', 'FREEZING_F', '__builtins__', '__doc__', '__file__',
'__name__', '__package__', 'to_centigrade', 'to_fahrenheit' ]
```

NOTE

The docstrings are triple quoted strings in a Python module/program which are displayed as document when *help (<module-or-program-name>)* command is issued.

General docString conventions are :

- First letter of first line is a capital letter.
- Second line is blank line.