Python Supplement

August 22, 2024

Ramasamy Kandasamy

1. Core

`.__name__`	Name of an object, function, module, etc. Eg: `foo.__name__ # Returns 'foo'`. Eg: `f = foo` Eg: `f.__name__ # Returns 'foo'`.
`__name__`	When not an attribute, it hold the name of the script. If the script is run on its own then `__name__ == "__main__"`. Else it holds the name of the script. If the script is `foo.py` Then `__name__ == "foo"`.

`globals()`
.get('foo')
	Gives dictionary representing the global symbol table.
	Returns the object the name `foo`.

id(x)	Gives the address of the variable `x`. Can be used to check if two variables point to the same object. The address cannot be derefenced like a C pointer.
*	Unpacks a sequence like a list or tuple to function arguments. Eg: `l = [1, 2, 3]` Eg: `foo(*l)` Eg: `a, *b, c = [1, 2, 3, 4, 5] # b = [2, 3, 4]`
**	Upacks a dictionary into keyword arguments. Eg: `kwargs = 'a': 1,'b' = 2`. Eg: `foo(**kwargs)` Inside function foo `a` and `b` becomes variables with values `1` and `2` respectively.
`zip`	Zip two iterables. Eg: `x = [1, 2]`. Eg: `y = [3, 4]`. Eg: `w = list(zip(x, y))`. Eg: `w = [(1, 3), (2, 4)]`.
`enumerate`	Enumerate iterables. Eg: `for id, x in enumerate(l):`. Eg: `for id, (x, y) in enumerate(zip(l1, l2)):`.

Special Variables and Naming Conventions

`_foo`	Internal use. Not for public access.
`__foo`	Name mangling to prevent accidental overrides. Interpreter changes the name to prevent override in child class.
`__foo__`	Special variables that are part of python, don't create your own.
`foo_`	To avoid conflict with keywords.
`__file__`	Path to current script of file.
`__name__`	Name of the module or `"__main__"` if run directly.
`__version__`	Version of the module. Does not work for all (eg: os). Eg: `np.__version__`. Eg: `os.__version__`. This does not work.

TODO: Generators,

2. OS, Shutil, Sys, Etc.

os

`chdir()`	
`getcwd()`	
`listdir()`	
`mkdir()`	Create a directory at the path.
`makedirs()`	Recursively creates the directories. Like `mkdir -p` in bash.
`rmdir()`	To delete **empty** directories.
`rename()`	`os.rename(src, dst)`. Rename for files and directories.
`system()`	Execute system commands. Return the exit status. Eg: `exit_status = os.system('ls')`.

shutil

`rmtree()`	`shutil.rmtree('mydir')`.
`move()`	`shutil.move(src, des)`.
`copy()`	`shutil.copy(src, des)`. Copy single file without preserving meta data.
`copy2()`	Copy single file with preserving meta data.
`copytree()`	Copy entire directory and it's contents.

os.path

`isdir()`	
`isfile()`	
`exists()`	
`join()`	
`dirname()`	
`basename()`	
`abspath()`	Return the absolute path to a file. Eg: `os.path.abspath('foo')`.
`realpath()`	Real path Resolves any symlinks along the way.

sys

`argv`	Command line arguments as a list of strings.
`exit()`	Exit and return an exit status to the calling process.
`version`	Get python version.
`path`	Manage list of search paths for modules. `sys.path.append('foo')`. Add the directory `foo` to the path.

CAUTION!!

In `sys.path.append()`, relative path is acceptable, but it is relative to the directory from where the script is being executed, not relative to where the script file is located.

Best practice:

```
script_dir = os.path.dirname(os.path.abspath(__file__))
sys.path.append(os.path.join(script\_dir,\
relative/path/to/directory'))
```