BinaryTides
Coding, Software, Tech and Reviews

Python socket – network programming tutorial

By Silver Moon | July 22, 2012 138 Comments

Network programming in python

This is a quick guide/tutorial on socket programming in python. Socket programming python is very similar to C.

To summarise the basics, sockets are the fundamental "things" behind any kind of network communications done by your computer. For example when you type www.google.com in your web browser, it opens a socket and connects to google.com to fetch the page and show it to you. Same with any chat client like gtalk or skype. Any network communication goes through a socket.

In this tutorial we shall be programming tcp sockets in python. You can also program udp sockets in python.

Before you begin

This tutorial assumes that you already have a basic knowledge of python.

So lets begin with sockets.

Creating a socket

This first thing to do is create a socket. The `socket.socket` function does this.
Quick Example :

```
#Socket client example in python

import socket   #for sockets

#create an AF_INET, STREAM socket (TCP)
s = socket.socket(socket.AF_INET, socket.SOC

print 'Socket Created'
```

Function `socket.socket` creates a socket and returns a socket descriptor which can be used in other socket related functions

The above code will create a socket with the following properties ...

Address Family : AF_INET (this is IP version 4 or IPv4)
Type : SOCK_STREAM (this means connection oriented TCP protocol)

Error handling

If any of the socket functions fail then python throws an exception called socket.error which must be caught.

```
#handling errors in python socket programs

import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socke
except socket.error, msg:
    print 'Failed to create socket. Error c
    sys.exit();

print 'Socket Created'
```

Ok , so you have created a socket successfully. But what next ?
Next we shall try to connect to some server using this socket. We can connect to www.google.com

Note

Apart from SOCK_STREAM type of sockets there is another type called SOCK_DGRAM which indicates the UDP protocol. This type of socket is non-connection socket. In this tutorial we shall stick to SOCK_STREAM or TCP sockets.

Connect to a Server

We connect to a remote server on a certain port number. So we need 2 things , IP address and port number to connect to. So you need to know the IP address of the remote server you are connecting to. Here we used the ip address of google.com as a sample.

First get the IP address of the remote host/url

Before connecting to a remote host, its ip address is needed. In python the getting the ip address is quite simple.

```
1  import socket   #for sockets
2  import sys  #for exit
3
4  try:
5      #create an AF_INET, STREAM socket (TCP)
6      s = socket.socket(socket.AF_INET, socke
7  except socket.error, msg:
8      print 'Failed to create socket. Error c
9      sys.exit();
10
11 print 'Socket Created'
12
13 host = 'www.google.com'
14
15 try:
16     remote_ip = socket.gethostbyname( host
17
18 except socket.gaierror:
19     #could not resolve
20     print 'Hostname could not be resolved.
21     sys.exit()
22
23 print 'Ip address of ' + host + ' is ' + re
```

Now that we have the ip address of the remote host/system, we can connect to ip on a certain 'port' using the connect function.

Quick example

```
1  import socket   #for sockets
2  import sys  #for exit
3
```

```
4  try:
5      #create an AF_INET, STREAM socket (TCP)
6      s = socket.socket(socket.AF_INET, socke
7  except socket.error, msg:
8      print 'Failed to create socket. Error c
9      sys.exit();
10
11 print 'Socket Created'
12
13 host = 'www.google.com'
14 port = 80
15
16 try:
17     remote_ip = socket.gethostbyname( host
18
19 except socket.gaierror:
20     #could not resolve
21     print 'Hostname could not be resolved.
22     sys.exit()
23
24 print 'Ip address of ' + host + ' is ' + re
25
26 #Connect to remote server
27 s.connect((remote_ip , port))
28
29 print 'Socket Connected to ' + host + ' on
```

Run the program

```
$ python client.py
Socket Created
Ip address of www.google.com is 74.125.236.83
Socket Connected to www.google.com on ip 74.125.236.8
```

It creates a socket and then connects. Try connecting to a port different from port 80 and you should not be able to connect which indicates that the port is not open for connection. This logic can be used to build a port scanner.

OK, so we are now connected. Lets do the next thing , sending some data to the remote server.

Free Tip

The concept of "connections" apply to SOCK_STREAM/TCP type of sockets. Connection means a reliable "stream" of data such that there can be multiple such streams each having communication of its own. Think of this as a pipe which is not interfered by data from other pipes. Another important property of stream connections is that packets have an "order" or "sequence".

