

Topics:

- **Modules:**

- Importing module
- The from..import statement
- Name of Module
- Making your own modules
- The dir()function
- The Python Module

- Math module OS Module Sys Module Random module
Calendar module Date and Time module Turtle Module Thinker Module
String Module List Module

Modules

- Modules are used to categorize the code in python into smaller parts
- Grouping related code into modules makes us easier to understand and use
- Module is a file contains functions, classes and variables
- A module must be imported into your file by using “import”
- Module is a .py file that has definitions of all functions and variables and may have runnable code to be used in another program
- Modules allows us to reuse one or more functions in the programs, even in the programs in which they are not defined.
- **import Statement**
 - Any python source file can be used as module by executing an import statement in any other python source code
 - **Syntax: import module1, module2,.....moduleN**
 - When the interpreter encounters the import statement, it imports the module if it is present
 - Module is loaded only once regardless of the number of times it is imported

- Modules are pre-written pieces of code that are used to perform common tasks like generating random numbers, performing mathematical operations etc...
- The basic way to use module is to add **import module_name** as the first line of the program and then writing module_name.var to access functions and values with the name var in the module

```
#Program to print the sys.path variable
import sys # Standard Library Module
print('Python Path=\n',sys.path)
```

```
# sys module ( system ) -
# Functionalities related to the Python Interpreter
# and its environment
```

- When the import sys statement is executed, python looks for the sys.py module in one of the directories listed in it sys.path variable.If the file is found, then the statements in the module is executed

• **Locating Modules**

- A module imported in a program must be located and loaded into memory before it can be used
- When you import a module, the Python interpreter searches for the module in the following sequences –
 - The current directory.
 - If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH environment variable.
 - Then searches in Python installation specific path(C:\Python37\Lib)
 - If not found then an error ImportError exception is generated
- Once a module is located, it is loaded in memory. A compiled version of the module with file extension .pyc is generated
- Next time when the module is imported, this .pyc file is loaded, rather than the .py file to save the time of recompiling

```
>>> help('modules')
```

Please wait a moment while I gather a list of all available modules...

Module	asyncio	history	run
__future__	asyncore	hmac	runpy
__main__	atexit	html	runscript
_abc	audioop	http	s
_ast	autocomplete	hyperparser	sched
_asyncio	autocomplete_w	idle	scrolledlist
_bisect	autoexpand	idle_test	search
_blake2	base64	idlelib	searchbase
_bootlocale	bdb	imaplib	searchengine
_bz2	binascii	imghdr	secrets
_codecs	binhex	imp	select
_codecs_cn	bisect	importlib	selectors
_codecs_hk	browser	inspect	setuptools
_codecs_iso2022	builtins	io	shelve
_codecs_jp	bz2	iomenu	shlex
_codecs_kr	cProfile	ipaddress	shutil
_codecs_tw	calendar	itertools	signal
_collections	calltip_w	json	site
_collections_abc	calltips	keyword	smtpd
_compat_pickle	cgi	lib2to3	smtplib
_compression	gitb	linecache	sndhdr

NOTE: 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.

- **Writing modules**
- Writing Python modules is very simple.
- To create a module of your own, simply create a hello .py file with the module name, and then import it using the Python file name (without the .py extension) using the import command.

```
#save this program as hello.py
def print_func( par ):
    print( "Hello : ", par)
    return
```

#save this program as import.py

```
# Import module hello
import hello
# Now you can call defined function that module as follows
hello.print_func("Ravikanth")|
```

```
#create a file by name called addition.py
```

```
def add(a,b):  
    c=a+b  
    print("c:",c)  
    return
```

test.py - C:\Users\Ravikanth K\Desktop\MM\test.py (3.7.0)

File Edit Format Run Options Window Help

```
import addition as a  
# a is a code for import file called addition  
a.add(10,20)  
a.add(100,200)
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04
4)] on win32

Type "copyright", "credits" or "license()" for m
>>>

===== RESTART: C:\Users\Ravikanth K\Des

c: 30

c: 300

>>> |

Multiple Import Statements

File Edit Format Run Options Window Help

```
#Filename: message.py
def msg_method():
    print("Today looks something different in my life")
    return
```

display.py - C:\Users\Administrator\Desktop\display.py (3.7.0)

File Edit Format Run Options Window Help

```
#Filename: display.py
def display_method():
    print("You know U R working with Me")
    return
```

multiimport.py - C:/Users/Administrator/Desktop/multiimport.py (3.7.0)

File Edit Format Run Options Window Help

