There are many online resources available to get help in python programming, [#"Here is my list of online resources"](http://www.linuxnix.com/python-builtin-helpdir-help-type-and-___doc_-functions/#%22Here%20is%20my%20list%20of%20online%20resources%22). Instead of this python has many built in methods and functions which we can use to get much of information about python, python built in modules and other methods. This built in methods are very useful whether you are new to python programming or an experience programmer.

Following are the four basic in-built functions of python which we can help us to understand more about python built in modules, libraries, and functions;

* Python dir() function
* Python help() function
* Python type() function
* Python \_\_doc\_\_ function

| **Function** | **Description** |
| --- | --- |
| dir() | Returns the attributes of the object or module. |
| help() | Returns the python built in documentation about the object. |
| type() | Returns the type of object. |
| \_\_doc\_\_ | Returns the doc-string of object or module. |

**Python dir() function:**

    The dir function is one of the most useful functions in Python when it comes  
to digging around in Python's interactive mode. You provide it with the name of a module or object, and it returns lots of useful information about the contents of that module or object, the default object is the current program module, the one which the interpreter is now running. Let's try some examples;

>>> dir()

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_']

>>> dir(\_\_builtins\_\_)

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError', 'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'ReferenceError', 'RuntimeError', 'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '\_', '\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_name\_\_', '\_\_package\_\_', 'abs', 'all', 'any', 'apply', 'basestring', 'bin', 'bool', 'buffer', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'cmp', 'coerce', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'execfile', 'exit', 'file', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'intern', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'long', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'raw\_input', 'reduce', 'reload', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'unichr', 'unicode', 'vars', 'xrange', 'zip']

Start python interpreter and without doing anything type **"dir()"** without quotes and press enter, this will return objects in current scope i.e **"\_\_builtin\_\_"**, **"\_\_doc\_\_"**, **"\_\_name\_\_"** and **"\_\_package\_\_"**. This is nothing but default behavior of **dir()** which returns objects in current scope, but when you call **dir()** with an argument it will returns the attributes of the given object or in simple term the methods of respective object, as shown in above example as we have called **dir(\_\_builtins\_\_)** where **"\_\_builtins\_\_"** is an argument passed to dir will return the different attributes of the **"\_\_builtins\_\_"**.

In "Python Variable Types"(Link to Tutorial) tutorial we learned different types of variable and data types. As you are new in python programming lets try to use **dir()** function with those variable type and data types to get more information about attributes of respective object or variable or data type.

>>> x = 55

>>> dir(x)

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_class\_\_', '\_\_cmp\_\_', '\_\_coerce\_\_', '\_\_delattr\_\_', '\_\_div\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_float\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_getnewargs\_\_', '\_\_hash\_\_', '\_\_hex\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_int\_\_', '\_\_invert\_\_', '\_\_long\_\_', '\_\_lshift\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_neg\_\_', '\_\_new\_\_', '\_\_nonzero\_\_', '\_\_oct\_\_', '\_\_or\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdiv\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_ror\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', '\_\_trunc\_\_', '\_\_xor\_\_', 'bit\_length', 'conjugate', 'denominator', 'imag', 'numerator', 'real']

>>> dir(int)

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_class\_\_', '\_\_cmp\_\_', '\_\_coerce\_\_', '\_\_delattr\_\_', '\_\_div\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_float\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_getnewargs\_\_', '\_\_hash\_\_', '\_\_hex\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_int\_\_', '\_\_invert\_\_', '\_\_long\_\_', '\_\_lshift\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_neg\_\_', '\_\_new\_\_', '\_\_nonzero\_\_', '\_\_oct\_\_', '\_\_or\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdiv\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_ror\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', '\_\_trunc\_\_', '\_\_xor\_\_', 'bit\_length', 'conjugate', 'denominator', 'imag', 'numerator', 'real']

In above example **'x'** is variable assigned value 5, which is an integer, and **dir(x)** returns the various attributes of **'x'** and **'int'** is one of the object defined in **'\_\_builtins\_\_'** as we have seen in above example of **dir(\_\_builtins\_\_)**, **dir(int)** returns the same attributes as **dir(x)** because **'x'** is an integer variable.