Other sockets like UDP , ICMP , ARP dont have a concept of "connection". These are non-connection based communication. Which means you keep sending or receiving packets from anybody and everybody.

Sending Data

Function `sendall` will simply send data.

Lets send some data to google.com

```
import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socke
except socket.error, msg:
    print 'Failed to create socket. Error c
    sys.exit();

print 'Socket Created'

host = 'www.google.com'
port = 80

try:
    remote_ip = socket.gethostbyname( host

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved.
    sys.exit()

print 'Ip address of ' + host + ' is ' + re

#Connect to remote server
s.connect((remote_ip , port))

print 'Socket Connected to ' + host + ' on

#Send some data to remote server
message = "GET / HTTP/1.1\r\n\r\n"

try :
    #Set the whole string
    s.sendall(message)
except socket.error:
    #Send failed
    print 'Send failed'
    sys.exit()

print 'Message send successfully'
```

In the above example , we first connect to an ip address and then send the string message "GET / HTTP/1.1\r\n\r\n" to it. The message is actually an "http command" to fetch the mainpage of a website.

Now that we have send some data , its time to receive a reply from the server. So lets do it.

Receiving Data

Function `recv` is used to receive data on a socket. In the following example we shall send the same message as the last example and receive a reply from the server.

```
#Socket client example in python

import socket   #for sockets
import sys  #for exit

#create an INET, STREAMing socket
try:
    s = socket.socket(socket.AF_INET, socke
except socket.error:
    print 'Failed to create socket'
    sys.exit()

print 'Socket Created'

host = 'www.google.com';
port = 80;

try:
    remote_ip = socket.gethostbyname( host

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved.
    sys.exit()

#Connect to remote server
s.connect((remote_ip , port))

print 'Socket Connected to ' + host + ' on

#Send some data to remote server
message = "GET / HTTP/1.1\r\n\r\n"

try :
    #Set the whole string
    s.sendall(message)
except socket.error:
    #Send failed
```

```
    print 'Send failed'
    sys.exit()

print 'Message send successfully'

#Now receive data
reply = s.recv(4096)

print reply
```

Here is the output of the above code :

```
$ python client.py
Socket Created
Ip address of www.google.com is 74.125.236.81
Socket Connected to www.google.com on ip 74.125.236.8
Message send successfully
HTTP/1.1 302 Found
Location: http://www.google.co.in/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie: expires=; expires=Mon, 01-Jan-1990 00:00:
Set-Cookie: path=; expires=Mon, 01-Jan-1990 00:00:00
Set-Cookie: domain=; expires=Mon, 01-Jan-1990 00:00:0
Set-Cookie: expires=; expires=Mon, 01-Jan-1990 00:00:
Set-Cookie: path=; expires=Mon, 01-Jan-1990 00:00:00
Set-Cookie: domain=; expires=Mon, 01-Jan-1990 00:00:0
Set-Cookie: expires=; expires=Mon, 01-Jan-1990 00:00:
Set-Cookie: path=; expires=Mon, 01-Jan-1990 00:00:00
Set-Cookie: domain=; expires=Mon, 01-Jan-1990 00:00:0
Set-Cookie: expires=; expires=Mon, 01-Jan-1990 00:00:
Set-Cookie: path=; expires=Mon, 01-Jan-1990 00:00:00
Set-Cookie: domain=; expires=Mon, 01-Jan-1990 00:00:0
Set-Cookie: PREF=ID=51f26964398d27b0:FF=0:TM=13430260
```

Google.com replied with the content of the page we requested.

Quite simple!

Now that we have received our reply, its time to close the socket.

Close socket

Function `close` is used to close the socket.

```
s.close()
```

Thats it.

Lets Revise

So in the above example we learned how to :
1. Create a socket
2. Connect to remote server
3. Send some data
4. Receive a reply

Its useful to know that your web browser also does the same thing when you open www.google.com
This kind of socket activity represents a **CLIENT**. A client is a system that connects to a remote system to fetch data.

The other kind of socket activity is called a **SERVER**. A server is a system that uses sockets to receive incoming connections and provide them with data. It is just the opposite of Client. So www.google.com is a server and your web browser is a client. Or more technically www.google.com is a HTTP Server and your web browser is an HTTP client.

