2023

Python for Network Applications

BUILDING SOLUTIONS FOR NETWORK COMMUNICATION NEEDS

CORE360 IT SERVICES
TECHNOLOGY | BUSINESS SOLUTIONS | TRAINING

General T. De Leon, Marulas, Valenzuela City
Mobile(s): 09228720135 | 09954334855
Email: johnreygoh@gmail.com
Facebook: https://www.facebook.com/johnrey.goh

Contents

Sockets, IPv4, and Client/Server Programming

Python's socket module has both class-based and instances-based utilities. The difference between a class-based and instance-based method is that the former doesn't need an instance of a socket object. This is a very intuitive approach. For example, in order to print your machine's IP address, you don't need a socket object. Instead, you can just call the socket's class-based methods. On the other hand, if you need to send some data to a server application, it is more intuitive that you create a socket object to perform that explicit operation.

Printing local and remote machine's name and IP address

Sometimes, you need to quickly discover some information about your machine, for example, the hostname, IP address, number of network interfaces, and so on.

```
import socket

def print_machine_info():
	host_name = socket.gethostname()
	ip_address = socket.gethostbyname(host_name)
	print ("Host name: %s" %host_name)
	print ("IP address: %s" %ip_address)

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

Get a remote machine's IP address

```
import socket

def get_remote_machine_info():
	remote_host = 'www.python.org'
	try:
		print ("IP address of %s: %s" %(remote_host,
		socket.gethostbyname(remote_host)))
	except socket.error as err_msg:
		print ("%s: %s" %(remote_host, err_msg))

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

Finding a service name, given the port and protocol

```
import socket

def find_service_name():
	protocolname = 'tcp'
	for port in [80, 25]:
		print("Port: %s => service name: %s" %(port,socket.getservbyport(port, protocolname)))
	print("Port: %s => service name: %s" %(53,socket.getservbyport(53,'udp')))

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

Enumerating machine intefaces (linux and mac only)

```
import sys
import socket
import fcntl
import struct
import array

SIOCGIFCONF = 0x8912            #from C library sockios.h
STUCT_SIZE_32 = 32
STUCT_SIZE_64 = 40
PLATFORM_32_MAX_NUMBER = 2**32
DEFAULT_INTERFACES = 8

def list_interfaces():
        interfaces = []
        max_interfaces = DEFAULT_INTERFACES
        is_64bits = sys.maxsize > PLATFORM_32_MAX_NUMBER
        struct_size = STUCT_SIZE_64 if is_64bits else STUCT_SIZE_32
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        while True:
                bytes = max_interfaces * struct_size
                interface_names = array.array('B', b'\0' * bytes)
                sock_info = fcntl.ioctl(
                sock.fileno(),
                SIOCGIFCONF,
                struct.pack('iL', bytes, interface_names.buffer_info()[0]))
                outbytes = struct.unpack('iL', sock_info)[0]
                if outbytes == bytes:
                        max_interfaces *= 2
                else:
                        break

        namestr = interface_names.tostring()
        for i in range(0, outbytes, struct_size):
                interfaces.append((namestr[i:i+16].split(b'\0', 1)[0]).decode('ascii', 'ignore'))
        return interfaces

if __name__ == '__main__':
        interfaces = list_interfaces()
        print ("This machine has %s network interfaces: %s." %(len(interfaces), interfaces))
```

*Note: Download and Install NPcap to send and receive info from network machines.
https://npcap.com/#download

Enumerating machine interfaces (windows) using scapy

```
import psutil

def get_network_interfaces():
    interfaces = psutil.net_if_addrs()
    return list(interfaces.keys())

# Get the list of network interfaces
network_interfaces = get_network_interfaces()

# Print the results
print("Network Interfaces:")
for interface in network_interfaces:
    print(interface)
```

Enumerate Active Machines IP in LAN

```
from scapy.all import ARP, Ether, srp

def get_active_ips(ip_range):
    # Create an ARP request packet
    arp_request = Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip_range)

    # Send the packet and receive the response
    result = srp(arp_request, timeout=3, verbose=0)[0]

    # Extract the IP addresses from the response
    active_ips = [res[1].psrc for res in result]

    return active_ips

# Specify the IP range of your LAN
ip_range = "192.168.254.1/24"

# Get the active IP addresses
active_ips = get_active_ips(ip_range)

# Print the results
print("Active IP addresses:")
for ip in active_ips:
    print(ip)
```

Setting and getting the default socket timeout

You can make an instance of a socket object and call a gettimeout() method to get the default timeout value and the settimeout() method to set a specific timeout value. This is very useful in developing custom server applications.
We first create a socket object inside a test_socket_timeout() function. Then, we can use the getter/setter instance methods to manipulate timeout values.

