Advanced Concepts

Ayyappan Murugesan
Phone: 7010774329
WhatsApp: 8870483557

Topics

- Class
- Modules
- Important Libraries
- File
- Standard Library
- Threads
- Socket Programming

Python Class

Ayyappan Murugesan

Phone: 7010774329

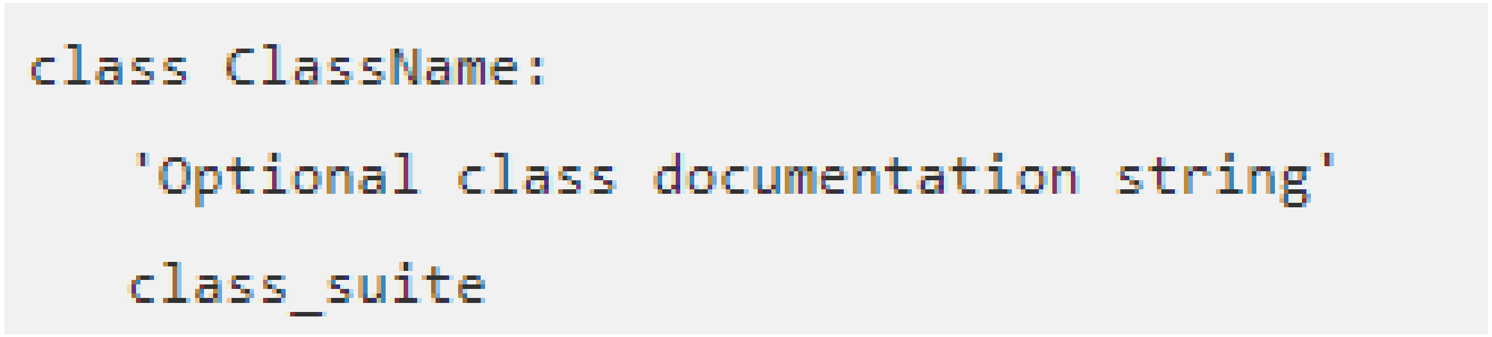
WhatsApp: 8870483557

Python Class

- A class is an arrangement of variables and functions into a single logical entity. It works as a template for creating objects.
- Every object can use class variables and functions as its members.
- The object is a working instance of a class created at runtime.

Creating class

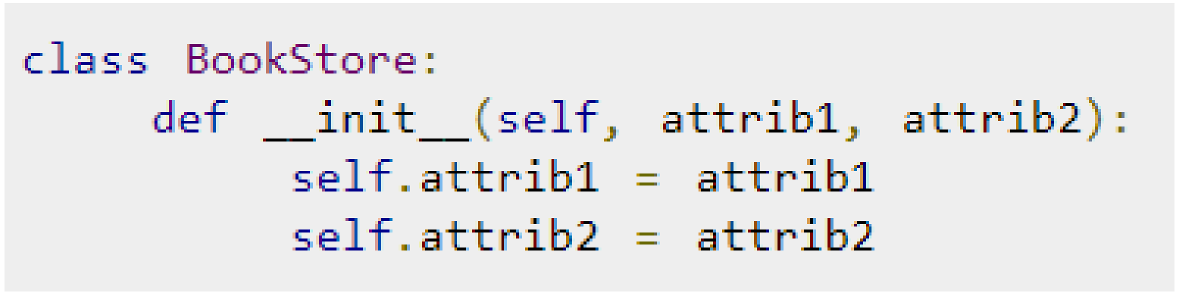
- The class statement creates a new class definition.



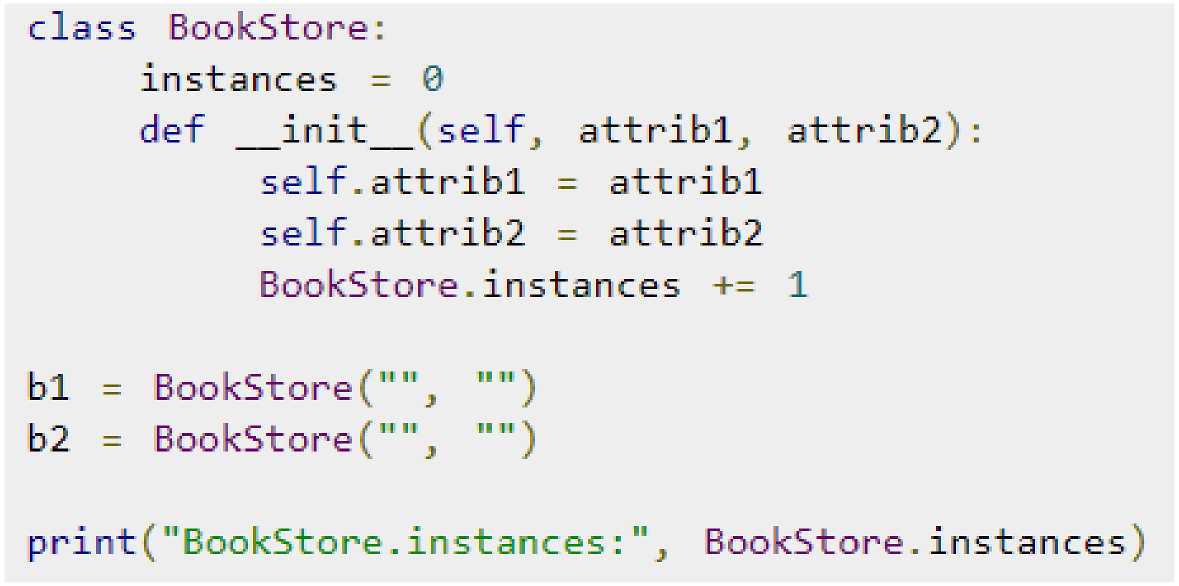
```
class ClassName:
   'Optional class documentation string'
   class_suite
```

5 Terms

- **class keyword**
- **instance attributes**
- **class attributes**
- **self keyword**
- **__init__ keyword**

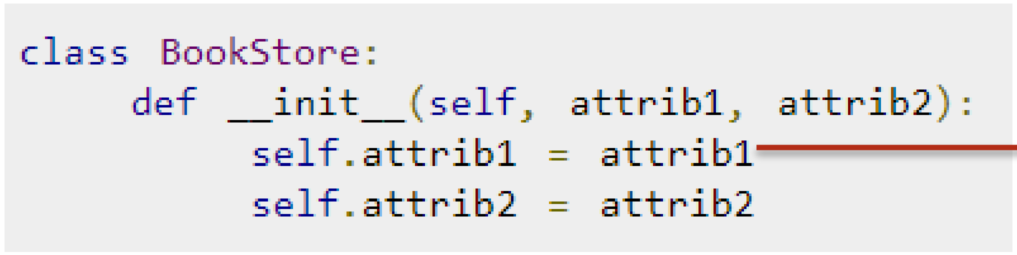


```
class BookStore:
        def __init__(self, attrib1, attrib2):
                self.attrib1 = attrib1
                self.attrib2 = attrib2
```



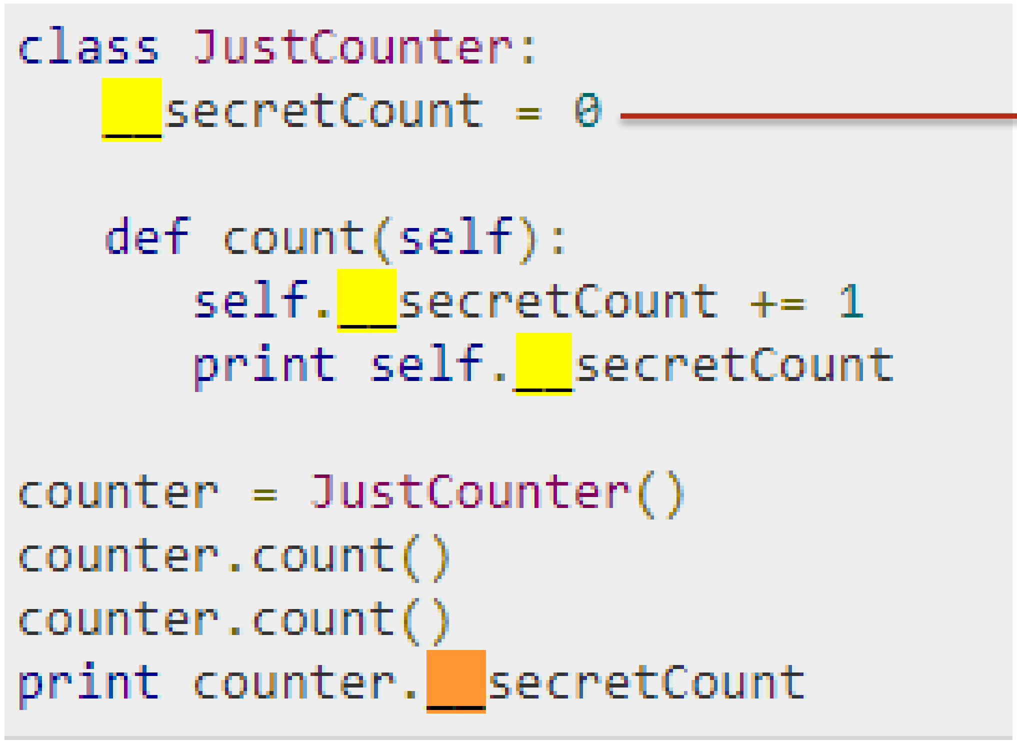
```
class BookStore:
        instances = 0
        def __init__(self, attrib1, attrib2):
                self.attrib1 = attrib1
                self.attrib2 = attrib2
                BookStore.instances += 1

b1 = BookStore("", "")
b2 = BookStore("", "")

print("BookStore.instances:", BookStore.instances)
```

PUBLIC & PRIVATE (DATA HIDING) ATTRIBUTES



```
class BookStore:
    def __init__(self, attrib1, attrib2):
        self.attrib1 = attrib1
        self.attrib2 = attrib2
```

Public

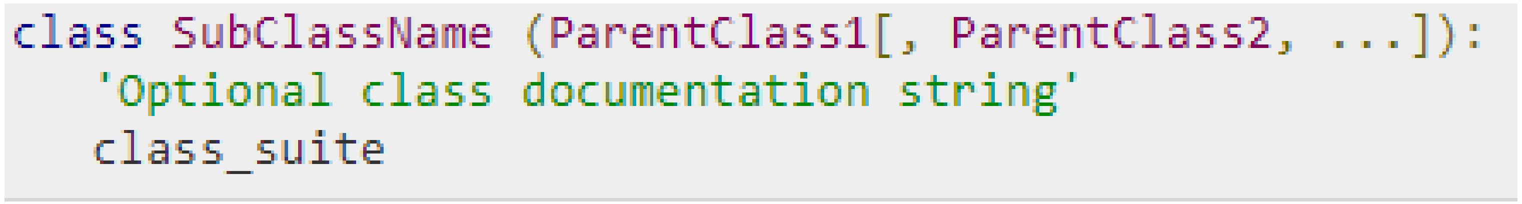


```
class JustCounter:
    __secretCount = 0

    def count(self):
        self.__secretCount += 1
        print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

Private

Class Inheritance

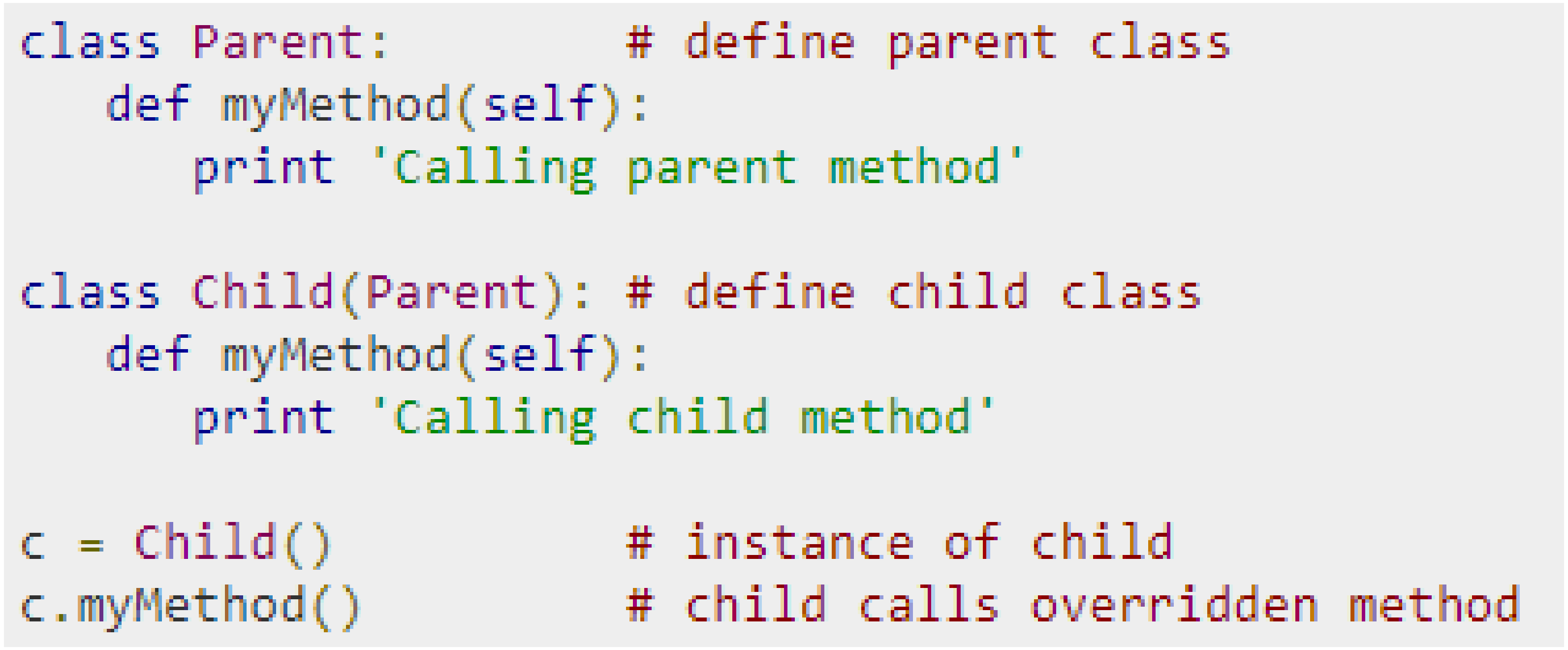
- Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.



```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Optional class documentation string'
   class_suite
```

```
class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print "Calling parent constructor"

   def parentMethod(self):
      print 'Calling parent method'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
   def __init__(self):
      print "Calling child constructor"

   def childMethod(self):
      print 'Calling child method'

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

Overriding Methods

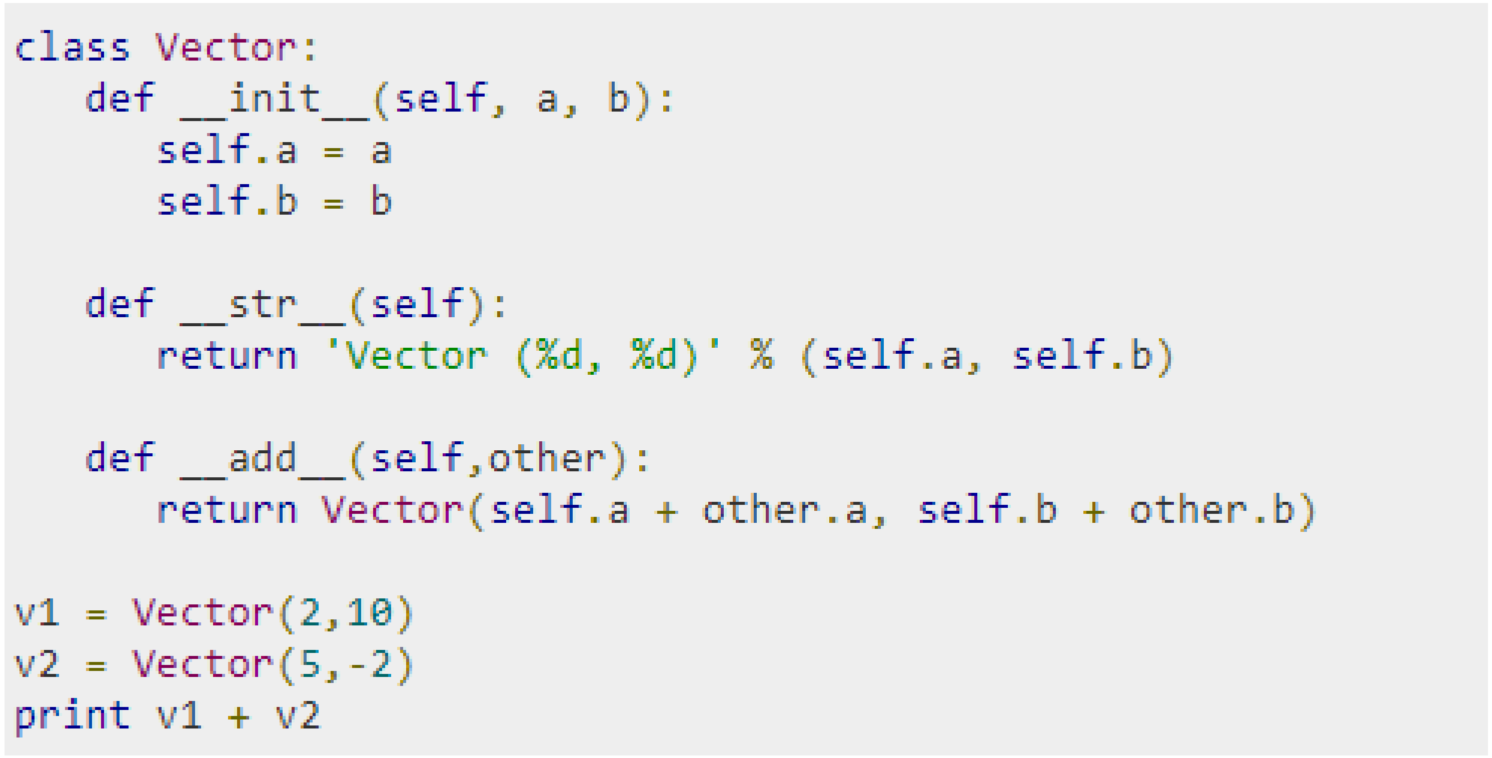


```
class Parent:        # define parent class
   def myMethod(self):
      print 'Calling parent method'

class Child(Parent): # define child class
   def myMethod(self):
      print 'Calling child method'

c = Child()          # instance of child
c.myMethod()         # child calls overridden method
```