>>> y = "Hello World!"

>>> dir(y)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_formatter\_field\_name\_split', '\_formatter\_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

>>> name\_list = ["John", "Jerry", "Jimmy", "Jackie"]

>>> dir(name\_list)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_delslice\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_iadd\_\_', '\_\_imul\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_setslice\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

>>> name\_tuple = ("John", "55", "Jerry", "5", "Jimmy", "545")

>>> dir(name\_tuple)

['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_getslice\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'count', 'index']

>>> name\_dict = {"John": 55, "Jerry": 545, "Jimmy": 999}

>>> dir(name\_dict)

['\_\_class\_\_', '\_\_cmp\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'has\_key', 'items', 'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems', 'viewkeys', 'viewvalues']

Above examples shows the list of attributes of data types of python i.e String, List, Tuple and Dictionary.

**Python help() function:**

Help function, name it self defines the usage of this function. This function returns the help related to python module, object  or method if it is called with rrespective argument but without any argument it will return the help related to currently running programming module.

Start new fresh python interpreter, type help() and press enter. It will show you python inbuilt help utility, where you can get python built in documentation about the object, methods, and attributes. as shown in examples below.

>>> help()

Welcome to Python 2.7! This is the online help utility.

If this is your first time using Python, you should definitely check out

the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing

Python programs and using Python modules. To quit this help utility and

return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",

"keywords", or "topics". Each module also comes with a one-line summary

of what it does; to list the modules whose summaries contain a given word

such as "spam", type "modules spam".

help> int

Help on class int in module \_\_builtin\_\_:

class int(object)

| int(x=0) -> int or long

| int(x, base=10) -> int or long

|

| Convert a number or string to an integer, or return 0 if no arguments

| are given. If x is floating point, the conversion truncates towards zero.

| If x is outside the integer range, the function returns a long instead.

|

| If x is not a number or if base is given, then x must be a string or

| Unicode object representing an integer literal in the given base. The

| literal can be preceded by '+' or '-' and be surrounded by whitespace.

| The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to

| interpret the base from the string as an integer literal.

As shown in above example by typing keywords, modules, or topic name help utility will return detailed documentation of the same. There is another way to get documentation using help function is passing argument to help function as shown in example below;

>>> help(str)

Help on class str in module \_\_builtin\_\_:

class str(basestring)

| str(object='') -> string

|

| Return a nice string representation of the object.

| If the argument is a string, the return value is the same object.

|

| Method resolution order:

| str

| basestring

| object

|

**Python type() function:**

During programming session we deal with some objects, modules or complex data types but we need to know the type of an object is ? so, in this case type() functions is very useful. Type-checking is convenient, easy to implement, and can save your life when debugging. Lets try few examples;

>>> x = 55

>>> type(x)

<type 'int'>

>>> y = "Hello World!"

>>> type(y)

<type 'str'>

>>> name\_list = ["John", "Jerry", "Jimmy", "Jackie"]

>>> type(name\_list)

<type 'list'>

>>> name\_tuple = ("John", "55", "Jerry", "5", "Jimmy", "545")

>>> type(name\_tuple)

<type 'tuple'>

>>> name\_dict = {"John": 55, "Jerry": 545, "Jimmy": 999}

>>> type(name\_dict)

<type 'dict'>

>>> type(\_\_builtins\_\_)

<type 'module'>

>>> def My\_String():

... print "Hello World!"

...

>>> type(My\_String)

<type 'function'>

**Python \_\_doc\_\_ function:**

Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods. An object's docstring is defined by including a string constant as the first statement in the object's definition. It's specified in source code that is used, like a comment, to document a specific part of code. Unlike conventional source code comments the docstring should describe what the function does. All functions should have a docstring. This allows the program to inspect these comments at run time, for instance as an interactive help system, or as metadata.

Docstrings can be accessed by the \_\_doc\_\_ attribute on objects.

Following are the examples of using docstring and how to specify docstring for function;

>>> int

<type 'int'>

>>> print int.\_\_doc\_\_

int(x=0) -> int or long

int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments

are given. If x is floating point, the conversion truncates towards zero.