```
import socket

def test_socket_timeout():
	s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	print ("Default socket timeout: %s" %s.gettimeout())
	s.settimeout(100)
	print ("Current socket timeout: %s" %s.gettimeout())

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

Handling socket errors

In any networking application, it is very common that one end is trying to connect, but the other party is not responding due to networking media failure or any other reason. The Python socket library has an elegant method of handing these errors via the socket.error exceptions.

To try this code:

```
handlesocketerrors.py --host localhost --port 80
```

```
import sys
import socket
import argparse

def main():
	# setup argument parsing
	parser = argparse.ArgumentParser(description='Socket Error Examples')
	parser.add_argument('--host', action="store", dest="host", required=False)
	parser.add_argument('--port', action="store", dest="port", type=int, required=False)
	parser.add_argument('--file', action="store", dest="file", required=False)
	given_args = parser.parse_args()
	host = given_args.host
	port = given_args.port
	filename = given_args.file

	# First try-except block -- create socket
	try:
		s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
	except socket.error as e:
		print ("Error creating socket: %s" % e)
		sys.exit(1)

	# Second try-except block -- connect to given host/port
	try:
		s.connect((host, port))
	except socket.gaierror as e:
		print ("Address-related error connecting to server: %s" % e)
		sys.exit(1)
	except socket.error as e:
```

```
            print ("Connection error: %s" % e)
            sys.exit(1)

    # Third try-except block -- sending data
    try:
            msg = "GET %s HTTP/1.0\r\n\r\n" % filename
            s.sendall(msg.encode('utf-8'))
    except socket.error as e:
            print ("Error sending data: %s" % e)
            sys.exit(1)

    while 1:
            # Fourth tr-except block – waiting to receive data from remote host
            try:
                    buf = s.recv(2048)
            except socket.error as e:
                    print ("Error receiving data: %s" % e)
                    sys.exit(1)
            if not len(buf):
                    break

            # write the received data
            sys.stdout.write(buf.decode('utf-8'))

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

Modifying socket's send/receive buffer sizes

The default socket buffer size may not be suitable in many circumstances. In such circumstances, you can change the default socket buffer size to a more suitable value.

```
import socket

SEND_BUF_SIZE = 4096
RECV_BUF_SIZE = 4096

def modify_buff_size():
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
    # Get the size of the socket's send buffer
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [Before]:%d" %bufsize)

    sock.setsockopt(socket.SOL_TCP,         socket.TCP_NODELAY, 1)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, SEND_BUF_SIZE)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, RECV_BUF_SIZE)
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [After]:%d" %bufsize)

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

Writing a SNTP client

Let us first define two constants: NTP_SERVER and TIME1970. NTP_SERVER is the server address to which our client will connect, and TIME1970 is the reference time on January 1,1970 (also called Epoch). You may find the value of the Epoch time or convert to the Epoch time at http://www.epochconverter.com/. The actual client creates a UDP socket (SOCK_DGRAM) to connect to the server following the UDP protocol. The client then needs to send the SNTP protocol data ('\x1b' + 47 * '\0') in a packet. Our UDP client sends and receives data using the sendto() and recvfrom() methods. When the server returns the time information in a packed array, the client needs a specialized struct module to unpack the data. The only interesting data is located in the 11th element of the array. Finally, we need to subtract the reference value, TIME1970, from the unpacked value to get the actual current time.