Now its time to do some server tasks using sockets.

Programming socket servers

OK now onto server things. Servers basically do the following :

1. Open a socket
2. Bind to a address(and port).
3. Listen for incoming connections.
4. Accept connections
5. Read/Send

We have already learnt how to open a socket. So the next thing would be to bind it.

Bind a socket

Function `bind` can be used to bind a socket to a particular address and port. It needs a sockaddr_in structure similar to connect function.

Quick example

```
import socket
import sys

HOST = ''   # Symbolic name meaning all ava
PORT = 8888 # Arbitrary non-privileged port

s = socket.socket(socket.AF_INET, socket.SO
print 'Socket created'

try:
    s.bind((HOST, PORT))
except socket.error , msg:
    print 'Bind failed. Error Code : ' + st
    sys.exit()

print 'Socket bind complete'
```

Now that bind is done, its time to make the socket listen to connections. We bind a socket to a particular IP address and a certain port number. By doing this we ensure that all incoming data which is directed towards this port number is received by this application.

This makes it obvious that you cannot have 2 sockets bound to the same port. There are exceptions to this rule but we shall look into that in some other article.

Listen for incoming connections

After binding a socket to a port the next thing we need to do is listen for connections. For this we need to put the socket in listening mode. Function `socket_listen` is used to put the socket in listening mode. Just add the following line after bind.

```
s.listen(10)
print 'Socket now listening'
```

The parameter of the function listen is called backlog. It controls the number of incoming connections that are kept "waiting" if the program is already busy. So by specifying 10, it means that if 10 connections are already waiting to be processed, then the 11th

connection request shall be rejected. This will be more clear after checking socket_accept.

Now comes the main part of accepting new connections.

Accept connection

Function `socket_accept` is used for this.

```
import socket
import sys

HOST = ''   # Symbolic name meaning all ava
PORT = 8888 # Arbitrary non-privileged port

s = socket.socket(socket.AF_INET, socket.SO
print 'Socket created'

try:
    s.bind((HOST, PORT))
except socket.error , msg:
    print 'Bind failed. Error Code : ' + st
    sys.exit()

print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'

#wait to accept a connection - blocking cal
conn, addr = s.accept()

#display client information
print 'Connected with ' + addr[0] + ':' + s
```

Output

Run the program. It should show

```
$ python server.py
Socket created
Socket bind complete
Socket now listening
```

So now this program is waiting for incoming connections on port 8888. Dont close this program , keep it running.
Now a client can connect to it on this port. We shall use the telnet client for testing this. Open a terminal and type

```
$ telnet localhost 8888
```

It will immediately show

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.
```

And the server output will show

```
$ python server.py
Socket created
Socket bind complete
Socket now listening
Connected with 127.0.0.1:59954
```

So we can see that the client connected to the server. Try the above steps till you get it working perfect.

We accepted an incoming connection but closed it immediately. This was not very productive. There are lots of things that can be done after an incoming connection is established. Afterall the connection was established for the purpose of communication. So lets reply to the client.

Function `sendall` can be used to send something to the socket of the incoming connection and the client should see it. Here is an example :

```
1  import socket
2  import sys
3
4  HOST = ''   # Symbolic name meaning all ava
5  PORT = 8888 # Arbitrary non-privileged port
6
7  s = socket.socket(socket.AF_INET, socket.SO
8  print 'Socket created'
9
10 try:
11     s.bind((HOST, PORT))
12 except socket.error , msg:
13     print 'Bind failed. Error Code : ' + st
14     sys.exit()
15
```

```
print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'

#wait to accept a connection - blocking cal
conn, addr = s.accept()

print 'Connected with ' + addr[0] + ':' + s

#now keep talking with the client
data = conn.recv(1024)
conn.sendall(data)

conn.close()
s.close()
```

Run the above code in 1 terminal. And connect to this server using telnet from another terminal and you should see this :

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
happy
happy
Connection closed by foreign host.
```

So the client(telnet) received a reply from server.

We can see that the connection is closed immediately after that simply because the server program ends after accepting and sending reply. A server like www.google.com is always up to accept incoming connections.

It means that a server is supposed to be running all the time. After-all its a server meant to serve. So we need to keep our server RUNNING non-stop. The simplest way to do this is to put the `accept` in a loop so that it can receive incoming connections all the time.