Base Overriding Methods

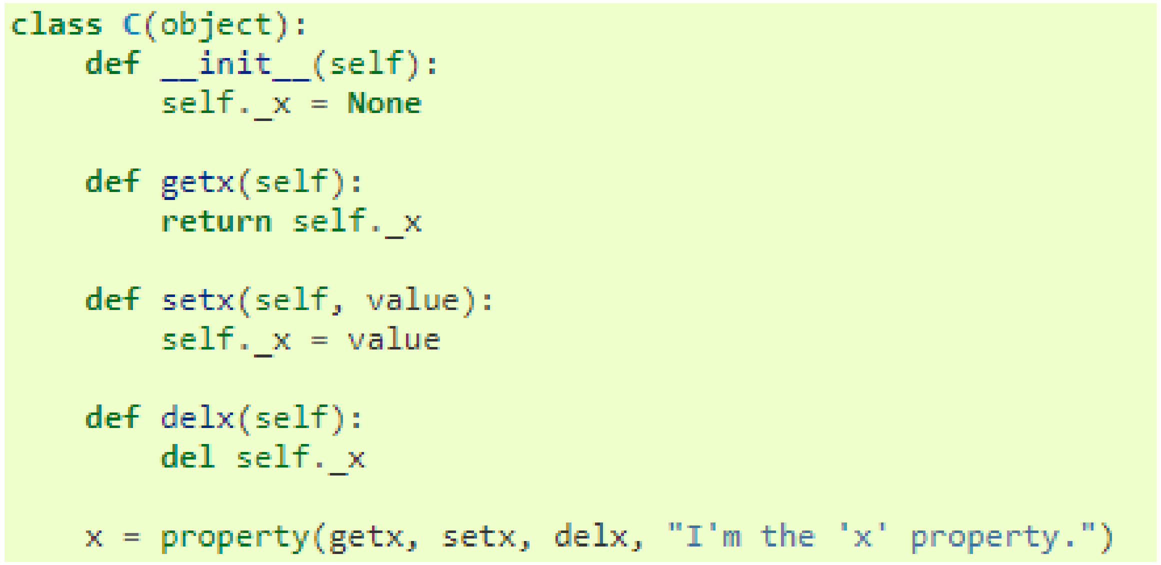
- __init__ (self [,args...])
- __del__(self)
- __repr__(self)
- __str__(self)

Overloading Operators

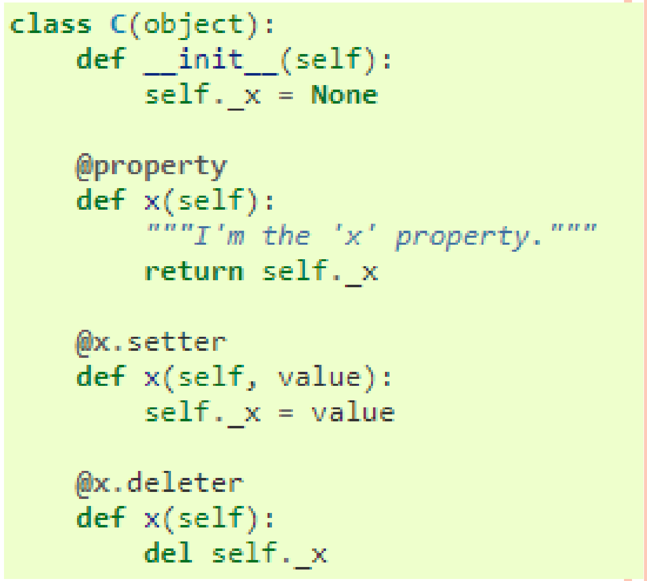


```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

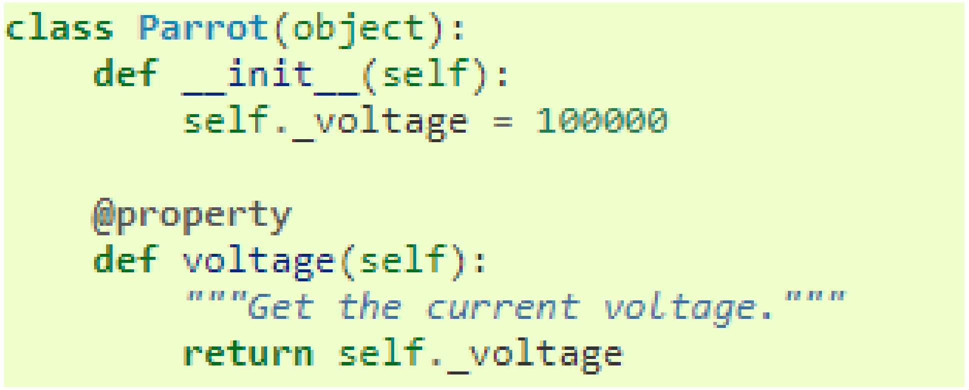
Getter , setter & Deleter



```python
class C(object):
    def __init__(self):
        self._x = None

    def getx(self):
        return self._x

    def setx(self, value):
        self._x = value

    def delx(self):
        del self._x

    x = property(getx, setx, delx, "I'm the 'x' property.")
```



```python
class C(object):
    def __init__(self):
        self._x = None

    @property
    def x(self):
        """I'm the 'x' property."""
        return self._x

    @x.setter
    def x(self, value):
        self._x = value

    @x.deleter
    def x(self):
        del self._x
```



```python
class Parrot(object):
    def __init__(self):
        self._voltage = 100000

    @property
    def voltage(self):
        """Get the current voltage."""
        return self._voltage
```

Class Method & Static Method

- A class method receives the class as its first argument.
- classmethod() or with decorator @classmethod.

- A static method receives neither the instance nor the class as its first argument
- staticmethod() or with decorator @staticmethod

```
class B(object):

    Count = 0

    def dup_string(x):
        s1 = '%s%s' % (x, x,)
        return s1
    dup_string = staticmethod(dup_string)

    @classmethod
    def show_count(cls, msg):
        print '%s  %d' % (msg, cls.Count, )

def test():
    print B.dup_string('abcd')
    B.show_count('here is the count: ')
```

New-style classes

```
class C(list):
  def get_len(self):
    return len(self)

c = C((11,22,33))
c.get_len()

c = C((11,22,33,44,55,66,77,88))
print c.get_len()
# Prints "8".
```

```
class D(dict):
    def __init__(self, data=None, name='no_name'):
        if data is None:
            data = {}
        dict.__init__(self, data)
        self.name = name
    def get_len(self):
        return len(self)
    def get_keys(self):
        content = []
        for key in self:
            content.append(key)
        contentstr = ', '.join(content)
        return contentstr
    def get_name(self):
        return self.name

def test():
    d = D({'aa': 111, 'bb':222, 'cc':333})
    # Prints "3"
    print d.get_len()
    # Prints "'aa, cc, bb'"
    print d.get_keys()
    # Prints "no_name"
    print d.get_name()
```

Import the class from file

```
from folder.file import Klasa
```

Or, with `from folder import file`:

```
from folder import file
k = file.Klasa()
```

Or again:

```
import folder.file as myModule
k = myModule.Klasa()
```

Modules

Ayyappan Murugesan
Phone: 7010774329
WhatsApp: 8870483557

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 file consisting of Python code

Locating Modules

- The current directory (sys.path)
- 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/. On Window, this default path is normally <Install directory>/lib

Asset Functions

Function	Description
dir( )	The list contains the names of all the modules, variables and functions that are defined
globals()	return all the names that can be accessed
locals()	return all the names that can be accessed locally
reload()	if you want to reexecute the top-level code in a module, you can use the *reload()* function

Packages in Python

- A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages

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

MyLib/Pots.py
MyLib/Pots1.py
MyLib/Pots2.py

MyLib/__init__.py

```
from Pots import Pots
from Pots1 import Pots1
from Pots2 import Pots2
```

```
import MyLib
MyLib.Pots()
MyLib.Pots1()
MyLib.Pots2()
```

Important library

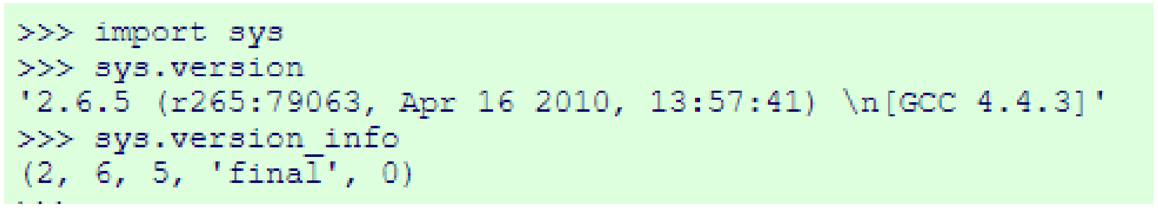
Ayyappan Murugesan

Phone: 7010774329

WhatsApp: 8870483557

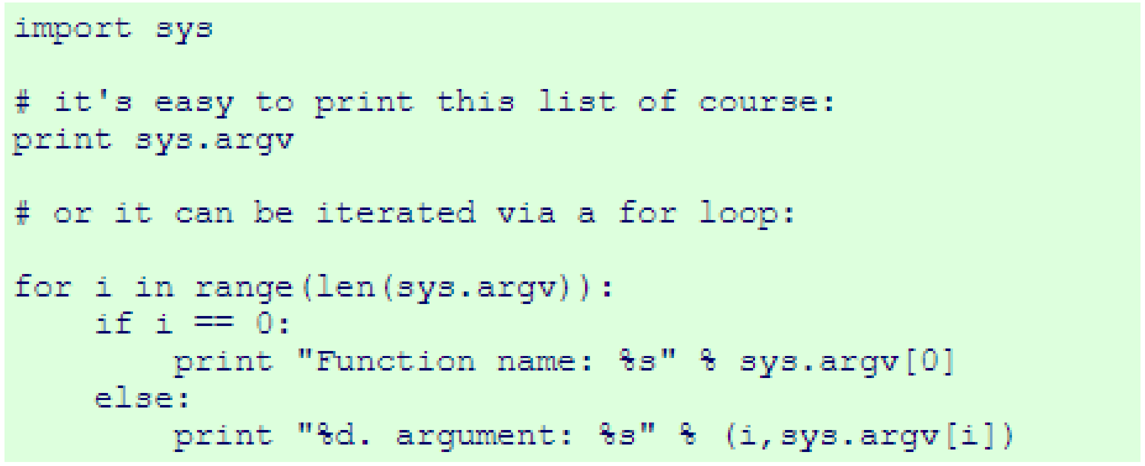
SYS

- System-specific parameters and functions



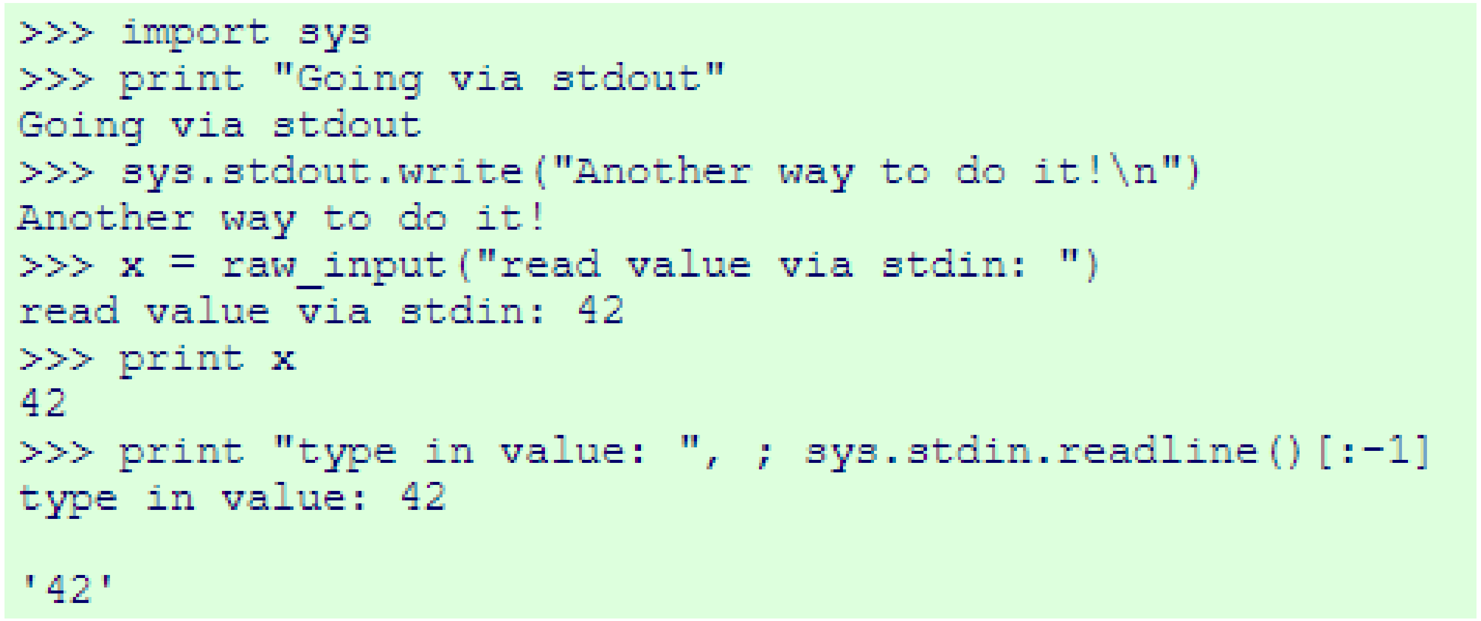
```
>>> import sys
>>> sys.version
'2.6.5 (r265:79063, Apr 16 2010, 13:57:41) \n[GCC 4.4.3]'
>>> sys.version_info
(2, 6, 5, 'final', 0)
```

- Command Line Argument



```
import sys

# it's easy to print this list of course:
print sys.argv

# or it can be iterated via a for loop:

for i in range(len(sys.argv)):
    if i == 0:
        print "Function name: %s" % sys.argv[0]
    else:
        print "%d. argument: %s" % (i,sys.argv[i])
```

- Standard data streams



```
>>> import sys
>>> print "Going via stdout"
Going via stdout
>>> sys.stdout.write("Another way to do it!\n")
Another way to do it!
>>> x = raw_input("read value via stdin: ")
read value via stdin: 42
>>> print x
42
>>> print "type in value: ", ; sys.stdin.readline()[:-1]
type in value: 42