Simple network time protocol client

```
import socket
import struct
import sys
import time

NTP_SERVER = "0.asia.pool.ntp.org"
TIME1970 = 2208988800

def sntp_client():
        client = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
        data = '\x1b' + 47 * '\0'
        client.sendto( data.encode('utf-8'), ( NTP_SERVER, 123 ))
        data, address = client.recvfrom( 1024 )

        if data:
                print ('Response received from:', address)

        t = struct.unpack( '!12I', data )[10]
        t -= TIME1970
        print ('\tTime=%s' % time.ctime(t))

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

Writing a client/server application

Sample TCP Server [A]

```
import socket
import threading

bind_ip = "0.0.0.0"
bind_port = 9999
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((bind_ip,bind_port))
server.listen(5)
print "[*] Listening on %s:%d" % (bind_ip,bind_port)
```

```
# this is our client-handling thread
def handle_client(client_socket):
  # print out what the client sends
  request = client_socket.recv(1024)
  print "[*] Received: %s" % request
  # send back a packet
  client_socket.send("ACK!")
  client_socket.close()

while True:
  client,addr = server.accept()
  print "[*] Accepted connection from: %s:%d" % (addr[0],addr[1])
  # spin up our client thread to handle incoming data
  client_handler = threading.Thread(target=handle_client,args=(client,))
  client_handler.start()
```

Sample TCP Client [A]

```
import socket
target_host = "www.google.com"
target_port = 80
# create a socket object
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# connect the client
client.connect((target_host,target_port))
# send some data
client.send("GET / HTTP/1.1\r\nHost: google.com\r\n\r\n")
# receive some data
response = client.recv(4096)
print response
```

Sample UDP Client [A]

```
import socket
target_host = "127.0.0.1"
target_port = 80
# create a socket object
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# send some data
client.sendto("AAABBBCCC",(target_host,target_port))
# receive some data
data, addr = client.recvfrom(4096)
print data
```

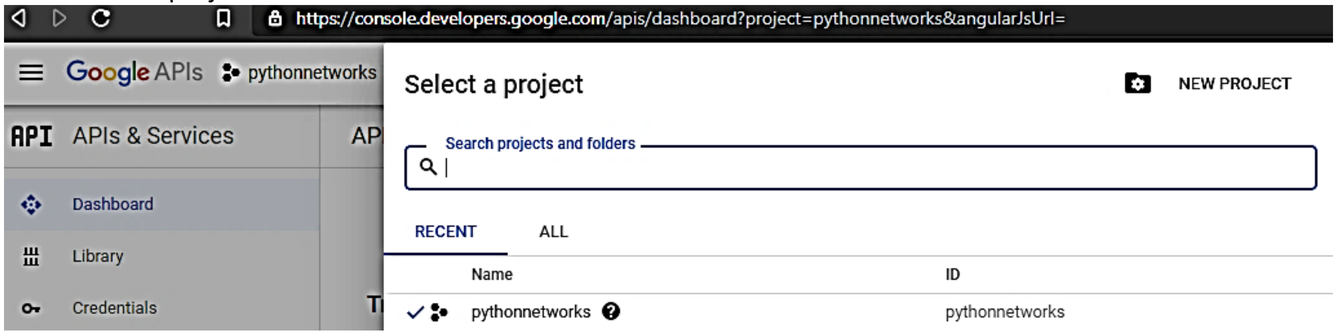
Web-scraping and Other Practical Applications

Searching for business addresses using the Google Maps API

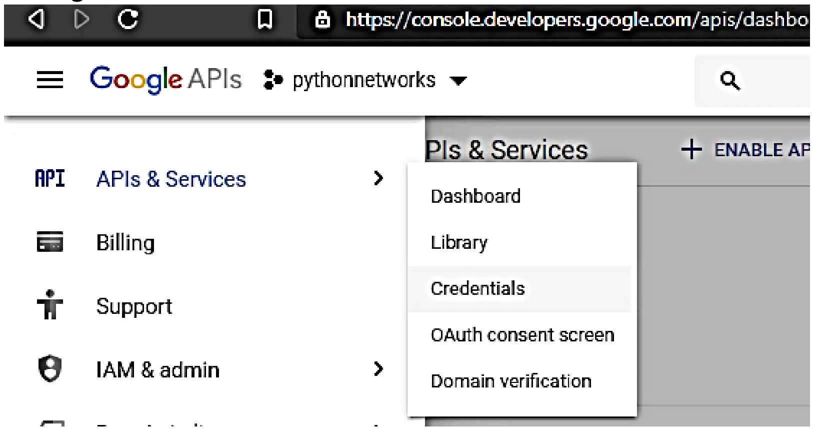
a. Login to