```
#Filename: multiimport.py
import message, display
message.msg_method()
display.display_method()
```

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Administrator/Desktop/multiimport.py
Today looks something different in my life
You know U R working with Me
>>>
```


from import statement

- A module may contain definition for many variables and functions. When you import a module, you can use any variable or function defined in that module. But if you want to use only selected variables or functions, then you can use the from...import statement.
- syntax: from modulename import name1[, name2[, ... nameN]]
- The another way we can import the modules as from import
- To import more than one item from a module, use a comma separated list. For example, to import the value of pi and sqrt() from the math module you can write,
- **Example**

```
# from import statement
#import math
from math import sqrt
print("math.sqrt(10) :", sqrt(10))
print('math.sqrt(5) : ', sqrt(5))
```

```
from math import pi
print("PI=",pi)
```

```
# from import statement
#import math
from math import sqrt,pi
print("math.sqrt(10):",sqrt(10))
print('math.sqrt(5):',sqrt(5))
print('math.pi:',pi)
```

The *from...import ** Statement

- It is also possible to import all names [functions] from a module by using * (Astrick) symbol – as **import ***

from modname import *

- modname indicates the Module Name
- * indicates all the functions of that module
- The import * statement imports all names except those beginning with an underscore (_)
- **Ex:**

```
from math import *  
print("math.sqrt(10) :", sqrt(10))  
print('math.sqrt(5) : ', sqrt(5))  
print('math.pi: ', pi)  
print("math.pow(10,2) :", pow(10,2))
```

- Import a module with a different name using the **as** keyword. This is particularly more important when a module is either has a long or confusing name

```
# Program to show the use of 'as' keyword
from math import sqrt as s
print(s(81))|
```

Name of Module

- Every module has a name. You can find the name of a module by using the **__name__** attribute of the module

```
# Program to print name of a module in which the statemts are written
print("hello")
print("Name of this module is:", __name__)
```

hello

Name of this module is: __main__

>>>

Note: For every standalone program written by the user the name of the module is __main__

The dir() Function

- dir() is a built-in function that lists the identifiers defined in a module.
- These identifiers may include functions, classes and variables.
- If no name is specified, the dir() will return the list of names defined in the current module.
- Ex:

```
# Program to demonstrate the use of dir() function
def print_var(x):
    print(x)
x=50
print_var(x)
print(dir())
```

```
50
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__'
__, '__package__', '__spec__', 'print_var', 'x']
>>> |
```

```
#Import built-in module math
import math
content = dir(math)
print( content)
```

```
['__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', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan'
, 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'ta
u', 'trunc']
```

```
>>> |
```

```
#Import built-in module turtle
import turtle
content = dir(turtle)
print( content)|
```

OS Module

- The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.
- Python os module can be used to perform tasks such as finding the name of present working directory, changing current working directory, checking if certain files or directories exist at a location, creating new directories, deleting existing files or directories, walking through a directory and performing operations on every file in the directory that satisfies some user-defined criteria, and a lot more.

Method	Description
rename()	To rename a file. It takes two arguments, existing_file_name and new_file_name.
remove()	To delete a file. It takes one argument. Pass the name of the file which is to be deleted as the argument of method.
mkdir()	To create a directory. A directory contains the files. It takes one argument which is the name of the directory.
chdir()	To change the current working directory. It takes one argument which is the name of the directory.
getcwd()	It gives the current working directory.
rmdir()	To delete a directory. It takes one argument which is the name of the directory.
listdir()	It displays all the files and sub directories inside a directory

Syntax with example:

Syntax: `os.rename(existing_file_name, new_file_name)`

- Ex: **import** `os`
`os.rename('mno.txt', 'pqr.txt')`

Syntax: `os.remove(file_name)`

- Ex: **import** `os`
`os.remove('mno.txt')`

Syntax: `os.mkdir("file_name")`

- Ex: **import** `os`
`os.mkdir("new")`

Syntax: `os.chdir("file_name")`

- Ex: **import** `os`
`os.chdir("new")`

Syntax: `os.getcwd()`

•Ex: **import** `os`

print (`os.getcwd()`)

Syntax: `os.rmdir("directory_name")`

•Ex: **import** `os`

`os.rmdir("new")`

NOTE: In order to delete a directory, it should be empty. In case directory is not empty first delete the files

Syntax: `os.listdir()`

`import os`

`print(os.listdir('D:\\\\Materail\\SL 2017'))`

Note: This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

```
import os
```

```
os.system("notepad") # It opens a Notepad
```

- **Note :** That `rmdir()` method can only remove empty directories.
- In order to remove a non-empty directory we can use the `rmtree()` method inside the `shutil` module.
- ```
>>>import shutil
 shutil.rmtree('test')
 os.listdir()
```