'42'
```

OS

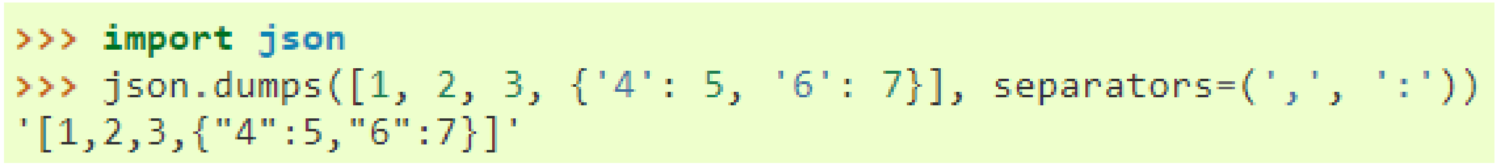
- The OS module in Python provides a way of using operating system dependent functionality.

Function	Description
os.system()	Executing a shell command
os.environ()	Get the users environment
os.getcwd()	Returns the current working directory.
os.getgid()	Return the real group id of the current process.
os.getuid()	Return the current process's user id
os.getpid()	Returns the real process ID of the current process
os.umask(mask)	Set the current numeric umask and return the previous umask
os.uname()	Return information identifying the current operating system.
os.chroot(path)	Change the root directory of the current process to path
os.listdir(path)	Return a list of the entries in the directory given by path
os.mkdir(path)	Create a directory named path with numeric mode mode

Function	Description
os.makedirs(path)	Recursive directory creation function
os.remove(path)	Remove (delete) the file path.
os.removedirs(path)	Remove directories recursively.
os.rename(src, dst)	Rename the file or directory src to dst.
os.rmdir(path)	Remove (delete) the directory path.

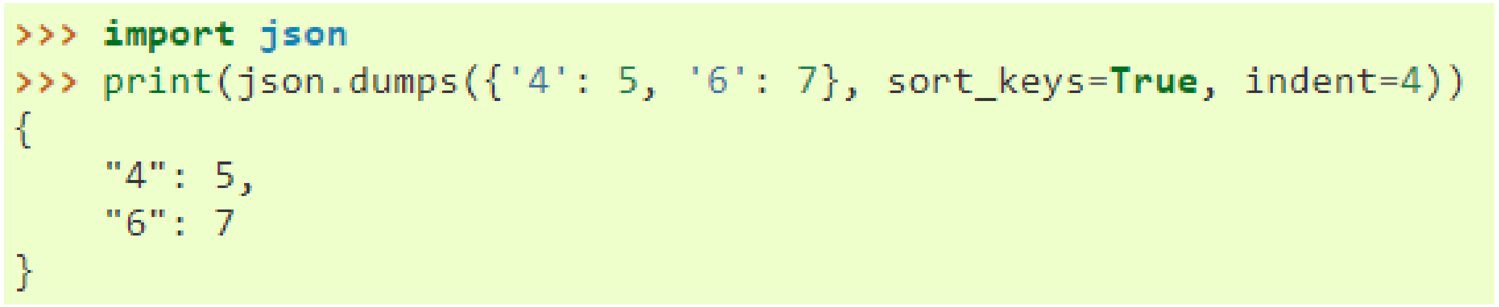
JSON

- Compact encoding



```
>>> import json
>>> json.dumps([1, 2, 3, {'4': 5, '6': 7}], separators=(',', ':'))
'[1,2,3,{"4":5,"6":7}]'
```

- Pretty printing



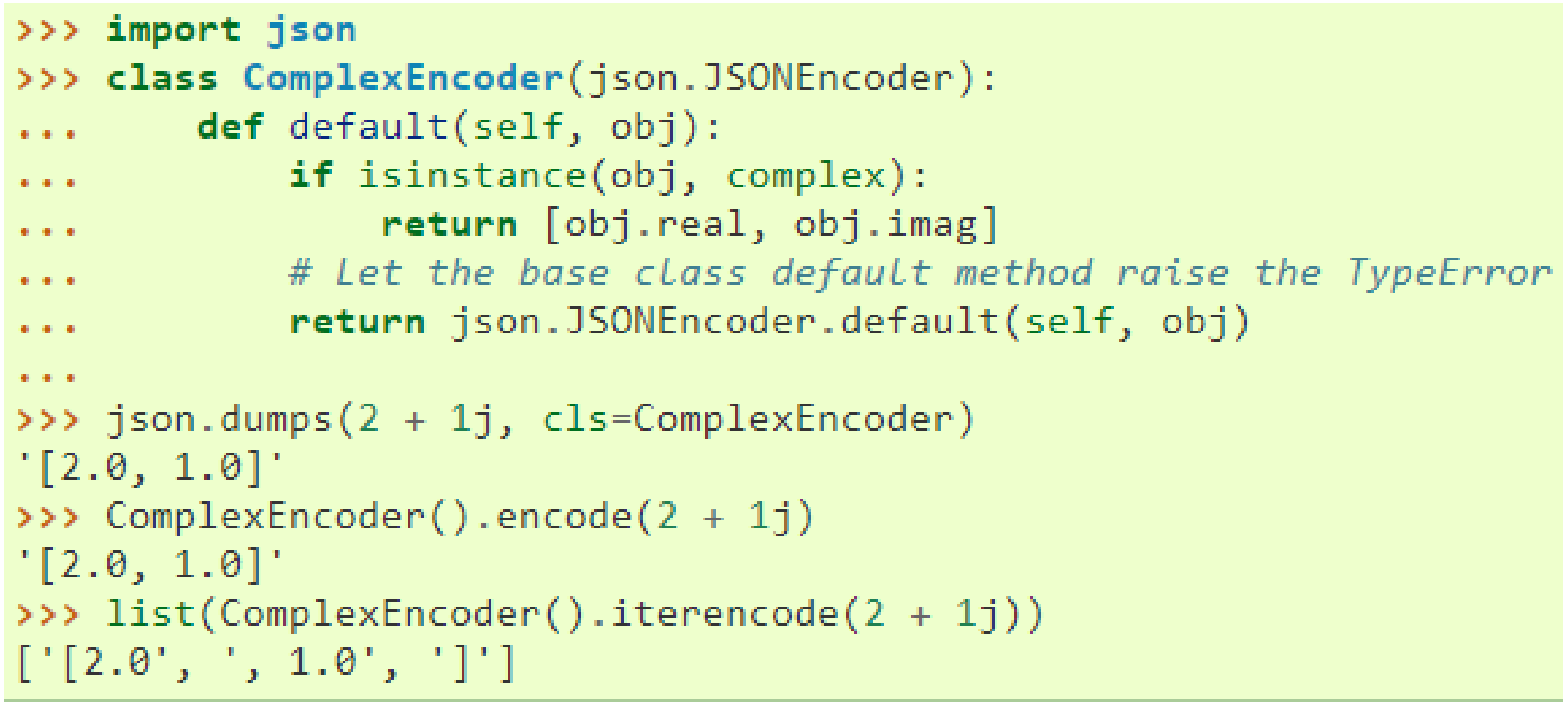
```
>>> import json
>>> print(json.dumps({'4': 5, '6': 7}, sort_keys=True, indent=4))
{
    "4": 5,
    "6": 7
}
```

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

Decoding JSON

```
>>> import json
>>> json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')
['foo', {'bar': ['baz', None, 1.0, 2]}]
>>> json.loads('"\\"foo\\bar"')
'"foo\x08ar'
>>> from io import StringIO
>>> io = StringIO('["streaming API"]')
>>> json.load(io)
['streaming API']
```

```
>>> import json
>>> def as_complex(dct):
...     if '__complex__' in dct:
...         return complex(dct['real'], dct['imag'])
...     return dct
...
>>> json.loads('{"__complex__": true, "real": 1, "imag": 2}',
...     object_hook=as_complex)
(1+2j)
>>> import decimal
>>> json.loads('1.1', parse_float=decimal.Decimal)
Decimal('1.1')
```

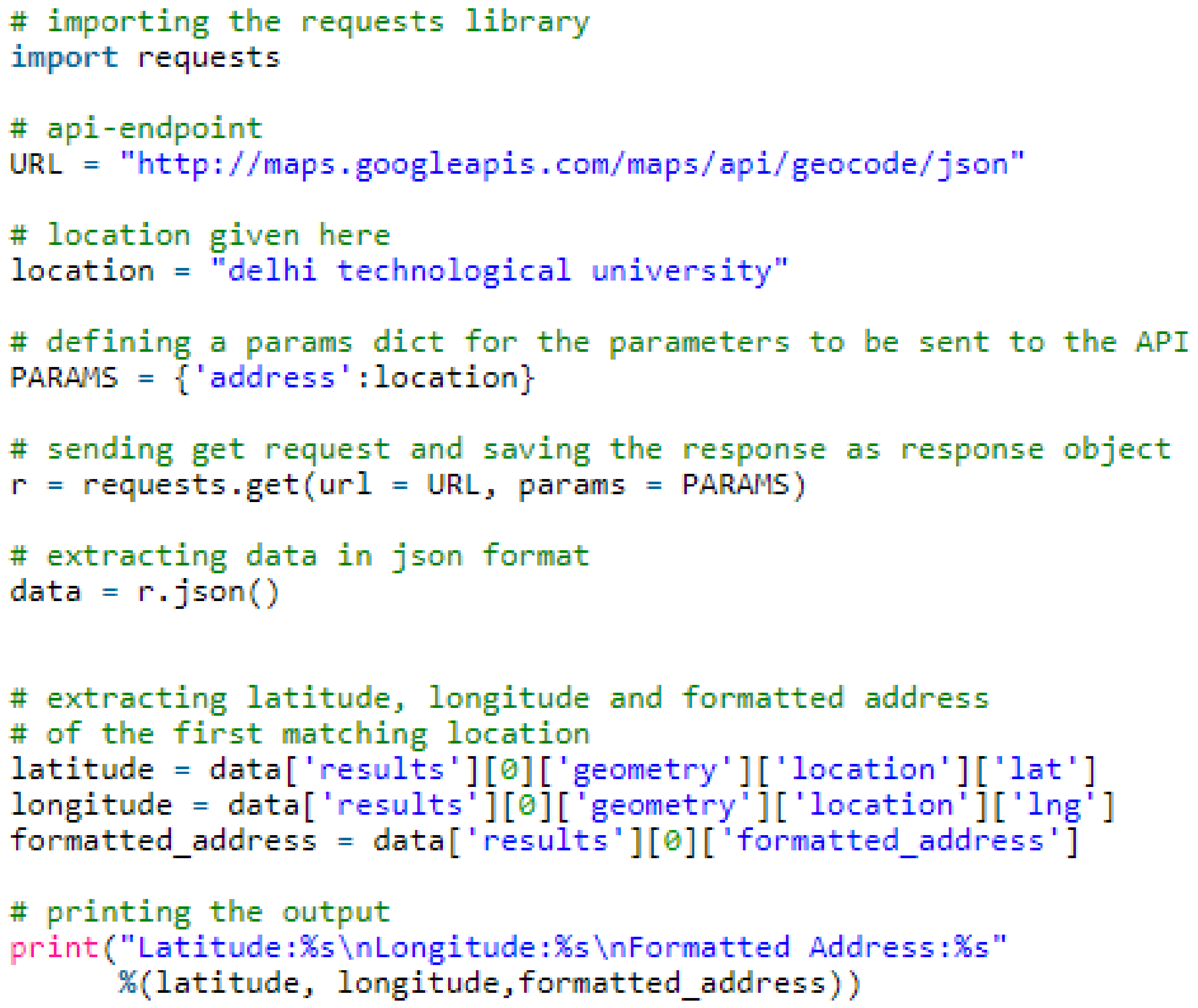


```
>>> import json
>>> class ComplexEncoder(json.JSONEncoder):
...     def default(self, obj):
...         if isinstance(obj, complex):
...             return [obj.real, obj.imag]
...         # Let the base class default method raise the TypeError
...         return json.JSONEncoder.default(self, obj)
...
>>> json.dumps(2 + 1j, cls=ComplexEncoder)
'[2.0, 1.0]'
>>> ComplexEncoder().encode(2 + 1j)
'[2.0, 1.0]'
>>> list(ComplexEncoder().iterencode(2 + 1j))
['[2.0', ', 1.0', ']']
```

HTTP (REQUESTS)

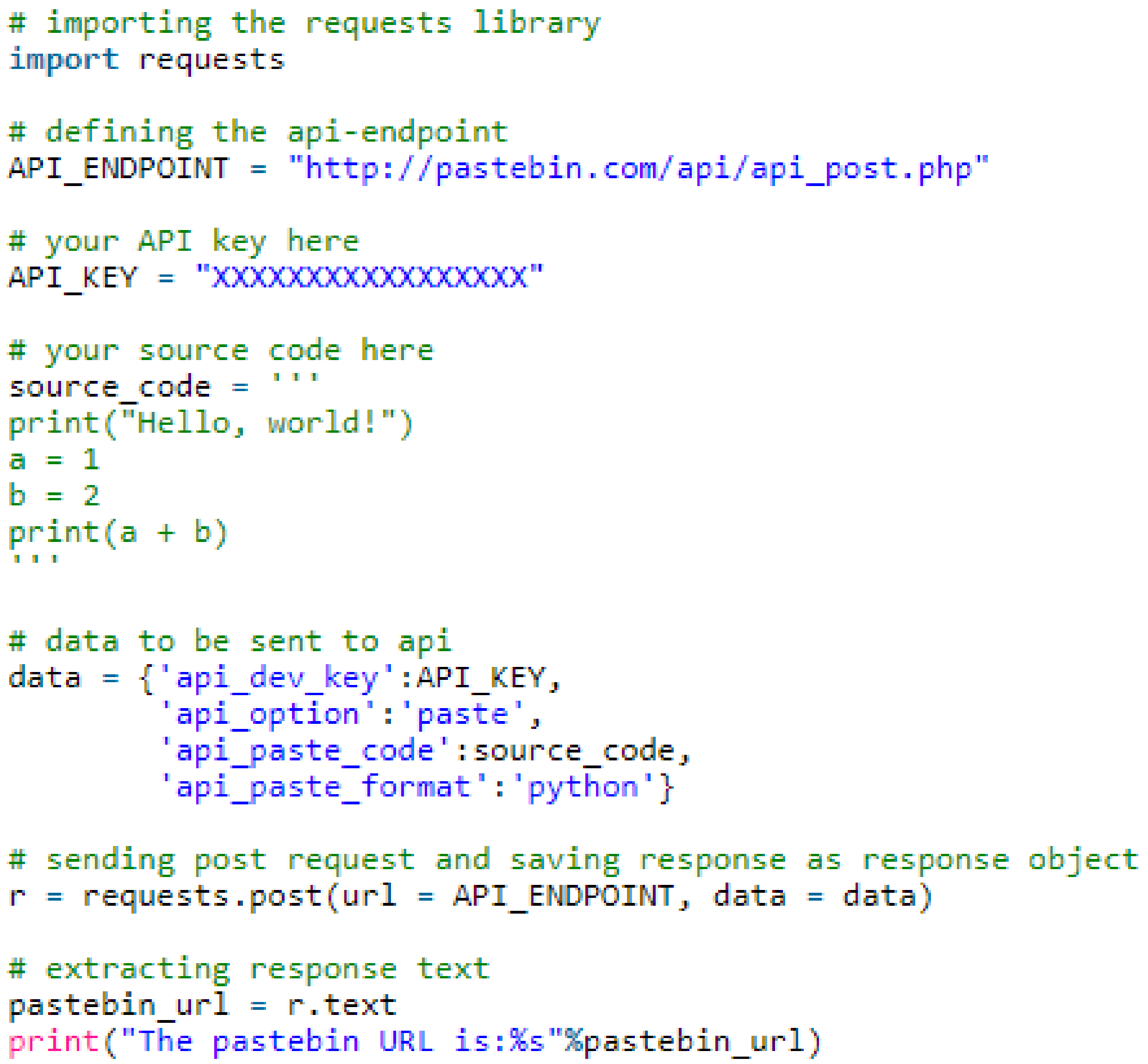
- HTTP is a set of protocols designed to enable communication between clients and servers. It works as a request-response protocol between a client and server
- HTTP Method
  - GET
  - POST

HTTP GET



```python
# importing the requests library
import requests

# api-endpoint
URL = "http://maps.googleapis.com/maps/api/geocode/json"

# location given here
location = "delhi technological university"

# defining a params dict for the parameters to be sent to the API
PARAMS = {'address':location}

# sending get request and saving the response as response object
r = requests.get(url = URL, params = PARAMS)

# extracting data in json format
data = r.json()


# extracting latitude, longitude and formatted address
# of the first matching location
latitude = data['results'][0]['geometry']['location']['lat']
longitude = data['results'][0]['geometry']['location']['lng']
formatted_address = data['results'][0]['formatted_address']

# printing the output
print("Latitude:%s\nLongitude:%s\nFormatted Address:%s"
      %(latitude, longitude,formatted_address))
```

HTTP POST



```python
# importing the requests library
import requests

# defining the api-endpoint
API_ENDPOINT = "http://pastebin.com/api/api_post.php"

# your API key here
API_KEY = "XXXXXXXXXXXXXXXXX"

# your source code here
source_code = '''
print("Hello, world!")
a = 1
b = 2
print(a + b)
'''

# data to be sent to api
data = {'api_dev_key':API_KEY,
        'api_option':'paste',
        'api_paste_code':source_code,
        'api_paste_format':'python'}

# sending post request and saving response as response object
r = requests.post(url = API_ENDPOINT, data = data)

# extracting response text
pastebin_url = r.text
print("The pastebin URL is:%s"%pastebin_url)
```

```
import json
url = 'https://api.github.com/some/endpoint'
payload = {'some': 'data'}
headers = {'content-type': 'application/json'}

r = requests.post(url, data=json.dumps(payload), headers=headers)
```

```
import requests, json

github_url = "https://api.github.com/user/repos"
data = json.dumps({'name':'test', 'description':'some test repo'})
r = requests.post(github_url, data, auth=('user', '*****'))

print r.json
```

```
r = requests.put("http://httpbin.org/put")
r = requests.delete("http://httpbin.org/delete")
r = requests.head("http://httpbin.org/get")
r = requests.options("http://httpbin.org/get")
```

Files

Ayyappan Murugesan
Phone: 7010774329
WhatsApp: 8870483557

OPEN

```
file object = open(file_name [, access_mode][, buffering])
```

Below are the parameter details.

<access_mode>- It's an integer representing the file open mode e.g. read, write, append, etc. It's an optional parameter. By default, it is set to read-only <r>. In this mode, we get data in text form after reading from the file. On the other hand, binary mode returns bytes. It's preferable for accessing the non-text files like an image or the Exe files. See the table in the next section. It lists down the available access modes.

<buffering>- The default value is 0 which means buffering won't happen. If the value is 1, then line buffering will take place while accessing the file. If it's greater than 1, then the buffering action will run as per the buffer size. In the case of a negative value, the default behavior is considered.

<file_name>- It's a string representing the name of the file you want to access.

Modes	Description
<r>	It opens a file in read-only mode while the file offset stays at the root.
<rb>	It opens a file in (binary + read-only) modes. And the offset remains at the root level.
<r+>	It opens the file in both (read + write) modes while the file offset is again at the root level.
<rb+>	It opens the file in (read + write + binary) modes. The file offset is again at the root level.
<w>	It allows write-level access to a file. If the file already exists, then it'll get overwritten. It'll create a new file if the same doesn't exist.
<wb>	Use it to open a file for writing in binary format. Same behavior as for write-only mode.
<w+>	It opens a file in both (read + write) modes. Same behavior as for write-only mode.
<wb+>	It opens a file in (read + write + binary) modes. Same behavior as for write-only mode.
<a>	It opens the file in append mode. The offset goes to the end of the file. If the file doesn't exist, then it gets created.
<ab>	It opens a file in (append + binary) modes. Same behavior as for append mode.
<a+>	It opens a file in (append + read) modes. Same behavior as for append mode.
<ab+>	It opens a file in (append + read + binary) modes. Same behavior as for append mode.