```
https://console.developers.google.com
```

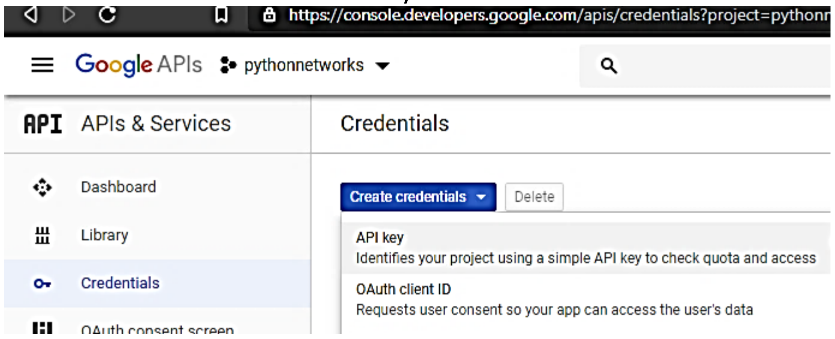
b. Create a new project there



c. Then go to credentials



d. Click create credential and API key


https://console.developers.google.com/apis/credentials?project=pythonn
Google APIs
pythonnetworks
APIs & Services
Dashboard
Library
Credentials
OAuth consent screen
Credentials
Create credentials
Delete
API key
Identifies your project using a simple API key to check quota and access
OAuth client ID
Requests user consent so your app can access the user's data

e. Install pygeocoder

```
\>pip install pygeocoder
```

f. For Python 3.3+, you have to update collections.py
C:\Users\<username>\AppData\Local\Programs\Python\Python310\Lib\collections\__init__.py

And add this:

```
from _collections_abc import Iterator
```

g. code

```
from pygeocoder import Geocoder

def search_business(business_name):
	results = Geocoder('AIzaSyB0zHaWS9DxjjZ...').geocode(business_name)
	for result in results:
		print(result)

if __name__ == '__main__':
	business_name = "Facebook"
	print("Searching %s" %business_name)
	search_business(business_name)
```

h. Free usage of google map api is limited on a daily basis

Searching for geographic coordinates using the Geopy

Sometimes you'd like to have a simple function that gives the geographic coordinates of a city by giving it just the name of that city. You may not be interested in installing any third-party libraries for this simple task.

a. Download and install geopy

```
\>pip install geopy
```

b. Sample code

```
from geopy.geocoders import Nominatim
import ssl
import certifi
import geopy.geocoders

def search_business(business_name):
    # Disable SSL certificate verification for Nominatim requests
    ctx = ssl.create_default_context(cafile=certifi.where())
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE

    geolocator = Nominatim(user_agent="myGeocoder", scheme='http', ssl_context=ctx)

    location = geolocator.geocode(business_name)

    if location:
        print("Location: ", location.address)
        print("Latitude: ", location.latitude)
        print("Longitude: ", location.longitude)
    else:
        print("Location not found.")

if __name__ == '__main__':
    business_name = "Facebook"
    print("Searching %s" % business_name)
    search_business(business_name)
```

sample output:

```
Searching Facebook
Location:  Facebook, Benhard, Maziwa sublocation, Kilimani location, Nairobi, Kilimani division, Westlands,
Nairobi, Kenya
Latitude:  -1.2772989
Longitude:  36.76095918258451
```

Searching for an article online

Wikipedia is a great site to gather information about virtually anything, for example, people, places, technology, and what not. If you like to search for something on Wikipedia from your Python script, this recipe is for you.

a. Download and install Wikipedia's API

```
\>pip install wikipedia
```

b. Sample code

```
import wikipedia
#print(wikipedia.summary("Cisco"))
#wikipedia.search("Cisco")
print(wikipedia.page("Cisco").content)
```

Reading news feed

If you are developing a social networking website with news and stories, you may be interested to present the news from various world news agencies

a. Download and import feedparser

```
\>pip install feedparser
```

b. Sample code

```
import feedparser

feed = feedparser.parse("https://finance.yahoo.com/rss/")

# Access feed title safely
feed_title = feed.feed.get('title', 'No feed title')

feed_entries = feed.entries

print("Feed Title: ", feed_title)

for entry in feed_entries:
    article_title = entry.get('title', 'No title')
    article_link = entry.get('link', 'No link')
    article_published_at = entry.get('published', 'No publish date')

    # Check if the published date is parsed
    article_published_at_parsed = entry.get('published_parsed', 'No parsed publish date')
    # Check if the author field exists
    article_author = entry.get('author', 'No author')