# Sys module

- The sys module provides information about constants, functions and methods of the Python interpreter. `dir(system)` gives a summary of the available constants, functions and methods. Like all the other modules, the sys module has to be imported with the import statement, i.e. `import sys`
- Like all other modules, the sys module has to be imported with `import stmt`.
- INFORMATION ON THE PYTHON INTERPRETER
- COMMAND-LINE ARGUMENTS
- CHANGING THE OUTPUT BEHAVIOUR OF THE INTERACTIVE PYTHON SHELL
- STANDARD DATA STREAMS
- REDIRECTIONS
- `Help("sys")`

• Ex:

```
import sys
print (sys.version)
print(sys.version_info)
print(sys.paltform)
```

```
import sys
print(sys.path)
print(sys.modules)
print(sys.maxunicode)
print(sys.executable)
```

```
import sys
for i in (sys.stdin, sys.stdout, sys.stderr):
 print(i)
```

## Math Module

- Provides access to mathematical functions like power, logarithmic, trigonometric, hyperbolic, angular conversion, constants etc

| Function       | Description                                                       |
|----------------|-------------------------------------------------------------------|
| abs()          | Absolute value of x: the positive distance between x & zero       |
| ceil()         | Ceiling of x: Smallest integer not less than x                    |
| cmp(x,y)       | -1 if $x < y$ , 0 if $x == y$ , or 1 if $x > y$                   |
| exp(x)         | Exponential of x: $e^x$                                           |
| floor(x)       | Floor of x : The largest integer not greater than x               |
| max(x1,x2....) | Largest of its arguments: the value closest to positive infinity  |
| min(x1,x2....) | Smallest of its arguments: the value closest to negative infinity |
| pow(x,y)       | Value of $x^{**}y$                                                |
| round(x[,n])   | x rounded to n digits from the decimal point                      |
| sqrt(x)        | Square root of x for $x > 0$                                      |

## String Module

- Includes built-in methods to manipulate strings. Consider the string , str=Infosys

| Function             | Description                                                                                 |
|----------------------|---------------------------------------------------------------------------------------------|
| str.count("s")       | Returns count of occurrence of character "s" in string str                                  |
| str.startswith("s")  | Returns true if string str starts with character "s"                                        |
| str.endswith("s")    | Returns true if string str ends with character "s"                                          |
| str.find("s")        | Returns index position of character "s" in string str if found else -1                      |
| str.replace("s","S") | Replaces all occurrences of character "s" with character "S" in string str                  |
| str.isdigit()        | Checks if all the characters in string str are digits and returns true or false accordingly |
| str.upper()          | Converts all the characters in string str to uppercase                                      |
| str.lower()          | Coverts all the characaters in string str to lowercase                                      |



# List Module

| Function                            | Description                                                      |
|-------------------------------------|------------------------------------------------------------------|
| <code>cmp(list1,list2)</code>       | Compares elements of both lists                                  |
| <code>len(list)</code>              | Gives total length of list                                       |
| <code>max(list)</code>              | Returns items from the list with maximum value                   |
| <code>min(list)</code>              | Converts a tuple to list                                         |
| <code>list(seq)</code>              | Converts a tuple to list                                         |
| <code>list.append(obj)</code>       | Appends object obj to list                                       |
| <code>list.count(obj)</code>        | Returns count of how many times obj occurs in list               |
| <code>list.insert(index,obj)</code> | Inserts object obj into list of offset index                     |
| <code>obj=list.pop()</code>         | Removes the items at position -1 from list and assigns it to obj |
| <code>list.remove(obj)</code>       | Removes object obj from list                                     |
| <code>list.reverse()</code>         | Returns the order of items in list                               |
| <code>sorted(list)</code>           | Sorts items in list                                              |

# Date and Time Module

- Supplies classes for manipulating dates and times in both simple and complex ways
- The time module provides a number of functions that deal with dates and the time within a day
- import time module : Ex: print(time.localtime())

| Function         | Description                                                                                |
|------------------|--------------------------------------------------------------------------------------------|
| time.clock ( )   | Returns current time in seconds, given as a floating point number                          |
| time.gmtime()    | Returns current UTC date and time( not affected by timezone)                               |
| time.localtime() | Returns the number of hours difference between your timezone and the UTC timezone (London) |
| time.time()      | Returns the number of seconds                                                              |
| time.sleep(secs) | Suspends execution of the current thread for the given number of seconds                   |
| time.daylight()  | Returns 0 if you are not currently in Daylight Saving time                                 |

## Ex:

```
import time
print(time.localtime())
```

**Output:**

```
import time
print(time.ctime())
```

**Output:**

```
import time
now = time.localtime(time.time())
print (time.asctime(now))
print (time.strftime("%y/%m/%d %H:%M", now))
print (time.strftime("%a %b %d", now))
print (time.strftime("%c", now))
```