```
#Open a file in write and binary mode.
fob = open("app.log", "wb")

#Display file name.
print "File name: ", fob.name
#Display state of the file.
print "File state: ", fob.closed
#Print the opening mode.
print "Opening mode: ", fob.mode
#Output the softspace value.
print "Softspace flag: ", fob.softspace
```

```
Python 2.7.10
[GCC 4.8.2] on Linux

File name:   app.log
File state:   False
Opening mode:   wb
Softspace flag:   0
```

- Basic Close operation

```
f = open("app.log",encoding = 'utf-8')
# do file operations.
f.close()
```

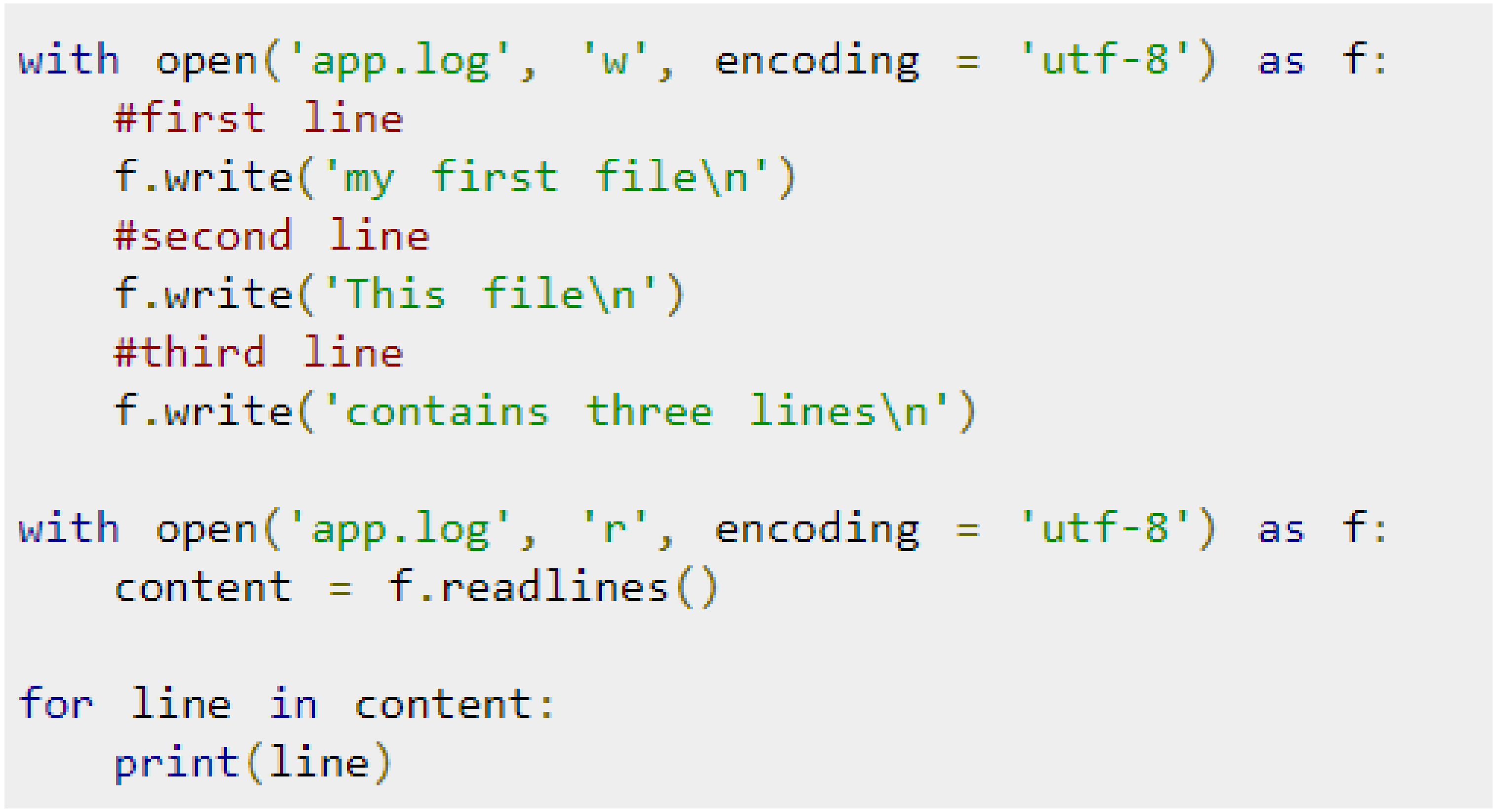
- Using Exception with Close

```
try:
    f = open('app.log', encoding = 'utf-8')
    # do file operations.
finally:
    f.close()
```

- Close A File Using ‘With’ Clause In Python

```
with open('app.log', encoding = 'utf-8') as f:
    #do any file operation.
```

Write & Read (WITH)

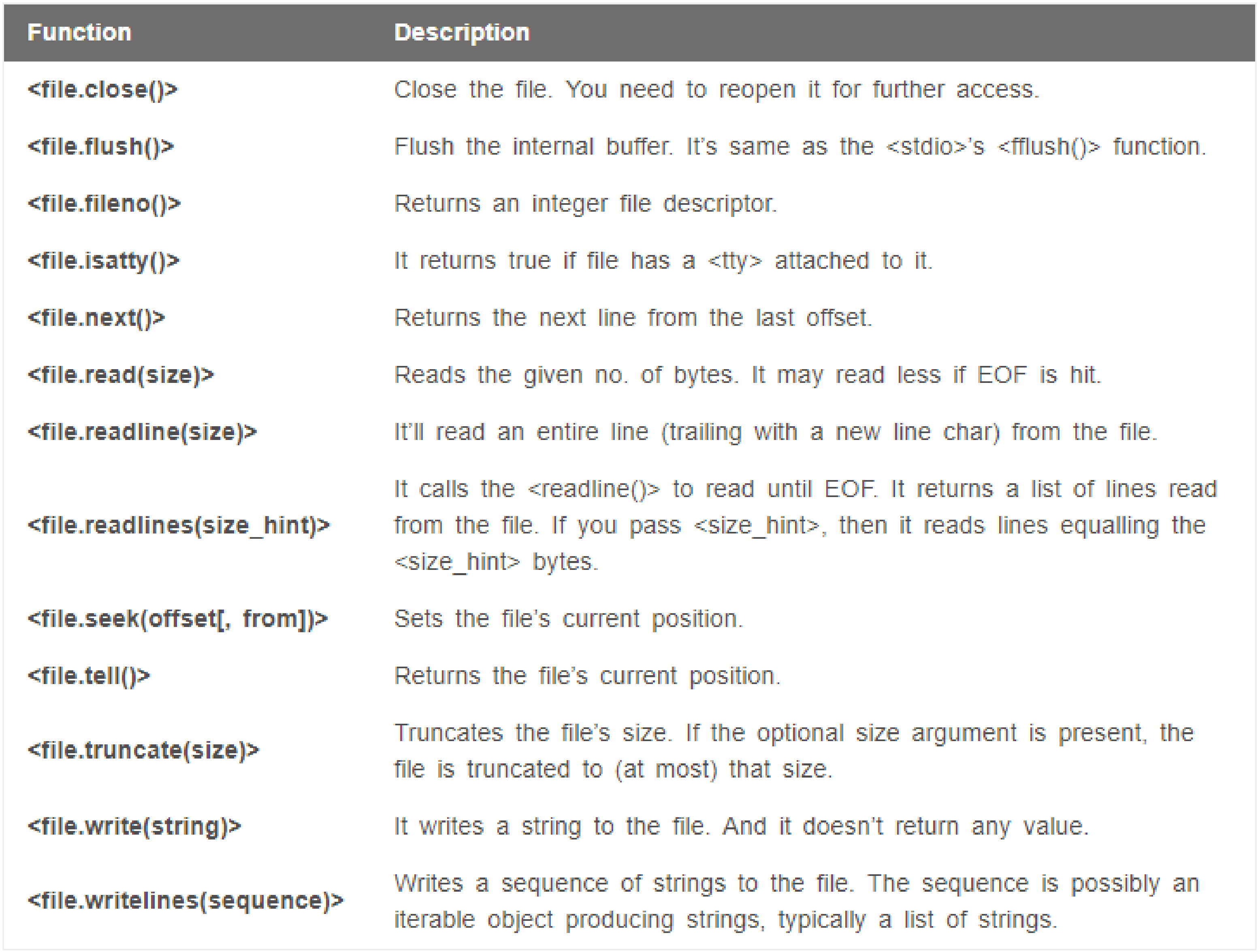


```python
with open('app.log', 'w', encoding = 'utf-8') as f:
    #first line
    f.write('my first file\n')
    #second line
    f.write('This file\n')
    #third line
    f.write('contains three lines\n')

with open('app.log', 'r', encoding = 'utf-8') as f:
    content = f.readlines()

for line in content:
    print(line)
```

```
with open('app.log', 'w', encoding = 'utf-8') as f:
    #first line
    f.write('my first file\n')
    #second line
    f.write('This file\n')
    #third line
    f.write('contains three lines\n')

f = open('app.log', 'r', encoding = 'utf-8')
print(f.read(10))     # read the first 10 data
#'my first f'

print(f.read(4))     # read the next 4 data
#'ile\n'

print(f.read())      # read in the rest till end of file
#'This file\ncontains three lines\n'

print(f.read())  # further reading returns empty sting
#''
```

```python
with open('app.log', 'w', encoding = 'utf-8') as f:
    #first line
    f.write('It is my first file\n')
    #second line
    f.write('This file\n')
    #third line
    f.write('contains three lines\n')

#Open a file
f = open('app.log', 'r+')
data = f.read(19);
print('Read String is : ', data)

#Check current position
position = f.tell();
print('Current file position : ', position)

#Reposition pointer at the beginning once again
position = f.seek(0, 0);
data = f.read(19);
print('Again read String is : ', data)

#Close the opened file
f.close()
```

FILE FUNCTIONS



Function	Description
<file.close()>	Close the file. You need to reopen it for further access.
<file.flush()>	Flush the internal buffer. It's same as the <stdio>'s <fflush()> function.
<file.fileno()>	Returns an integer file descriptor.
<file.isatty()>	It returns true if file has a <tty> attached to it.
<file.next()>	Returns the next line from the last offset.
<file.read(size)>	Reads the given no. of bytes. It may read less if EOF is hit.
<file.readline(size)>	It'll read an entire line (trailing with a new line char) from the file.
<file.readlines(size_hint)>	It calls the <readline()> to read until EOF. It returns a list of lines read from the file. If you pass <size_hint>, then it reads lines equalling the <size_hint> bytes.
<file.seek(offset[, from])>	Sets the file's current position.
<file.tell()>	Returns the file's current position.
<file.truncate(size)>	Truncates the file's size. If the optional size argument is present, the file is truncated to (at most) that size.
<file.write(string)>	It writes a string to the file. And it doesn't return any value.
<file.writelines(sequence)>	Writes a sequence of strings to the file. The sequence is possibly an iterable object producing strings, typically a list of strings.

Standard Library

https://docs.python.org/3/library/

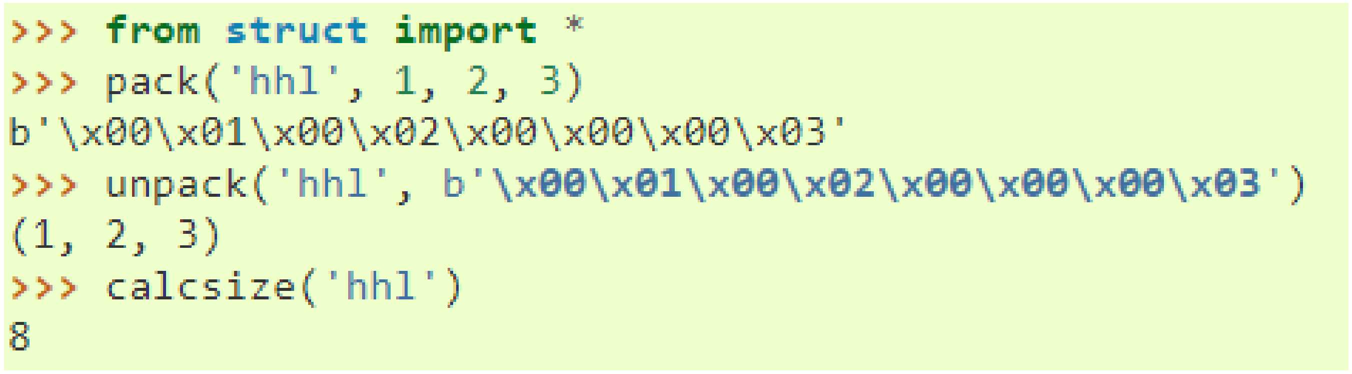
Ayyappan Murugesan
Phone: 7010774329
WhatsApp: 8870483557

Text Processing Services

- string — Common string operations
- re — Regular expression operations
- difflib — Helpers for computing deltas
- textwrap — Text wrapping and filling
- unicodedata — Unicode Database
- stringprep — Internet String Preparation
- readline — GNU readline interface
- rlcompleter — Completion function for GNU readline

Binary Data Services

- struct — Interpret bytes as packed binary data
- codecs — Codec registry and base classes