    # Accessing the summary/content of the article
    content = entry.get('summary', 'No summary')

    # Check if tags exist
    article_tags = [tag.term for tag in entry.tags] if 'tags' in entry else 'No tags'

    print("Title:", article_title)
    print("Link:", article_link)
    print("Published at:", article_published_at)
    print("Published at (parsed):", article_published_at_parsed)
    print("Author:", article_author)
    print("Content:", content)
    print("Tags:", article_tags)
    print()
```

Crawling links present in a web page

At times you would like to find a specific keyword present in a web page. In a web browser, you can use the browser's in-page search facility to locate the terms. Some browsers can highlight it. In a complex situation, you would like to dig deep and follow every URL present in a web page and find that specific term. This recipe will automate that task for you.

Sample code

```
import requests
from bs4 import BeautifulSoup

def web(page,WebUrl):
    try:
        if(page>0):
            url = WebUrl
            code = requests.get(url)
            plain = code.text
            s = BeautifulSoup(plain, "html.parser")
            for link in s.findAll('a', {'class':'styles__link___9msaS'}):
                tet = link.get('title')
                print(tet)
                tet_2 = link.get('href')
                print(url + tet_2)
                print()
    except ValueError as e:
        print(e)

web(1,'https://www.carousell.ph/categories/vehicles-32')
```

Programming Across Machine Boundaries

Executing a remote shell command

If you need to connect an old network switch or router via telnet, you can do so from a Python script instead of using a bash script or an interactive shell. This recipe will create a simple telnet session. It will show you how to execute shell commands to the remote host.

a. Install telnet service (linux) or enable telnet server(up to windows8 only)
   Control panel → programs → turn windows features on or off→telnet server

b. Windows 10 have no telnet server. You may download a 3rd party telnet server

```
https://sourceforge.net/projects/hk-telnet-server/files/latest/download
```

c. Sample code

```
import getpass
import sys
import telnetlib
HOST = "localhost"

def run_telnet_session():
        try:
                user = input("Enter your remote account: ")
                # Comment out the above line and uncomment the below line for Python 2.7.
                # user = raw_input("Enter your remote account: ")
                # password = getpass.getpass()
                password = input("password: ")
                session = telnetlib.Telnet(HOST)
                session.read_until(b"Username: ")
                session.write(user.encode('ascii') + b"\n")

                if password:
                        session.read_until(b"Password: ")
                        session.write(password.encode('ascii') + b"\n")

                session.write(b"notepad.exe\r\n")
                session.write(b"exit\r\n")
                print (session.read_all().decode('utf-8'))

        except ValueError as e:
                print(e)

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

Printing a remote machine's CPU information

a. Download and install paramiko

```
\>pip install paramiko
```

b. Windows 10 have no telnet server. You may download a 3rd party telnet server

```
https://sourceforge.net/projects/hk-telnet-server/files/latest/download
```

c. Sample code:

```
import getpass
import sys
import telnetlib
HOST = "localhost"

def run_telnet_session():
    try:
        user = input("Enter your remote account: ")
        # Comment out the above line and uncomment the below line for Python 2.7.
        # user = raw_input("Enter your remote account: ")
        # password = getpass.getpass()
        password = input("password: ")
        session = telnetlib.Telnet(HOST)
        session.read_until(b"Username: ")
        session.write(user.encode('ascii') + b"\n")

        if password:
            session.read_until(b"Password: ")
            session.write(password.encode('ascii') + b"\n")

        session.write(b"wmic cpu get caption, deviceid, name, numberofcores, maxclockspeed, status\r\n")
        session.write(b"exit\r\n")
        print (session.read_all().decode('utf-8'))

        # you can also output the result in a file like this:
        # fi="cpuinfo.txt"
        # fh=open(fi,'w')
        # fh.write(session.read_all().decode('utf-8'))
        # fh.close()


    except ValueError as e:
        print(e)

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

Copying a file to a remote machine by SFTP

If you want to upload or copy a file from your local machine to a remote machine securely, you can do so via Secure File Transfer Protocol (SFTP).

a. Download and install paramiko

```
\>pip install paramiko
```

b. Download and Install ssh server

Example: https://mobassh.mobatek.net/download.html

c. Set up users for your ssh server
d. Sample code