**Output:**

- **Tick:** Time intervals are floating-point numbers in units of seconds.
- `import time`  
`ticks=time.time()`  
`print(" number of ticks since 09:00AM Sep 14,2017:",ticks)`
- `import time`  
`localtime = time.localtime(time.time())`  
`print ("Local current time :", localtime)`
- `import time`  
`localtime = time.asctime( time.localtime(time.time()) )`  
`print ("Local current time :", localtime)`

- **What is TimeTuple?**
- Many of Python's time functions handle time as a tuple of 9 numbers, as shown below –
- | <b>Index</b> | <b>Field</b>     | <b>Values</b>                              |
|--------------|------------------|--------------------------------------------|
| 0            | 4-digit year     | 2017                                       |
| 1            | Month            | 1 to 12                                    |
| 2            | Day              | 1 to 31                                    |
| 3            | Hour             | 0 to 23                                    |
| 4            | Minute           | 0 to 59                                    |
| 5            | Second           | 0 to 61 <i>60 or 61 are leap – seconds</i> |
| 6            | Day of Week      | 0 to 6 <i>0 is Monday</i>                  |
| 7            | Day of year      | 1 to 366 <i>Julianday</i>                  |
| 8            | Daylight savings | -1, 0, 1, -1 means library determines DST  |

| ● Index | Attributes | Values                                     |
|---------|------------|--------------------------------------------|
| ● 0     | tm_year    | 2008                                       |
| ● 1     | tm_mon     | 1 to 12                                    |
| ● 2     | tm_mday    | 1 to 31                                    |
| ● 3     | tm_hour    | 0 to 23                                    |
| ● 4     | tm_min     | 0 to 59                                    |
| ● 5     | tm_sec     | 0 to 61 <i>60 or 61 are leap — seconds</i> |
| ● 6     | tm_wday    | 0 to 6 <i>0 is Monday</i>                  |
| ● 7     | tm_yday    | 1 to 366 <i>Julian day</i>                 |
| ● 8     | tm_isdst   | -1, 0, 1, -1 means library determines DST  |

# Calendar Module

- The calendar module supplies calendar-related functions, including functions to print a text calendar for a given month or year.
- By default, calendar takes Monday as the first day of the week and Sunday as the last one. To change this, call `calendar.setfirstweekday` function.

- **`calendar.firstweekday`**

- **`calendar.isleap`**

- **`calendar.month`**

- **`calendar.monthrange`**

- `import calendar`

```
cal = calendar.month(2017, 1)
```

```
print ("Here is the calendar:")
```

```
print (cal)
```

# random Module

- The random module contains a number of random number generators.  
Ex: Dias Game, OTP Generator
- It is most commonly used in Cryptography

- **Example 1:**

```
import the random module
import random
print(random.randint(0,9))
```

**Output:**

- **Example 2:**

```
import random
random choice from a list
for i in range(5):
```

```
 print (random.choice([1, 2, 3, 5, 9]))
```

**Output:**



# Turtle Module

- Python supports both
  - Character User Interface [ Core]
  - Graphical User Interface [ Advance]
- Turtle has 70+ methods as of python3.7 version
- Turtle graphics is a popular way for introducing programming to kids.
- Roughly it has the following features : Better animation of the turtle movements, Different turtle shapes, Fine control over turtle movement and screen updates, controlling background color background image, window and canvas size, Appearance of the TurtleScreen and the Turtles at startup and many more behind the screen

# Ex:1

```
import turtle as tt
tt.fd(100)
tt.pencolor("red")
tt.bgcolor("blue")
tt.done()
```

Output:

```
import turtle as tt
for i in range(5):
 tt.forward(50)
 tt.left(30)
 tt.pensize(200)
 tt.pencolor("orange")
 tt.bgcolor("yellow")
 tt.color("green")
```

# Turtle Methods

```
import turtle as tt
tt.forward(100)
tt.right(100)
tt.left(100)
tt.write("E2 ROCK STARTS",font("Arial",100,"BOLD"))
tt.up()
tt.down()
tt.bye()
tt.done()
tt.dot()
tt.degree()
tt.position()
```

tt.delay(10)

tt.speed(10)

tt.position()

tt.windowwidth()

tt.windowheight()

tt.screensize()

tt.circle()

tt.turtle() # it supports 3 shapes turtle,angle,arrow

- Many more....



**For Details Contact Me @ :**  
**9247448766**  
**[ravikanth27787@gmail.com](mailto:ravikanth27787@gmail.com)**