

PYTHON MODULES



**KAUSTAV GHOSH
DASTIDAR**

GIZMOFAACTS

Gizmofaacts.com

Python Modules

Preface

Always wanted to have a go at programming? No more excuses, Python is the ideal way to get started!

Python is an excellent programming language for both novices and experts. It is designed with code readability in mind, making it an excellent choice for beginners learning various programming concepts.

The language is widely used and has many libraries, allowing programmers to accomplish a lot with little code.

I hope you enjoyed my previous five modules, Python Fundamentals, Python Decision Making and Controls, Python Numbers and Strings, Python Lists and Tuples, and Python Date, Time, and Functions. If not, please proceed with a rush revision. Another interesting module I'll go over is: Python modules.

A module is a file that contains Python definitions and statements. A module is a file that contains Python code; for example, a module named "Program" would be a "Program.py" file. We use modules to break down large programmes into smaller, more manageable chunks. Modules allow for code reuse as well. Instead of copying their definitions into multiple programmes, we can define and import our most frequently used functions in a module.

This book is written to help you to learn Python programming quickly and effectively.

Enjoy Reading!

With Warm Regards,

Kaustav Ghosh Dastidar

Author & Founder of Gizmofacts



About The Author

The man behind **Gizmofacts**, **Kaustav Ghosh Dastidar** holds a Master Degree in Computer Science. He is usually turned on (metaphorically) by technology and gadgets. Born and raised in a quintessential middle class family he has been well aware of the ignorance the mass harbours about technology. Through Gizmofacts he wants to reach out to all those people, who he believes just need a little push to get into this unique and amazing world of science and software.

Moreover, Kaustav is well aware that nurturing an interest in gadgets doesn't come cheap. Hence he wants to also be an enabler who would provide all the 'need to know' financial details of different gadgets so that people can live their dreams remaining in their budget.

To know more about tips and tricks of softwares, gizmos and mobile apps, follow him in **Twitter**, **Facebook** and Google+.

You may also subscribe to Gizmofacts in **Youtube** for getting more information about software tips & tricks.

Disclaimer

Copyright @ 2022 Kaustav Ghosh Dastidar All rights reserved.

This eBook may not be copied or distributed without permission in any way. This publication's content is offered solely for informational reasons. The usage or misuse of this eBook, as well as any financial damage incurred by individuals or property as a direct or indirect result of using this eBook, are not the author's responsibility.

We are unable to guarantee your success or outcomes in the future due to some unforeseeable risks associated with doing business online. You accept that the author is not responsible for any success or failure of your business that is related in any way to the download and use of our information and that the use of our information should be based on your due diligence.

Without the author's prior written consent, no portion of this eBook may be copied or otherwise distributed in any way, including electronically, mechanically, by photocopy, recording, or any other method

Table of Content

Python - Modules.....	6-11
Next Steps?.....	12

Python - Modules

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. 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

The Python code for a module named `aname` normally resides in a file named `aname.py`. Here's an example of a simple module, `support.py`

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

The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

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

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module `support.py`, you need to put the following command at the top of the script –

```
#!/usr/bin/python  
  
# Import module support  
import support  
  
# Now you can call defined function that module as follows  
support.print_func("Zara")
```

When the above code is executed, it produces the following result –

```
Hello : Zara
```

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

The *from...import* Statement

Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax –

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

For example, to import the function fibonacci from the module fib, use the following statement –

```
from fib import fibonacci
```

This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module.

The *from...import ** Statement

Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax –

```
from modname import *
```

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

Locating Modules

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

- The current directory.

- The current directory.
- If the module isn't found, Python then searches each directory in the shell variable `PYTHONPATH`.
- 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.

The *PYTHONPATH* Variable

The `PYTHONPATH` is an environment variable, consisting of a list of directories. The syntax of `PYTHONPATH` is the same as that of the shell variable `PATH`.

Here is a typical `PYTHONPATH` from a Windows system –

```
set PYTHONPATH = c:\python20\lib;
```

And here is a typical `PYTHONPATH` from a 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).

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.

Each function has its own local namespace. Class methods follow the same scoping rule as ordinary functions.

Python makes educated guesses on whether variables are local or global. It assumes that any variable assigned 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.

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

For example, we define a variable *Money* in the global namespace. Within the function *Money*, we assign *Money* a value, therefore Python assumes *Money* as a local variable. However, we accessed the value of the local variable *Money* before setting it, so an `UnboundLocalError` is the result. Uncommenting the global statement fixes the problem.

```
#!/usr/bin/python

Money = 2000
def AddMoney():
    # Uncomment the following line to fix the code:
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```

The `dir()` Function

The `dir()` built-in function returns a sorted list of strings containing the names defined by a module.

The list contains the names of all the modules, variables and functions that are defined in a module. Following is a simple example –

```
#!/usr/bin/python

# Import built-in module math
import math

content = dir(math)
print content
```

When the above code is executed, it produces the following result –

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

Here, the special string variable `__name__` is the module's name, and `__file__` is the filename from which the module was loaded.

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.

If *locals()* is called from within a function, it will return all the names that can be accessed locally from that function.

If *globals()* is called from within a function, it will return all the names that can be accessed globally from that function.

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

The *reload()* Function

When the module is imported into a script, the code in the top-level portion of a module is executed only once.

Therefore, 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. The syntax of the *reload()* function is this –

```
reload(module_name)
```

Here, *module_name* is the name of the module you want to reload and not the string containing the module name. For example, to reload *hello module*, do the following –

```
reload(hello)
```

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 file *Pots.py* available in Phone directory. This file has following line of source code –

```
#!/usr/bin/python

def Pots():
    print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above –

- *Phone/Isdn.py* file having function `Isdn()`
- *Phone/G3.py* file having function `G3()`

Now, create one more file `__init__.py` in *Phone* directory –

- *Phone/__init__.py*

To make all of your functions available when you've imported *Phone*, you need to put explicit import statements in `__init__.py` as follows –

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

After you add these lines to `__init__.py`, you have all of these classes available when you import the *Phone* package.

```
#!/usr/bin/python

# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

When the above code is executed, it produces the following result –

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

Next Steps?

Have any questions while reading the eBook? Want to discuss programming, gadgets and gaming? You can email us (kaustav@gizmofacts.com) where you can ask any questions you have in min2d and subscribe to our Newsletter and visit our [Facebook Page](#).

You can also check our blog “[Gizmofacts](#)” where you can find a ton of useful “how-to-guides”, tutorials around blogging, programming, gadget reviews and gaming.