```
>>> from struct import *
>>> pack('hhl', 1, 2, 3)
b'\x00\x01\x00\x02\x00\x00\x00\x03'
>>> unpack('hhl', b'\x00\x01\x00\x02\x00\x00\x00\x03')
(1, 2, 3)
>>> calcsize('hhl')
8
```

Data Types

- datetime — Basic date and time types
- calendar — General calendar-related functions
- collections — Container datatypes
- collections.abc — Abstract Base Classes for Containers
- heapq — Heap queue algorithm
-]bisect — Array bisection algorithm
- array — Efficient arrays of numeric values
- weakref — Weak references
- types — Dynamic type creation and names for built-in types
- copy — Shallow and deep copy operations
- pprint — Data pretty printer
- reprlib — Alternate repr() implementation
- enum — Support for enumerations

Numeric and Mathematical Modules

- numbers — Numeric abstract base classes
- math — Mathematical functions
- cmath — Mathematical functions for complex numbers
- decimal — Decimal fixed point and floating point arithmetic
- fractions — Rational numbers
- random — Generate pseudo-random numbers
- statistics — Mathematical statistics functions

Functional Programming Modules

- itertools — Functions creating iterators for efficient looping
- functools — Higher-order functions and operations on callable objects
- operator — Standard operators as functions

File and Directory Access

- pathlib — Object-oriented filesystem paths
- os.path — Common pathname manipulations
- fileinput — Iterate over lines from multiple input streams
- stat — Interpreting stat() results
- filecmp — File and Directory Comparisons
- tempfile — Generate temporary files and directories
- glob — Unix style pathname pattern expansion
- fnmatch — Unix filename pattern matching
- linecache — Random access to text lines
- shutil — High-level file operations
- macpath — Mac OS 9 path manipulation functions

Data Persistence

- pickle — Python object serialization
- copyreg — Register pickle support functions
- shelve — Python object persistence
- marshal — Internal Python object serialization
- dbm — Interfaces to Unix “databases”
- sqlite3 — DB-API 2.0 interface for SQLite databases

Data Compression and Archiving

- zlib — Compression compatible with gzip
- gzip — Support for gzip files
- bz2 — Support for bzip2 compression
- lzma — Compression using the LZMA algorithm
- zipfile — Work with ZIP archives
- tarfile — Read and write tar archive files

File Formats

- csv — CSV File Reading and Writing
- configparser — Configuration file parser
- netrc — netrc file processing
- xdrlib — Encode and decode XDR data
- plistlib — Generate and parse Mac OS X .plist files

Cryptographic Services

- hashlib — Secure hashes and message digests
- hmac — Keyed-Hashing for Message Authentication
- secrets — Generate secure random numbers for managing secrets

Generic Operating System Services

- os — Miscellaneous operating system interfaces
- io — Core tools for working with streams
- time — Time access and conversions
- argparse — Parser for command-line options, arguments and sub-commands
- getopt — C-style parser for command line options
- logging — Logging facility for Python
- logging.config — Logging configuration
- logging.handlers — Logging handlers
- getpass — Portable password input
- curses — Terminal handling for character-cell displays
- curses.textpad — Text input widget for curses programs
- curses.ascii — Utilities for ASCII characters
- curses.panel — A panel stack extension for curses
- platform — Access to underlying platform’s identifying data
- errno — Standard errno system symbols
- ctypes — A foreign function library for Python

Concurrent Execution

- threading — Thread-based parallelism
- multiprocessing — Process-based parallelism
- The concurrent package
- concurrent.futures — Launching parallel tasks
- subprocess — Subprocess management
- sched — Event scheduler
- queue — A synchronized queue class
- dummy_threading — Drop-in replacement for the threading module
- _thread — Low-level threading API
- _dummy_thread — Drop-in replacement for the _thread module

Interprocess Communication and Networking

- socket — Low-level networking interface
- ssl — TLS/SSL wrapper for socket objects
- select — Waiting for I/O completion
- selectors — High-level I/O multiplexing
- asyncio — Asynchronous I/O, event loop, coroutines and tasks
- asyncore — Asynchronous socket handler
- asynchat — Asynchronous socket command/response handler
- signal — Set handlers for asynchronous events
- mmap — Memory-mapped file support

Internet Data Handling

- email — An email and MIME handling package
- json — JSON encoder and decoder
- mailcap — Mailcap file handling
- mailbox — Manipulate mailboxes in various formats
- mimetypes — Map filenames to MIME types
- base64 — Base16, Base32, Base64, Base85 Data Encodings
- binhex — Encode and decode binhex4 files
- binascii — Convert between binary and ASCII
- quopri — Encode and decode MIME quoted-printable data
- uu — Encode and decode uuencode files

Structured Markup Processing Tools

- html — HyperText Markup Language support
- html.parser — Simple HTML and XHTML parser
- html.entities — Definitions of HTML general entities
- XML Processing Modules
- xml.etree.ElementTree — The ElementTree XML API
- xml.dom — The Document Object Model API
- xml.dom.minidom — Minimal DOM implementation
- xml.dom.pulldom — Support for building partial DOM trees
- xml.sax — Support for SAX2 parsers
- xml.sax.handler — Base classes for SAX handlers
- xml.sax.saxutils — SAX Utilities
- xml.sax.xmlreader — Interface for XML parsers
- xml.parsers.expat — Fast XML parsing using Expat

Internet Protocols and Support

- webbrowser — Convenient Web-browser controller
- cgi — Common Gateway Interface support
- cgitb — Traceback manager for CGI scripts
- wsgiref — WSGI Utilities and Reference Implementation
- urllib — URL handling modules
- http — HTTP modules
- ftplib — FTP protocol client
- poplib — POP3 protocol client
- imaplib — IMAP4 protocol client
- nntplib — NNTP protocol client
- smtplib — SMTP protocol client
- smtpd — SMTP Server
- telnetlib — Telnet client
- uuid — UUID objects according to RFC 4122
- socketserver — A framework for network servers
- xmlrpc — XMLRPC server and client modules
- ipaddress — IPv4/IPv6 manipulation library

Multimedia Services

- audioop — Manipulate raw audio data
- aifc — Read and write AIFF and AIFC files
- sunau — Read and write Sun AU files
- wave — Read and write WAV files
- chunk — Read IFF chunked data
- colorsys — Conversions between color systems
- imghdr — Determine the type of an image
- sndhdr — Determine type of sound file
- ossaudiodev — Access to OSS-compatible audio devices

INTERNATIONALIZATION

- gettext — Multilingual internationalization services
- locale — Internationalization services

Program Frameworks

- turtle — Turtle graphics
- cmd — Support for line-oriented command interpreters
- shlex — Simple lexical analysis

Graphical User Interfaces with Tk

- tkinter — Python interface to Tcl/Tk
- tkinter.ttk — Tk themed widgets
- tkinter.tix — Extension widgets for Tk
- tkinter.scrolledtext — Scrolled Text Widget
- IDLE
- Other Graphical User Interface Packages

Development Tools

- typing — Support for type hints
- pydoc — Documentation generator and online help system
- doctest — Test interactive Python examples
- unittest — Unit testing framework
- 2to3 - Automated Python 2 to 3 code translation
- test — Regression tests package for Python

Debugging and Profiling

- bdb — Debugger framework
- faulthandler — Dump the Python traceback
- pdb — The Python Debugger
- The Python Profilers
- timeit — Measure execution time of small code snippets
- trace — Trace or track Python statement execution
- tracemalloc — Trace memory allocations

Software Packaging and Distribution

- distutils — Building and installing Python modules
- ensurepip — Bootstrapping the pip installer
- venv — Creation of virtual environments
- zipapp — Manage executable python zip archives

Python Runtime Services

- sys — System-specific parameters and functions
- sysconfig — Provide access to Python’s configuration information
- builtins — Built-in objects
- __main__ — Top-level script environment
- warnings — Warning control
- contextlib — Utilities for with-statement contexts
- abc — Abstract Base Classes
- atexit — Exit handlers
- traceback — Print or retrieve a stack traceback
- __future__ — Future statement definitions
- gc — Garbage Collector interface
- inspect — Inspect live objects
- site — Site-specific configuration hook
- fpectl — Floating point exception control

Custom Python Interpreters

- code — Interpreter base classes
- codeop — Compile Python code

Importing Modules

- zipimport — Import modules from Zip archives
- pkgutil — Package extension utility
- modulefinder — Find modules used by a script
- runpy — Locating and executing Python modules
- importlib — The implementation of import

Python Language Services

- parser — Access Python parse trees
- ast — Abstract Syntax Trees
- symtable — Access to the compiler’s symbol tables
- symbol — Constants used with Python parse trees
- token — Constants used with Python parse trees
- keyword — Testing for Python keywords
- tokenize — Tokenizer for Python source
- tabnanny — Detection of ambiguous indentation
- pyclbr — Python class browser support
- py_compile — Compile Python source files
- compileall — Byte-compile Python libraries
- dis — Disassembler for Python bytecode
- pickletools — Tools for pickle developers

Miscellaneous Services

- formatter — Generic output formatting

MS Windows Specific Services

- msilib — Read and write Microsoft Installer files
- msvcrt — Useful routines from the MS VC++ runtime
- winreg — Windows registry access
- winsound — Sound-playing interface for Windows

Unix Specific Services

- posix — The most common POSIX system calls
- pwd — The password database
- spwd — The shadow password database
- grp — The group database
- crypt — Function to check Unix passwords
- termios — POSIX style tty control
- tty — Terminal control functions
- pty — Pseudo-terminal utilities
- fcntl — The fcntl and ioctl system calls
- pipes — Interface to shell pipelines
- resource — Resource usage information
- nis — Interface to Sun’s NIS (Yellow Pages)
- syslog — Unix syslog library routines

Superseded Modules

- optparse — Parser for command line options
- imp — Access the import internals

Threads

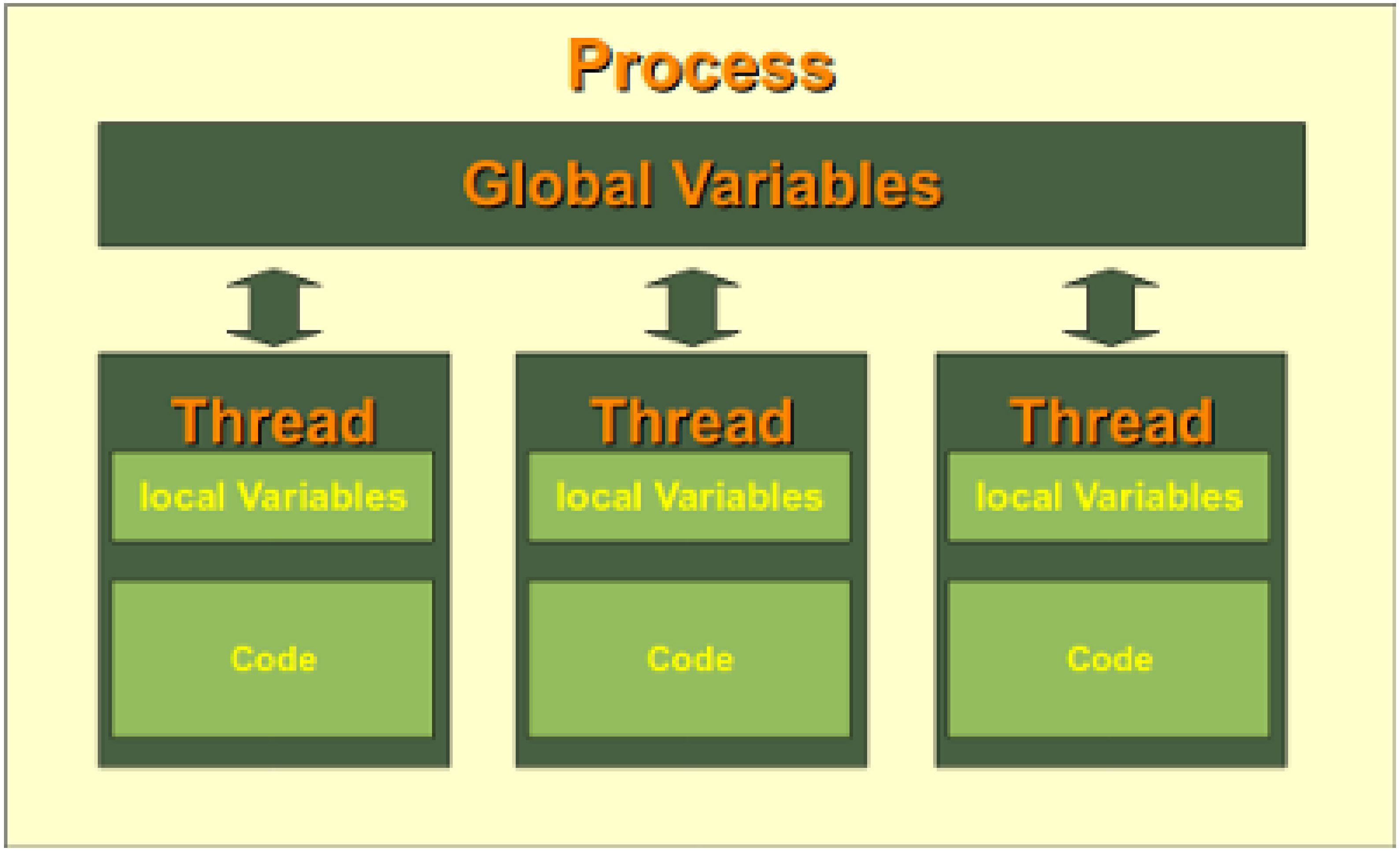
Ayyappan Murugesan

Phone: 7010774329

WhatsApp: 8870483557

THREAD

- Thread or a Thread of Execution is defined in computer science as the smallest unit that can be scheduled in an operating system
- Threads are normally created by a fork of a computer script or program in two or more parallel tasks
- Threads are usually contained in processes
- More than one thread can exist within the same process.
- These threads share the memory and the state of the process. In other words: They share the code or instructions and the values of its variables

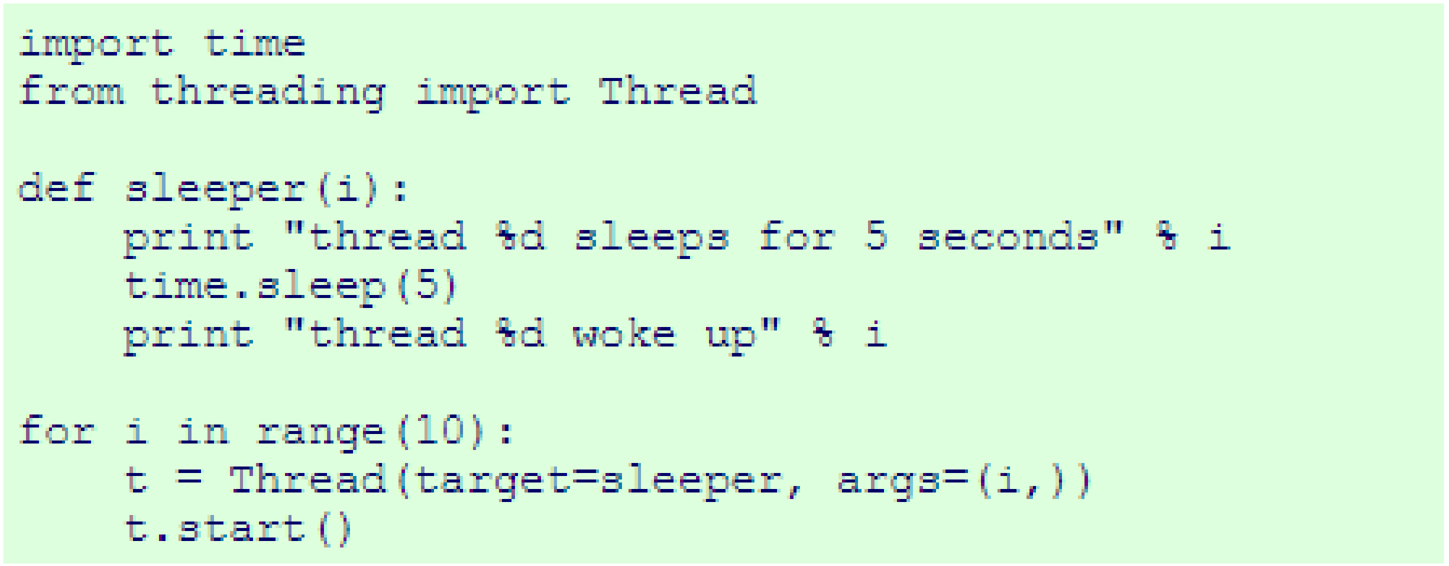

Process
Global Variables
Thread
local Variables
Code
Thread
local Variables
Code
Thread
local Variables
Code

Types

- Kernel threads
- User-space Threads or user threads
- Every process has at least one thread
- A process can start multiple threads
- The operating system executes these threads like parallel "processes“
- On a single processor machine, this parallelism is achieved by thread scheduling or timeslicing

Modules

- thread (_thread)
- threading