```
import argparse
import paramiko
import getpass
SOURCE = 'X:\\mydoc\\file1.txt'
DESTINATION ='X:\\mydoc\\ file2.txt'

def copy_file(hostname, port, username, password, src, dst):
        client = paramiko.SSHClient()
        client.load_system_host_keys()
        print (" Connecting to %s \n with username=%s...\n" %(hostname,username))

        t = paramiko.Transport(hostname, port)
        t.connect(username=username,password=password)
        sftp = paramiko.SFTPClient.from_transport(t)
        print ("Copying file: %s to path: %s" %(src, dst))

        sftp.put(src, dst)
        sftp.close()
        t.close()

if __name__ == '__main__':
        parser = argparse.ArgumentParser(description='Remote file copy')
        parser.add_argument('--host', action="store", dest="host", default='localhost')
        parser.add_argument('--port', action="store", dest="port", default=22, type=int)
        parser.add_argument('--src', action="store", dest="src", default=SOURCE)
        parser.add_argument('--dst', action="store", dest="dst", default=DESTINATION)
        given_args = parser.parse_args()
        hostname, port = given_args.host, given_args.port
        src, dst = given_args.src, given_args.dst
        user = input("Enter your remote account: ")
        # Comment out the above line and uncomment the below line for Python 2.7.
        # user = raw_input("Enter your remote account: ")
        # password = getpass.getpass("Enter password for %s: " %user)
        password = input("Enter password for %s: " %user)
        copy_file(hostname, port, user, password, src, dst)
```

Running a MySQL command remotely

If you ever need to administer a MySQL server remotely, this recipe is for you. It will show you how to send database commands to a remote MySQL server from a Python script. If you need to set up a web application that relies on a backend database, this recipe can be used as a part of your web application setup process.

a. Download and install apache web server and mysql database server
   *these two can be part of a package like ‘xampp’

b. Download and install the pymysql module

```
\>pip install pymysql
```

c. Sample code:

```
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print ("Database version : %s " % data)

# disconnect from server
db.close()
```

d. Sample code (create table)

```
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
   FIRST_NAME CHAR(20) NOT NULL,
   LAST_NAME CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()
```

e. Sample code (insert record)

```
#!/usr/bin/python3
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
  LAST_NAME, AGE, SEX, INCOME)
  VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
  # Execute the SQL command
  cursor.execute(sql)

  # Commit your changes in the database
  db.commit()

except:
  # Rollback in case there is any error
  db.rollback()

# disconnect from server
db.close()
```

f. Sample code (fetch records)

```
#!/usr/bin/python3
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM EMPLOYEE WHERE INCOME > '%d'" % (1000)

try:
  # Execute the SQL command
  cursor.execute(sql)

  # Fetch all the rows in a list of lists.
  results = cursor.fetchall()
  for row in results:
    fname = row[0]
```

```
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]

      # Now print fetched result
      print ("fname=%s,lname=%s,age=%d,sex=%s,income=%d" %(fname, lname, age, sex, income ))
except:
   print ("Error: unable to fetch data")

# disconnect from server
db.close()
```

Network Monitoring and Security

Note: sample of looping over a file with ip addresses

```
file1="hosts.txt"
fh=open(file1,"r")

data=fh.readlines()
hostlist=[]

for d in data:
        print(d.strip('\n'))
        hostlist.append(d.strip('\n'))

print(hostlist)
fh.close()
```

Creating a proxy server and sniffing packets on your network

If you are interested in sniffing packets on your local network, this recipe can be used as the starting point. Remember that you may not be able to sniff packets other than what is destined to your machine, as decent network switches will only forward traffic that is designated for your machine.

```
import sys
import socket
import threading



def server_loop(local_host, local_port, remote_host, remote_port, receive_first):
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        server.bind((local_host, local_port))
    except:
        print(f"[!!] Failed to listen on {local_host}:{local_port}")
        print("[!!] Check for other listening sockets or correct permissions")
        sys.exit(0)
    print(f"[*] Listening on {local_host}:{local_port}")

    server.listen(5)

    while True:
        client_socket, addr = server.accept()

        # print out the local connection information
        print(f"[==>] Received incoming connection from {addr[0]}:{addr[1]}")