If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or

Unicode object representing an integer literal in the given base. The

literal can be preceded by '+' or '-' and be surrounded by whitespace.

The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to

interpret the base from the string as an integer literal.

>>> int('0b100', base=0)

4

>>> list

<type 'list'>

>>> print list.\_\_doc\_\_

list() -> new empty list

list(iterable) -> new list initialized from iterable's items

>>> dict

<type 'dict'>

>>> print dict.\_\_doc\_\_

dict() -> new empty dictionary

dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:

d = {}

for k, v in iterable:

d[k] = v

dict(\*\*kwargs) -> new dictionary initialized with the name=value pairs

in the keyword argument list. For example: dict(one=1, two=2)

>>> def My\_String():

... """

... This Function Prints Hello World.

... """

... print "Hello World!"

...

>>> print My\_String.\_\_doc\_\_

This Function Prints Hello World.

As we have seen in above examples **\_\_doc\_\_** function returns the docstring of respective objects defines what it does, and we have defined function called "My\_String()" with docstring which helps user to understand what this function does.

In this tutorial we have seen four built in methods of python to get document or help on specific object, module or function. Following example shows the usage of combination of this four functions to get detailed information about python's builtin module "os".

>>> import os

>>> #using dir() with os module

...

>>> dir(os)

['EX\_CANTCREAT', 'EX\_CONFIG', 'EX\_DATAERR', 'EX\_IOERR', 'EX\_NOHOST', 'EX\_NOINPUT', 'EX\_NOPERM', 'EX\_NOUSER', 'EX\_OK', 'EX\_OSERR', 'EX\_OSFILE', 'EX\_PROTOCOL', 'EX\_SOFTWARE', 'EX\_TEMPFAIL', 'EX\_UNAVAILABLE', 'EX\_USAGE', 'F\_OK', 'NGROUPS\_MAX', 'O\_APPEND', 'O\_ASYNC', 'O\_CREAT', 'O\_DIRECT', 'O\_DIRECTORY', 'O\_DSYNC', 'O\_EXCL', 'O\_LARGEFILE', 'O\_NDELAY', 'O\_NOATIME', 'O\_NOCTTY', 'O\_NOFOLLOW', 'O\_NONBLOCK', 'O\_RDONLY', 'O\_RDWR', 'O\_RSYNC', 'O\_SYNC', 'O\_TRUNC', 'O\_WRONLY', 'P\_NOWAIT', 'P\_NOWAITO', 'P\_WAIT', 'R\_OK', 'SEEK\_CUR', 'SEEK\_END', 'SEEK\_SET', 'ST\_APPEND', 'ST\_MANDLOCK', 'ST\_NOATIME', 'ST\_NODEV', 'ST\_NODIRATIME', 'ST\_NOEXEC', 'ST\_NOSUID', 'ST\_RDONLY', 'ST\_RELATIME', 'ST\_SYNCHRONOUS', 'ST\_WRITE', 'TMP\_MAX', 'UserDict', 'WCONTINUED', 'WCOREDUMP', 'WEXITSTATUS', 'WIFCONTINUED', 'WIFEXITED', 'WIFSIGNALED', 'WIFSTOPPED', 'WNOHANG', 'WSTOPSIG', 'WTERMSIG', 'WUNTRACED', 'W\_OK', 'X\_OK', '\_Environ', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_copy\_reg', '\_execvpe', '\_exists', '\_exit', '\_get\_exports\_list', '\_make\_stat\_result', '\_make\_statvfs\_result', '\_pickle\_stat\_result', '\_pickle\_statvfs\_result', '\_spawnvef', 'abort', 'access', 'altsep', 'chdir', 'chmod', 'chown', 'chroot', 'close', 'closerange', 'confstr', 'confstr\_names', 'ctermid', 'curdir', 'defpath', 'devnull', 'dup', 'dup2', 'environ', 'errno', 'error', 'execl', 'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fchdir', 'fchmod', 'fchown', 'fdatasync', 'fdopen', 'fork', 'forkpty', 'fpathconf', 'fstat', 'fstatvfs', 'fsync', 'ftruncate', 'getcwd', 'getcwdu', 'getegid', 'getenv', 'geteuid', 'getgid', 'getgroups', 'getloadavg', 'getlogin', 'getpgid', 'getpgrp', 'getpid', 'getppid', 'getresgid', 'getresuid', 'getsid', 'getuid', 'initgroups', 'isatty', 'kill', 'killpg', 'lchown', 'linesep', 'link', 'listdir', 'lseek', 'lstat', 'major', 'makedev', 'makedirs', 'minor', 'mkdir', 'mkfifo', 'mknod', 'name', 'nice', 'open', 'openpty', 'pardir', 'path', 'pathconf', 'pathconf\_names', 'pathsep', 'pipe', 'popen', 'popen2', 'popen3', 'popen4', 'putenv', 'read', 'readlink', 'remove', 'removedirs', 'rename', 'renames', 'rmdir', 'sep', 'setegid', 'seteuid', 'setgid', 'setgroups', 'setpgid', 'setpgrp', 'setregid', 'setresgid', 'setresuid', 'setreuid', 'setsid', 'setuid', 'spawnl', 'spawnle', 'spawnlp', 'spawnlpe', 'spawnv', 'spawnve', 'spawnvp', 'spawnvpe', 'stat', 'stat\_float\_times', 'stat\_result', 'statvfs', 'statvfs\_result', 'strerror', 'symlink', 'sys', 'sysconf', 'sysconf\_names', 'system', 'tcgetpgrp', 'tcsetpgrp', 'tempnam', 'times', 'tmpfile', 'tmpnam', 'ttyname', 'umask', 'uname', 'unlink', 'unsetenv', 'urandom', 'utime', 'wait', 'wait3', 'wait4', 'waitpid', 'walk', 'write']