```
import time
from threading import Thread

def sleeper(i):
    print "thread %d sleeps for 5 seconds" % i
    time.sleep(5)
    print "thread %d woke up" % i

for i in range(10):
    t = Thread(target=sleeper, args=(i,))
    t.start()
```

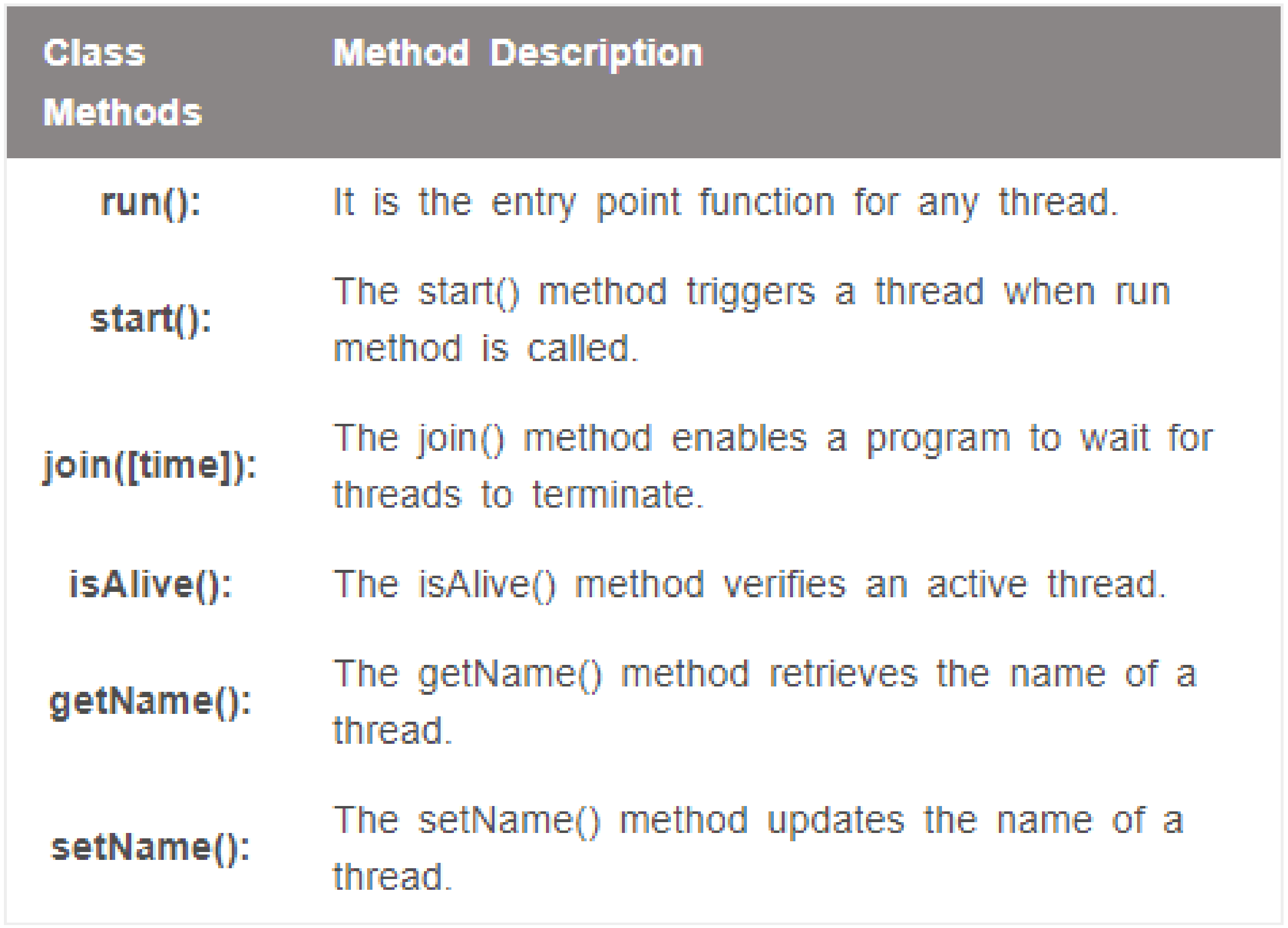
THREADING

threading.activeCount(): It finds the total no. of active thread objects.

threading.currentThread(): You can use it to determine the number of thread objects in the caller's thread control.

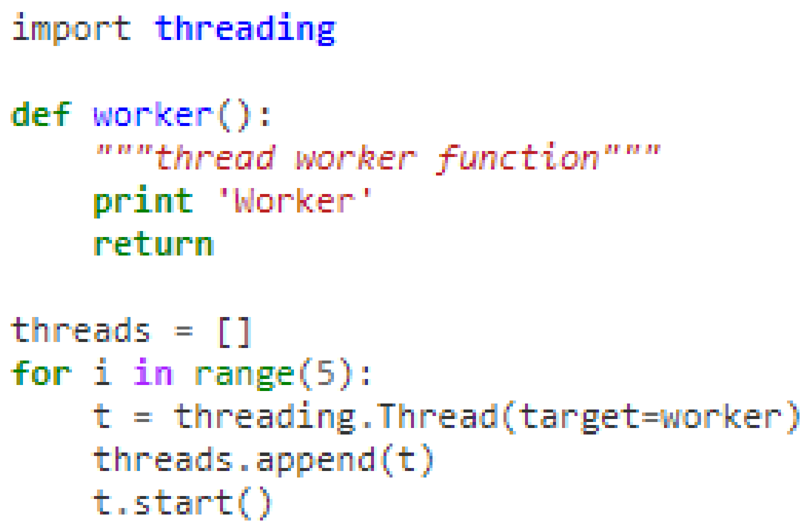
threading.enumerate(): It will give you a complete list of thread objects that are currently active.

THREADING.THREAD



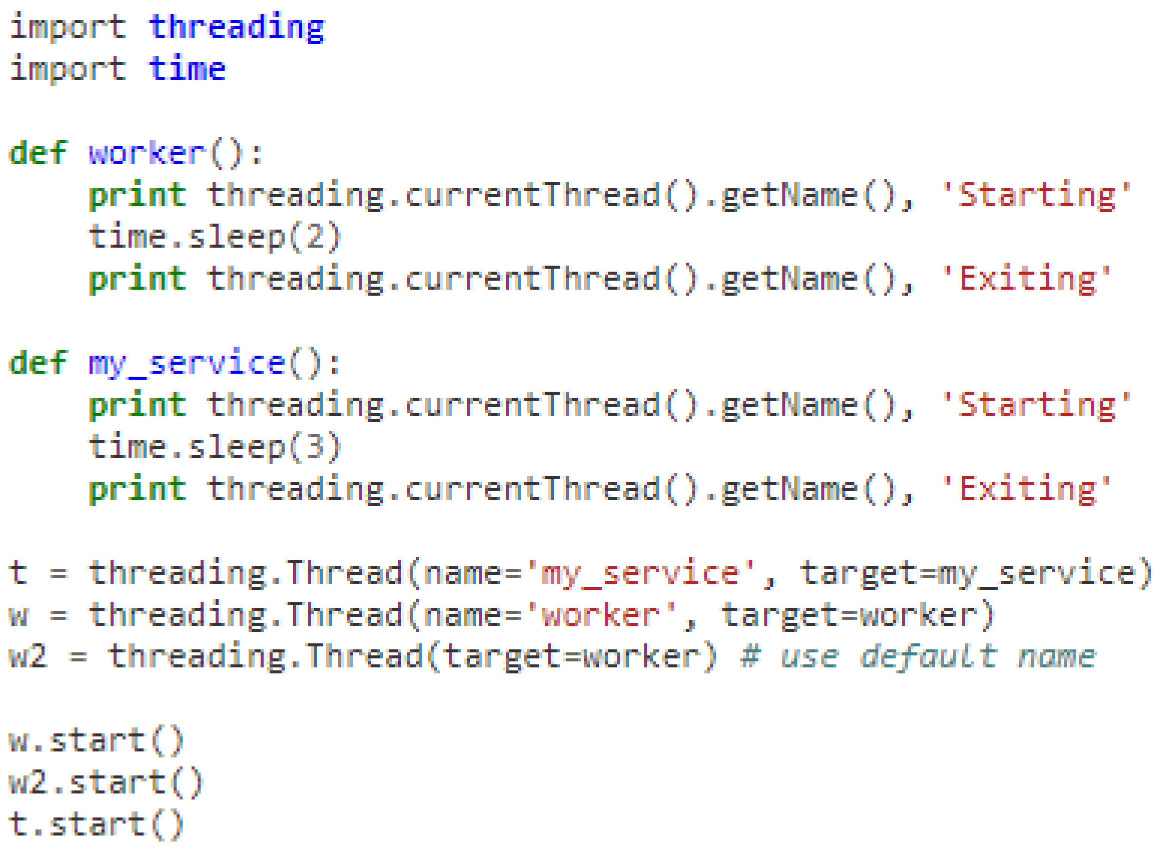
Class Methods	Method Description
run():	It is the entry point function for any thread.
start():	The start() method triggers a thread when run method is called.
join([time]):	The join() method enables a program to wait for threads to terminate.
isAlive():	The isAlive() method verifies an active thread.
getName():	The getName() method retrieves the name of a thread.
setName():	The setName() method updates the name of a thread.

THREAD OBJECTS



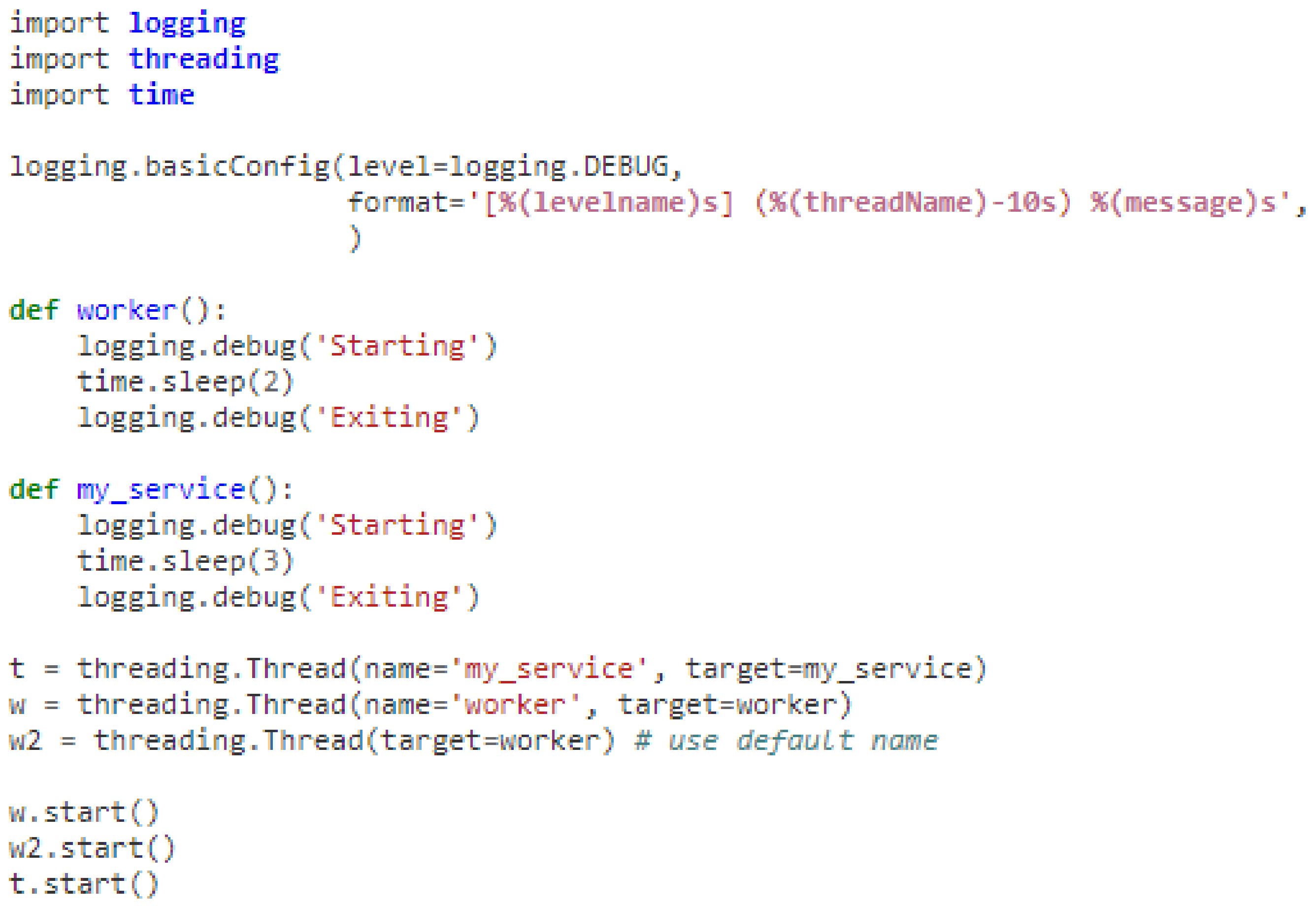
```
import threading

def worker():
    """thread worker function"""
    print 'Worker'
    return

threads = []
for i in range(5):
    t = threading.Thread(target=worker)
    threads.append(t)
    t.start()
```

Determining the Current Thread



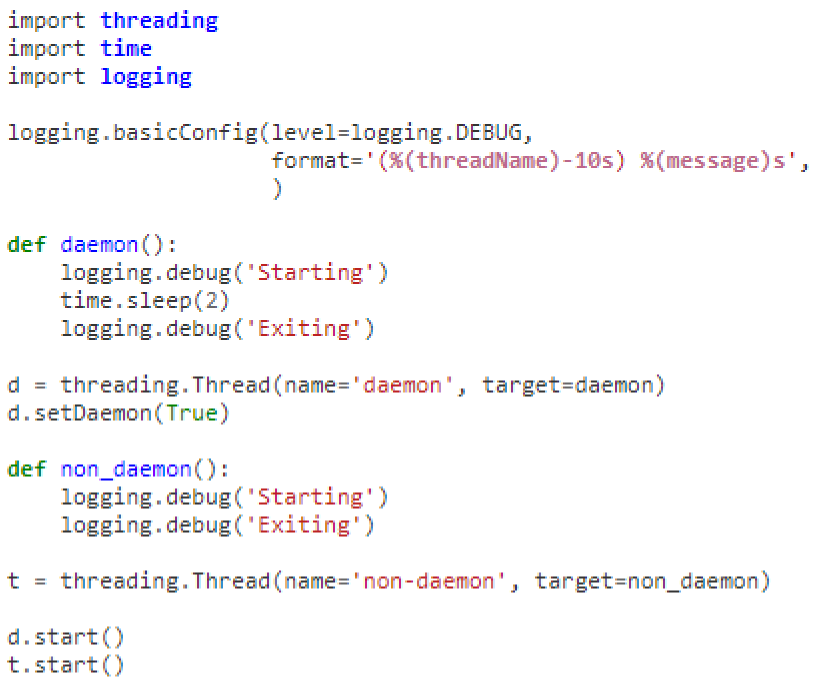
```python
import threading
import time

def worker():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(2)
    print threading.currentThread().getName(), 'Exiting'

def my_service():
    print threading.currentThread().getName(), 'Starting'
    time.sleep(3)
    print threading.currentThread().getName(), 'Exiting'

t = threading.Thread(name='my_service', target=my_service)
w = threading.Thread(name='worker', target=worker)
w2 = threading.Thread(target=worker) # use default name

w.start()
w2.start()
t.start()
```

WITH DEBUG LOGGING

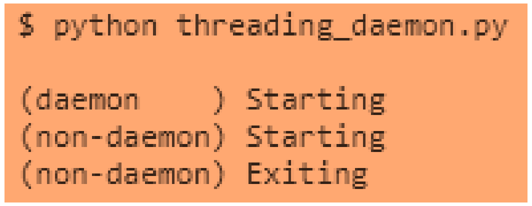


```
import logging
import threading
import time

logging.basicConfig(level=logging.DEBUG,
                    format='[%(levelname)s] (%(threadName)-10s) %(message)s',
                    )

def worker():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

def my_service():
    logging.debug('Starting')
    time.sleep(3)
    logging.debug('Exiting')

t = threading.Thread(name='my_service', target=my_service)
w = threading.Thread(name='worker', target=worker)
w2 = threading.Thread(target=worker) # use default name

w.start()
w2.start()
t.start()
```

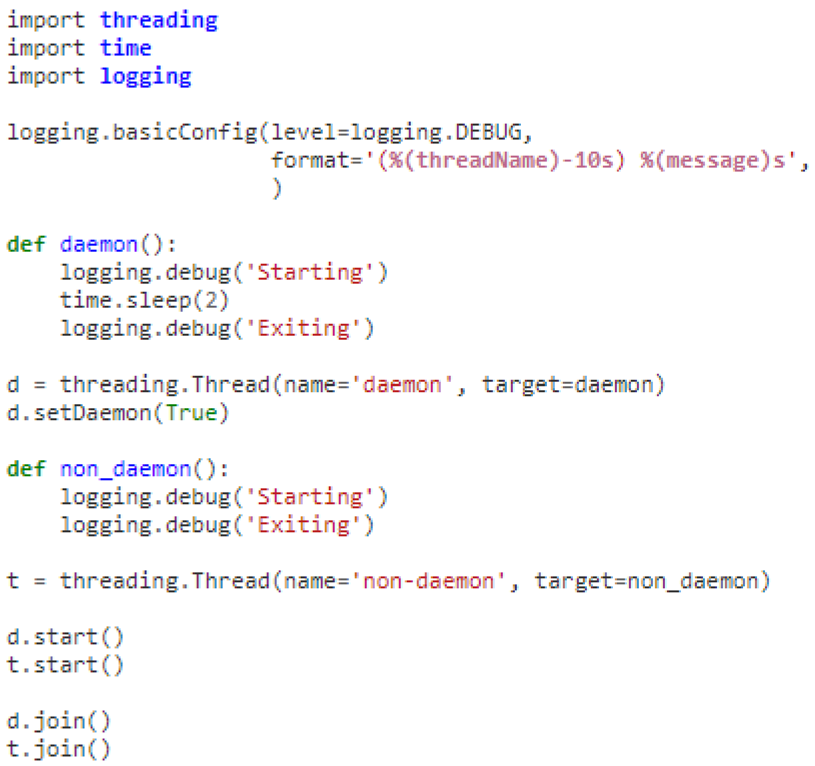
Daemon vs. Non-Daemon Threads



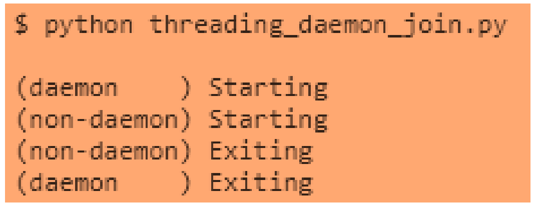
```python
import threading
import time
import logging