Live Server

So a live server will be alive always. Lets code this up

```
import socket
import sys

```

```
4  HOST = ''   # Symbolic name meaning all ava
5  PORT = 5000 # Arbitrary non-privileged port
6
7  s = socket.socket(socket.AF_INET, socket.SO
8  print 'Socket created'
9
10 try:
11     s.bind((HOST, PORT))
12 except socket.error , msg:
13     print 'Bind failed. Error Code : ' + st
14     sys.exit()
15
16 print 'Socket bind complete'
17
18 s.listen(10)
19 print 'Socket now listening'
20
21 #now keep talking with the client
22 while 1:
23     #wait to accept a connection - blocking
24     conn, addr = s.accept()
25     print 'Connected with ' + addr[0] + ':'
26
27     data = conn.recv(1024)
28     reply = 'OK...' + data
29     if not data:
30         break
31
32     conn.sendall(reply)
33
34 conn.close()
35 s.close()
```

We havent done a lot there. Just put the socket_accept in a loop.

Now run the server program in 1 terminal , and open 3 other terminals.
From each of the 3 terminal do a telnet to the server port.

Each of the telnet terminal would show :

```
$ telnet localhost 5000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
happy
OK .. happy
Connection closed by foreign host.
```

And the server terminal would show

```
$ python server.py
Socket created
Socket bind complete
Socket now listening
Connected with 127.0.0.1:60225
Connected with 127.0.0.1:60237
Connected with 127.0.0.1:60239
```

So now the server is running nonstop and the telnet terminals are also connected nonstop. Now close the server program. All telnet terminals would show "Connection closed by foreign host."

Good so far. But still there is not effective communication between the server and the client. The server program accepts connections in a loop and just send them a reply, after that it does nothing with them. Also it is not able to handle more than 1 connection at a time. So now its time to handle the connections , and handle multiple connections together.

Handling Connections

To handle every connection we need a separate handling code to run along with the main server accepting connections. One way to achieve this is using threads. The main server program accepts a connection and creates a new thread to handle communication for the connection, and then the server goes back to accept more connections.

We shall now use threads to create handlers for each connection the server accepts.

```
1  import socket
2  import sys
3  from thread import *
4
5  HOST = ''   # Symbolic name meaning all ava
6  PORT = 8888 # Arbitrary non-privileged port
7
8  s = socket.socket(socket.AF_INET, socket.SO
9  print 'Socket created'
10
11 #Bind socket to local host and port
12 try:
13     s.bind((HOST, PORT))
14 except socket.error , msg:
```

```
    print 'Bind failed. Error Code : ' + st
    sys.exit()

print 'Socket bind complete'

#Start listening on socket
s.listen(10)
print 'Socket now listening'

#Function for handling connections. This wi
def clientthread(conn):
    #Sending message to connected client
    conn.send('Welcome to the server. Type

    #infinite loop so that function do not
    while True:

        #Receiving from client
        data = conn.recv(1024)
        reply = 'OK...' + data
        if not data:
            break

        conn.sendall(reply)

    #came out of loop
    conn.close()

#now keep talking with the client
while 1:
    #wait to accept a connection - blocking
    conn, addr = s.accept()
    print 'Connected with ' + addr[0] + ':'

    #start new thread takes 1st argument as
    start_new_thread(clientthread ,(conn,))

s.close()
```

Run the above server and open 3 terminals like before. Now the server will create a thread for each client connecting to it.

The telnet terminals would show :

```
$ telnet localhost 8888
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Welcome to the server. Type something and hit enter
hi
OK...hi
asd
OK...asd
cv
OK...cv
```

The server terminal might look like this

```
$ python server.py
Socket created
Socket bind complete
Socket now listening
Connected with 127.0.0.1:60730
Connected with 127.0.0.1:60731
```

The above connection handler takes some input from the client and replies back with the same.

So now we have a server thats communicative. Thats useful now.

Conclusion

By now you must have learned the basics of socket programming in python. You can try out some experiments like writing a chat client or something similar.

When testing the code you might face this error

```
Bind failed. Error Code : 98 Message Address already
```

When it comes up, simply change the port number and the server would run fine.

If you think that the tutorial needs some addons or improvements or any of the code snippets above dont work then feel free to make a comment below so that it gets fixed.

Last Updated On : 9th January 2014

Related Post