>>> # using help() with os module

...

>>> help(os)

Help on module os:

NAME

os - OS routines for Mac, NT, or Posix depending on what system we're on.

FILE

/usr/lib/python2.7/os.py

MODULE DOCS

http://docs.python.org/library/os

DESCRIPTION

This exports:

- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.

- os.path is one of the modules posixpath, or ntpath

- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'

- os.curdir is a string representing the current directory ('.' or ':')

- os.pardir is a string representing the parent directory ('..' or '::')

- os.sep is the (or a most common) pathname separator ('/' or ':' or '\')

- os.extsep is the extension separator ('.' or '/')

- os.altsep is the alternate pathname separator (None or '/')

- os.pathsep is the component separator used in $PATH etc

- os.linesep is the line separator in text files ('r' or 'n' or 'rn')

- os.defpath is the default search path for executables

- os.devnull is the file path of the null device ('/dev/null', etc.)

Programs that import and use 'os' stand a better chance of being

portable between different platforms. Of course, they must then

only use functions that are defined by all platforms (e.g., unlink

and opendir), and leave all pathname manipulation to os.path

(e.g., split and join).

>>> # using type() with os module

...

>>> type(os)

<type 'module'>

>>> # using \_\_doc\_\_ with os module

...

>>> print os.\_\_doc\_\_

OS routines for Mac, NT, or Posix depending on what system we're on.

This exports:

- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.

- os.path is one of the modules posixpath, or ntpath

- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'

- os.curdir is a string representing the current directory ('.' or ':')

- os.pardir is a string representing the parent directory ('..' or '::')

- os.sep is the (or a most common) pathname separator ('/' or ':' or '\')

- os.extsep is the extension separator ('.' or '/')

- os.altsep is the alternate pathname separator (None or '/')

- os.pathsep is the component separator used in $PATH etc

- os.linesep is the line separator in text files ('r' or 'n' or 'rn')

- os.defpath is the default search path for executables

- os.devnull is the file path of the null device ('/dev/null', etc.)

Programs that import and use 'os' stand a better chance of being

portable between different platforms. Of course, they must then

only use functions that are defined by all platforms (e.g., unlink

and opendir), and leave all pathname manipulation to os.path

(e.g., split and join).

This is how you can use combination of all this to get documentation on specific object or module to solve many errors and problems during real programming session.