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

def daemon():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon', target=daemon)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()
```



```
$ python threading_daemon.py

(daemon    ) Starting
(non-daemon) Starting
(non-daemon) Exiting
```

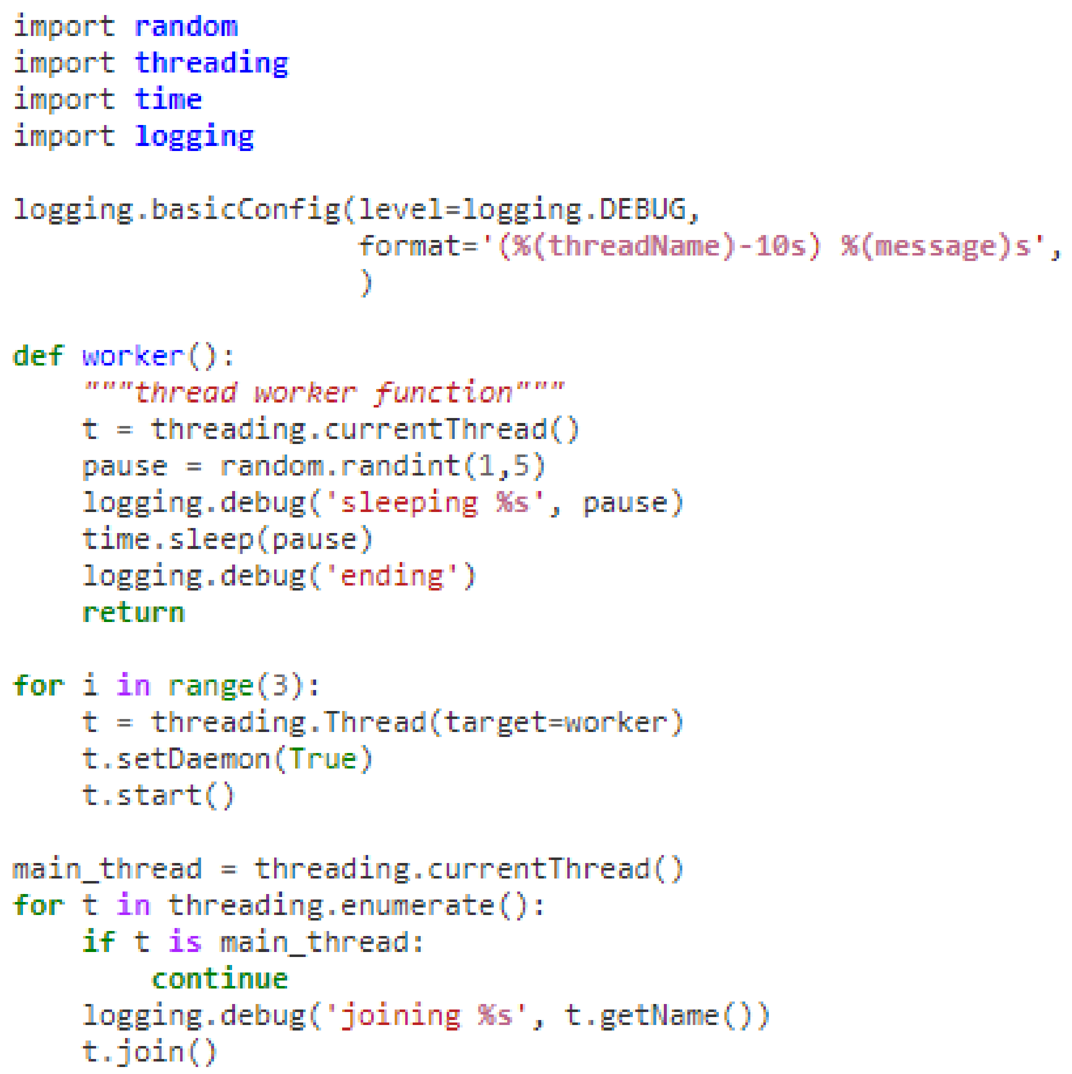


```python
import threading
import time
import logging

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

def daemon():
    logging.debug('Starting')
    time.sleep(2)
    logging.debug('Exiting')

d = threading.Thread(name='daemon', target=daemon)
d.setDaemon(True)

def non_daemon():
    logging.debug('Starting')
    logging.debug('Exiting')

t = threading.Thread(name='non-daemon', target=non_daemon)

d.start()
t.start()

d.join()
t.join()
```



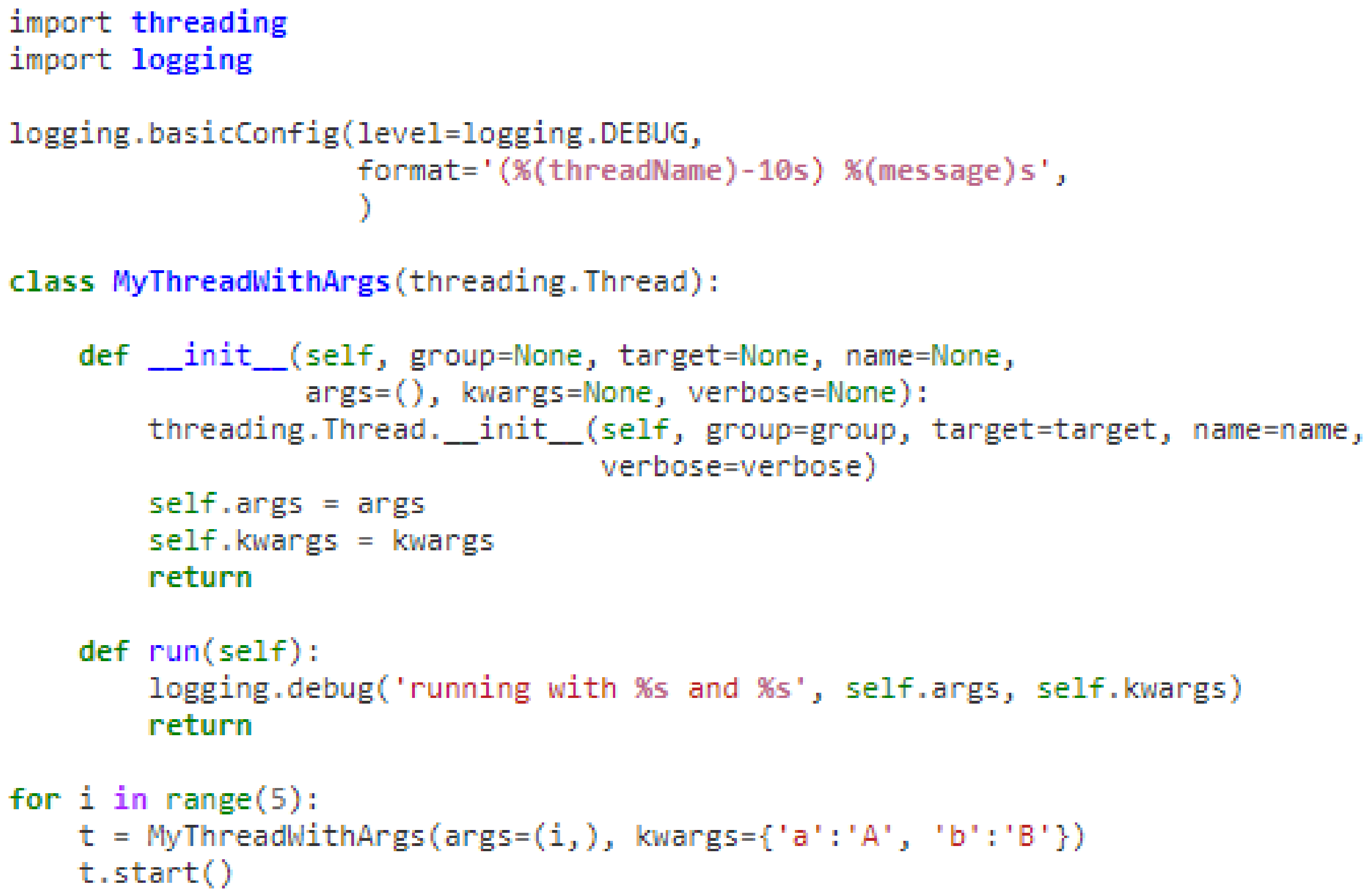
```
$ python threading_daemon_join.py

(daemon    ) Starting
(non-daemon) Starting
(non-daemon) Exiting
(daemon    ) Exiting
```

Enumerating All Threads



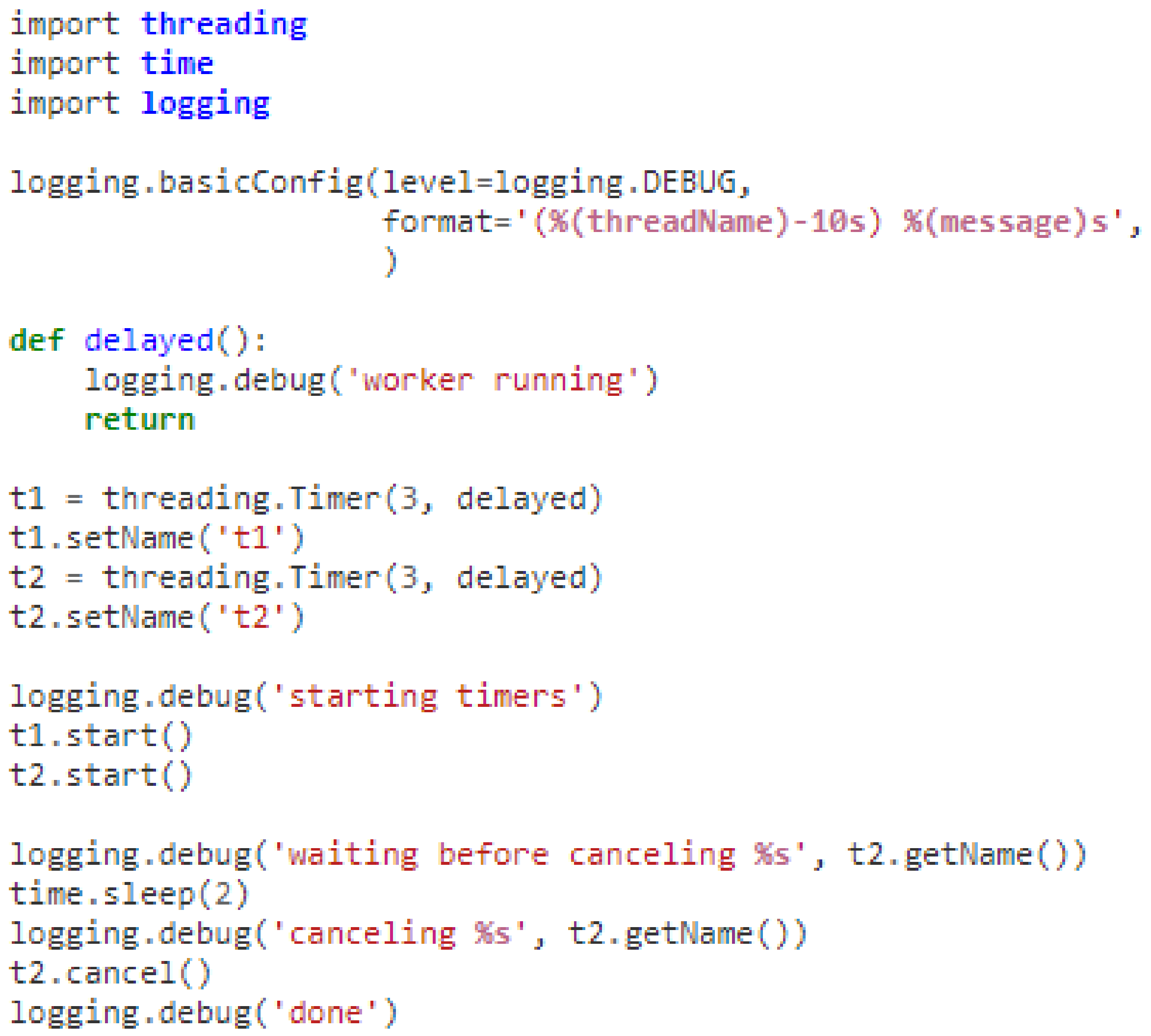
```python
import random
import threading
import time
import logging

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

def worker():
    """thread worker function"""
    t = threading.currentThread()
    pause = random.randint(1,5)
    logging.debug('sleeping %s', pause)
    time.sleep(pause)
    logging.debug('ending')
    return

for i in range(3):
    t = threading.Thread(target=worker)
    t.setDaemon(True)
    t.start()

main_thread = threading.currentThread()
for t in threading.enumerate():
    if t is main_thread:
        continue
    logging.debug('joining %s', t.getName())
    t.join()
```

SUBCLASSING THREAD



```
import threading
import logging

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

class MyThreadWithArgs(threading.Thread):

    def __init__(self, group=None, target=None, name=None,
                 args=(), kwargs=None, verbose=None):
        threading.Thread.__init__(self, group=group, target=target, name=name,
                                  verbose=verbose)
        self.args = args
        self.kwargs = kwargs
        return

    def run(self):
        logging.debug('running with %s and %s', self.args, self.kwargs)
        return

for i in range(5):
    t = MyThreadWithArgs(args=(i,), kwargs={'a':'A', 'b':'B'})
    t.start()
```

Timer Threads



```
import threading
import time
import logging

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

def delayed():
    logging.debug('worker running')
    return

t1 = threading.Timer(3, delayed)
t1.setName('t1')
t2 = threading.Timer(3, delayed)
t2.setName('t2')

logging.debug('starting timers')
t1.start()
t2.start()

logging.debug('waiting before canceling %s', t2.getName())
time.sleep(2)
logging.debug('canceling %s', t2.getName())
t2.cancel()
logging.debug('done')
```

```python
#Python multithreading example to print current date.
#1. Define a subclass using Thread class.
#2. Instantiate the subclass and trigger the thread.

import threading
import datetime

class myThread (threading.Thread):
     def __init__(self, name, counter):
          threading.Thread.__init__(self)
          self.threadID = counter
          self.name = name
          self.counter = counter
     def run(self):
          print "Starting " + self.name
          print_date(self.name, self.counter)
          print "Exiting " + self.name

def print_date(threadName, counter):
     datefields = []
     today = datetime.date.today()
     datefields.append(today)
     print "%s[%d]: %s" % ( threadName, counter, datefields[0] )

# Create new threads
thread1 = myThread("Thread", 1)
thread2 = myThread("Thread", 2)

# Start new Threads
thread1.start()
thread2.start()

thread1.join()
thread2.join()
print "Exiting the Program!!!"
```

Synchronizing Threads

```
#Python multithreading example to demonstrate locking.
#1. Define a subclass using Thread class.
#2. Instantiate the subclass and trigger the thread.
#3. Implement locks in thread's run method.

import threading
import datetime

exitFlag = 0

class myThread (threading.Thread):
     def __init__(self, name, counter):
          threading.Thread.__init__(self)
          self.threadID = counter
          self.name = name
          self.counter = counter
     def run(self):
          print "Starting " + self.name
          # Acquire lock to synchronize thread
          threadLock.acquire()
          print_date(self.name, self.counter)
          # Release lock for the next thread
          threadLock.release()
          print "Exiting " + self.name
```

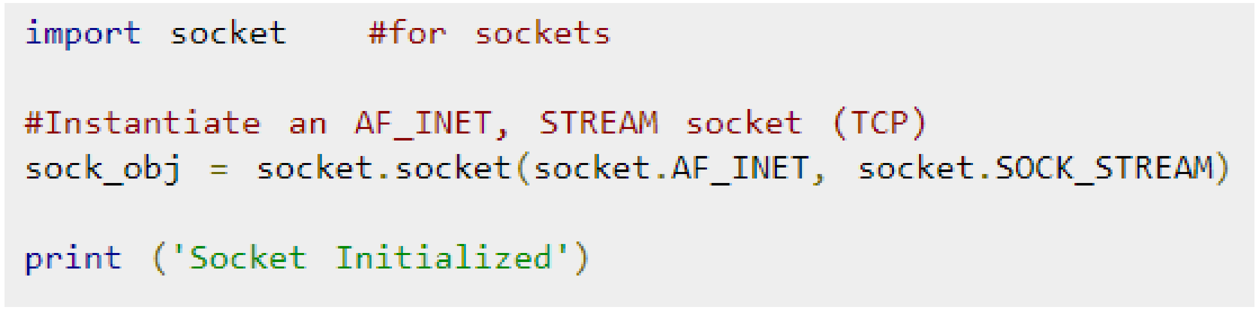
```
def print_date(threadName, counter):
     datefields = []
     today = datetime.date.today()
     datefields.append(today)
     print "%s[%d]: %s" % ( threadName, counter, datefields[0] )

threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread("Thread", 1)
thread2 = myThread("Thread", 2)

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
     t.join()
print "Exiting the Program!!!"
```

Socket

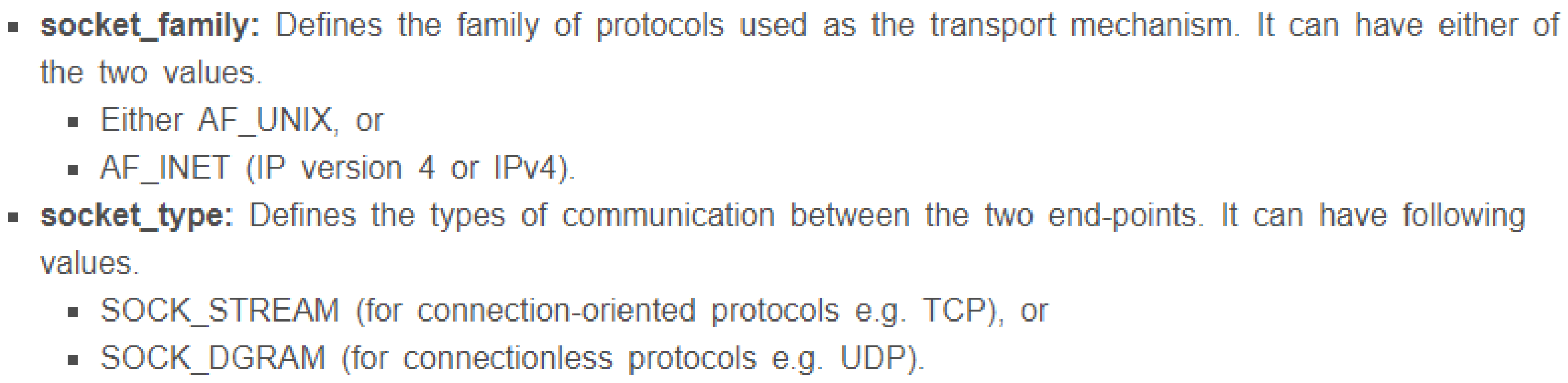
Ayyappan Murugesan
Phone: 7010774329
WhatsApp: 8870483557

INTRODUCTION

- *Sockets allow communication between processes that lie on the same machine, or on different machines working in diverse environment and even across different continents*

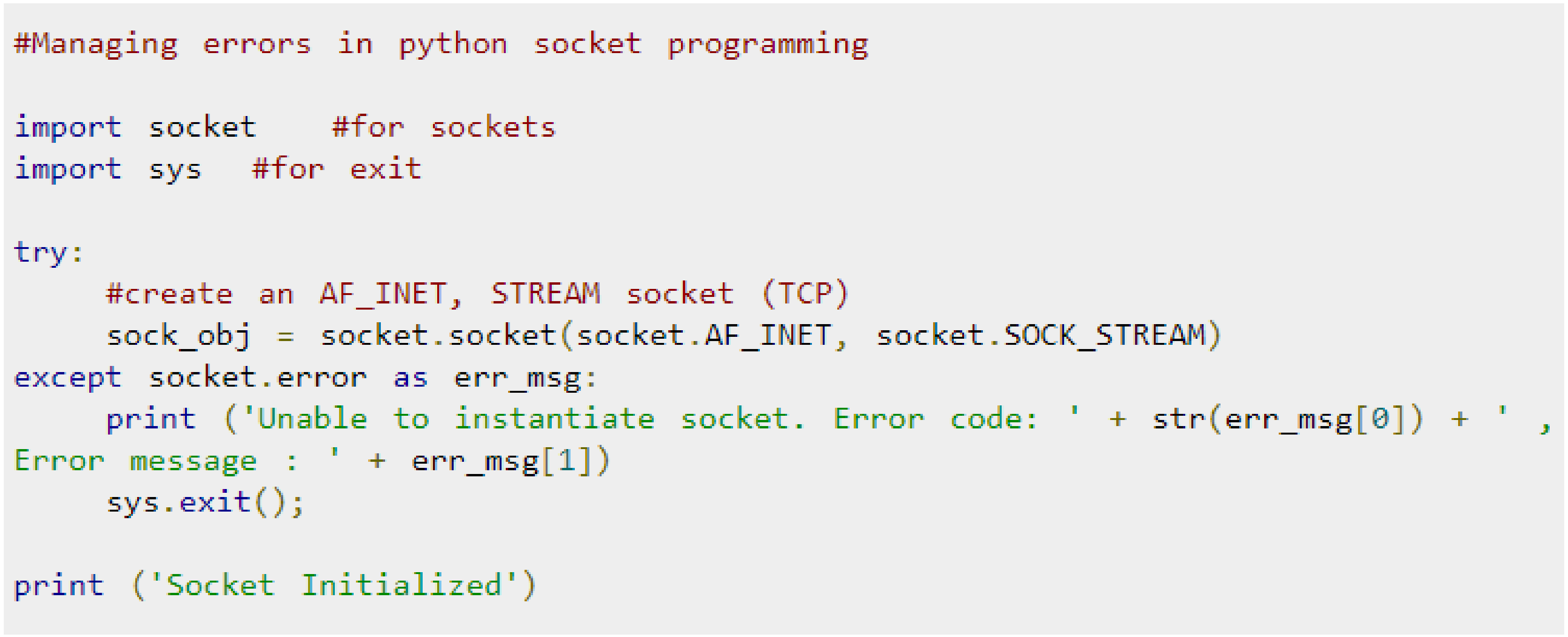


```
import socket    #for sockets

#Instantiate an AF_INET, STREAM socket (TCP)
sock_obj = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print ('Socket Initialized')
```



- **socket_family:** Defines the family of protocols used as the transport mechanism. It can have either of the two values.
  - Either AF_UNIX, or
  - AF_INET (IP version 4 or IPv4).
- **socket_type:** Defines the types of communication between the two end-points. It can have following values.
  - SOCK_STREAM (for connection-oriented protocols e.g. TCP), or
  - SOCK_DGRAM (for connectionless protocols e.g. UDP).

Socket Creation Error Handling



```
#Managing errors in python socket programming

import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    sock_obj = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error as err_msg:
    print ('Unable to instantiate socket. Error code: ' + str(err_msg[0]) + ' ,
Error message : ' + err_msg[1])
    sys.exit();

print ('Socket Initialized')
```

Server Socket Methods

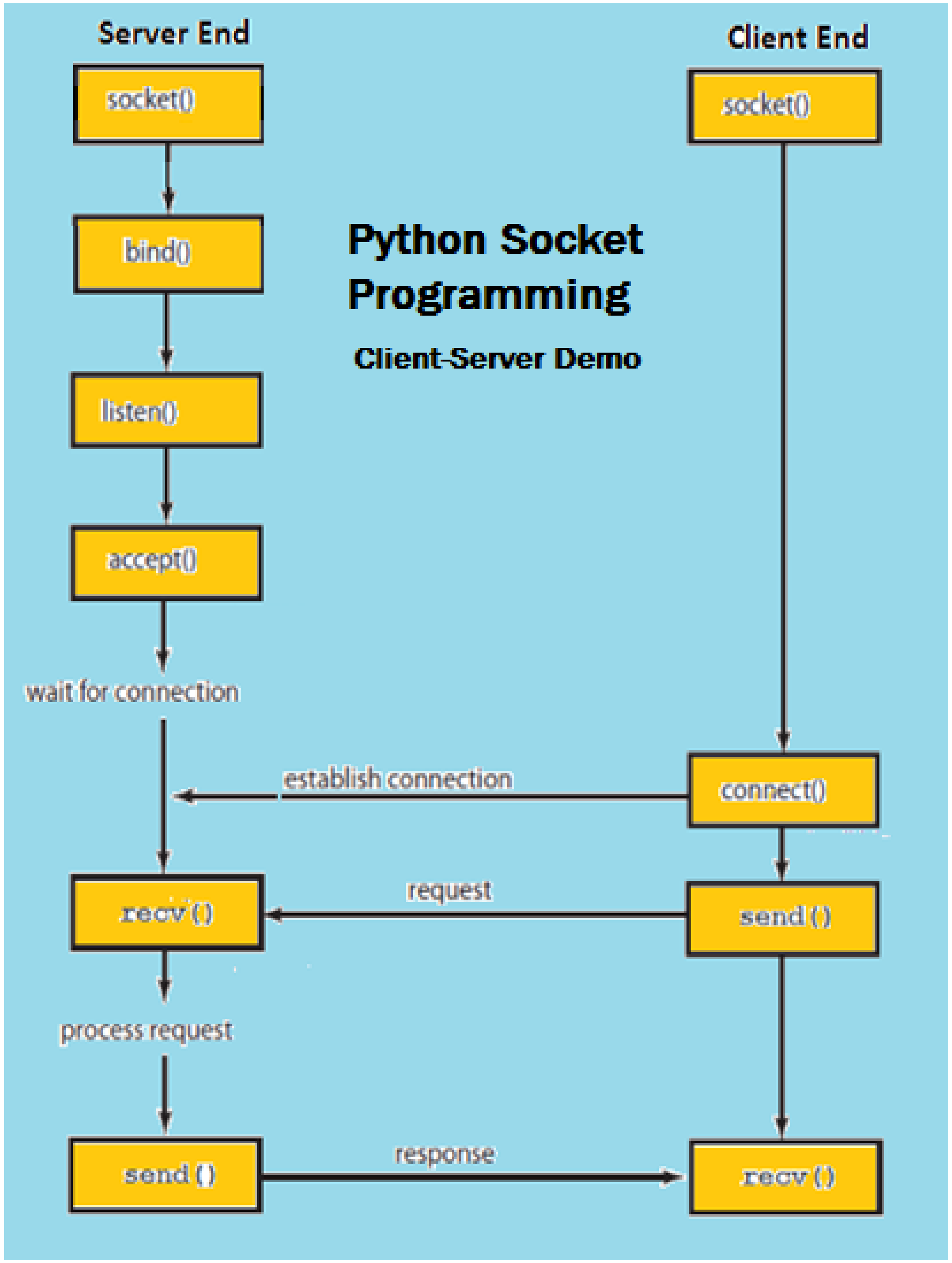
- **sock_object.bind(address):**
  - This method binds the socket to address (hostname, port number pair)
- **sock_object.listen(backlog):**
  - This method is used to listen to the connections associated with the socket.
  - The backlog parameter indicates the maximum number of queued connections.
  - Maximum value can go up to 5 and minimum should be at least zero.
- **sock_object.accept():**
  - This function returns (conn, address) pair where ‘conn’ is new socket object used to send and receive data on the communication channel and ‘address’ is the IP address tied to the socket on the another end of the channel.
  - Execution of accept() method returns a socket object that is different from the socket object created using socket.socket().
  - This new socket object is dedicatedly used to manage the communication with the particular client with which accept happened.
  - This mechanism also helps Server to maintain the connection with n number of clients simultaneously.

Client Socket Methods.

- **sock_object.connect():**This method is used to connect the client to host and port and initiate the connection towards the server.

General Socket Methods

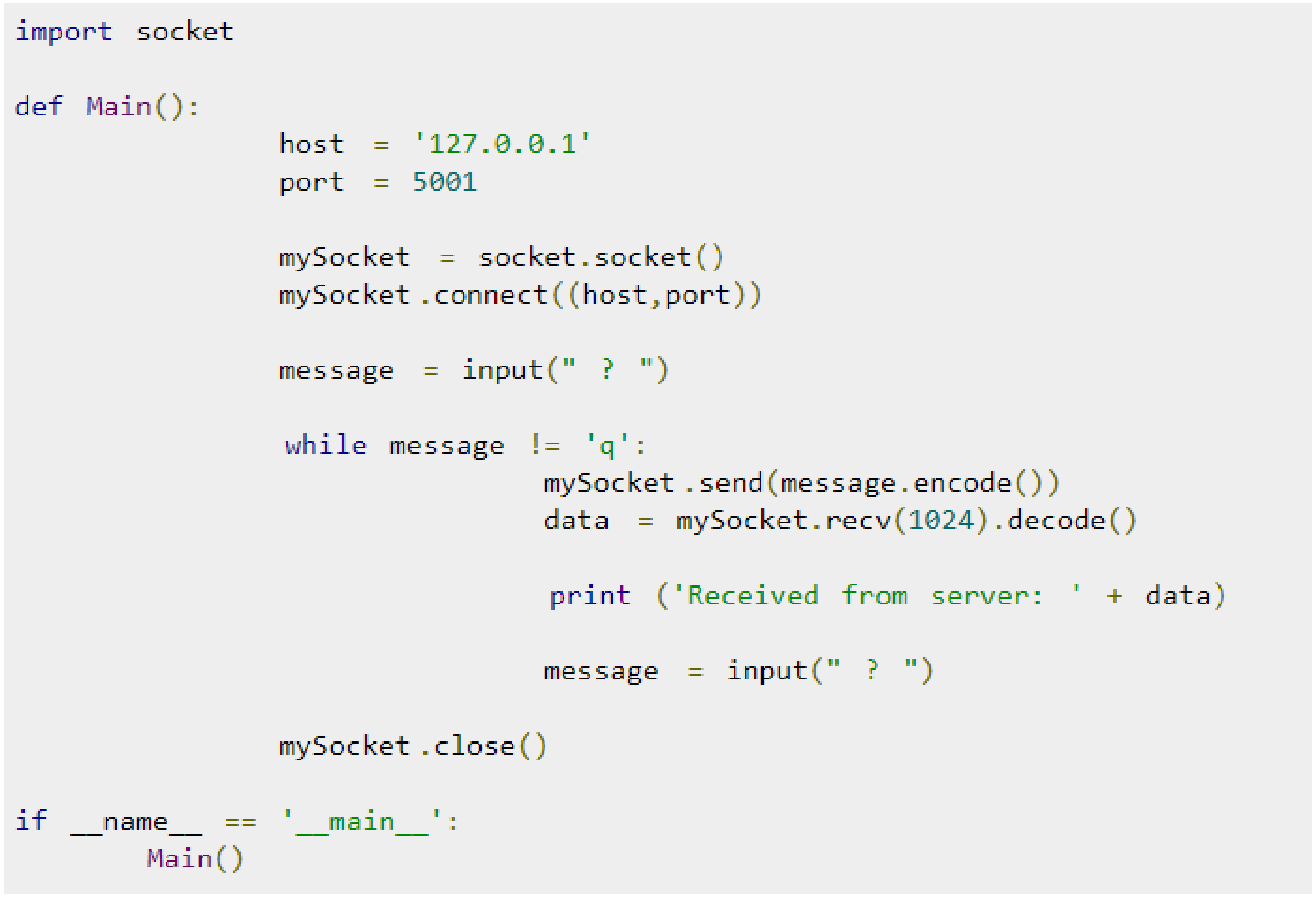
- **sock_object.recv():**
  - Use this method to receive messages at endpoints when the value of the protocol parameter is TCP.
- **sock_object.send():**
  - Apply this method to send messages from endpoints in case the protocol is TCP.
- **sock_object.recvfrom():**
  - Call this method to receive messages at endpoints if the protocol used is UDP.
- **sock_object.sendto():**
  - Invoke this method to send messages from endpoints if the protocol parameter is UDP.
- **sock_object.gethostname():**
  - This method returns the hostname.
- **sock_object.close():**
  - This method is used to close the socket. The remote endpoint will not receive data from this side.


Server End
Client End
socket()
socket()
bind()
Python Socket Programming
Client-Server Demo
listen()
accept()
wait for connection
establish connection
connect()
recv()
request
send()
process request
send()
response
recv()

Server Socket

```
import socket
import time


def Main():
    host = "127.0.0.1"
    port = 5001

    mySocket = socket.socket()
    mySocket.bind((host,port))

    mySocket.listen(1)
    conn, addr = mySocket.accept()
    print ("Connection from: " + str(addr))
    while True:
        data = conn.recv(1024).decode()
        if not data:
                                        break
        print ("from connected  user: " + str(data))

        data = str(data).upper()
        print ("Received from User: " + str(data))

        data = input(" ? ")
        conn.send(data.encode())

    conn.close()

if __name__ == '__main__':
                Main()
```

Client Socket



```python
import socket

def Main():
                host = '127.0.0.1'
                port = 5001

                mySocket = socket.socket()
                mySocket.connect((host,port))

                message = input(" ? ")

                while message != 'q':
                                mySocket.send(message.encode())
                                data = mySocket.recv(1024).decode()

                                print ('Received from server: ' + data)

                                message = input(" ? ")

                mySocket.close()

if __name__ == '__main__':
        Main()
```

Any Query??

Ayyappan Murugesan

Phone: 7010774329

WhatsApp: 8870483557