        # start a thread to talk to the remote host
        proxy_thread = threading.Thread(target=proxy_handler,
                                args=(client_socket, remote_host, remote_port, receive_first))

        proxy_thread.start()


def main():
    # no fancy command-line parsing here
    if len(sys.argv[1:]) != 5:
        print("Usage: ./tcpproxy.py [localhost] [localport] [remotehost] "
              "[remoteport] [receive_first]")
        print("Example: ./tcpptoxy.py 127.0.0.1 9000 10.12.132.1 9000 True")
        sys.exit(0)

    # setup local listening parameters
    local_host = sys.argv[1]
    local_port = int(sys.argv[2])

    # setup remote target
    remote_host = sys.argv[3]
    remote_port = int(sys.argv[4])

    # this tells our proxy to connect and receive data
    # before sending to the remote host
    receive_first = sys.argv[5]

    if "True" in receive_first:
        receive_first = True
    else:
        receive_first = False

    # now spin up our listening socket
    server_loop(local_host, local_port, remote_host, remote_port, receive_first)


def proxy_handler(client_socket, remote_host, remote_port, receive_first):
    # connect to the remote host
    remote_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
remote_socket.connect((remote_host, remote_port))

# receive data from the remote end if necessary
if receive_first:

    remote_buffer = receive_from(remote_socket)
    hexdump(remote_buffer)

    # send it to our response handler
    remote_buffer = response_handler(remote_buffer)

    # if we have data to send to our local client, send it
    if len(remote_buffer):
        print(f"[<==] Sending {len(remote_buffer)} bytes to localhost.")
        client_socket.send(remote_buffer)

# now lets loop and read from local,
while True:

    # read from local host
    local_buffer = receive_from(client_socket)

    if len(local_buffer):
        print(f"[==>] Received {len(local_buffer)} bytes from localhost.")
        hexdump(local_buffer)

        # send it to our request handler
        local_buffer = request_handler(local_buffer)

        # send off the data to the remote host
        remote_socket.send(local_buffer)
        print("[==>] Sent to remote.")

    # receive back the response
    remote_buffer = receive_from(remote_socket)

    if len(remote_buffer):
        print(f"[<==] Received {len(remote_buffer)} bytes from remote.")
        hexdump(remote_buffer)

        # send to response handler
        remote_buffer = response_handler(remote_buffer)

        # send the response to the local socket
        client_socket.send(remote_buffer)

        print("[<==] Sent to localhost.")

    # if no more data on the either side, close the connections
    if not len(local_buffer) or not len(remote_buffer):
            client_socket.close()
            remote_socket.close()
            print("[*] No more data, Closing connections.")

            break


def hexdump(src, length=16):
    result = []
    digits = 4 if isinstance(src, str) else 2
    for i in range(0, len(src), length):
        s = src[i:i + length]
        hexa = " ".join(map("{0:0>2X}".format, s))
        text = "".join([chr(x) if 0x20 <= x < 0x7F else "." for x in s])
        result.append("%04X   %-*s   %s" % (i, length * (digits + 1), hexa, text))
    print("\n".join(result))


def receive_from(connection):
    buffer = b""

    # we set a 2 second timeout; depending on your
    # target, this may need to be adjusted
    connection.settimeout(2)

    try:
        # keep reading into the buffer until
        # there's no more data or we timeout
        count = 0
        while True:
            count += 1
            data = connection.recv(4096)

            if not data:
                break

            buffer += data

    except:
        pass

    return buffer


# modify any requests destined for the remote host
def request_handler(buffer):
    # perform packet modifications
    return buffer
```

```
# modify any responses destined for the local host
def response_handler(buffer):
   # perform packet modification
   return buffer


main()
```

Saving packets in the pcap format using the pcap dumper

The pcap format, abbreviated from packet capture, is a common file format for saving network data. If you want to save your captured network packets to a file and later reuse them for further processing, this can be a working example for you.

Sample code:

```
import os
from scapy.all import *

pkts = []
count = 0
pcapnum = 0

def write_cap(x):
	global pkts
	global count
	global pcapnum
	pkts.append(x)
	count += 1

	if count == 3:
		pcapnum += 1
		pname = "pcap%d.pcap" % pcapnum
		wrpcap(pname, pkts)
		pkts = []
		count = 0

def test_dump_file():
	print ("Testing the dump file...")
	dump_file = "./pcap1.pcap"

	if os.path.exists(dump_file):
		print ("dump fie %s found." %dump_file)
		pkts = sniff(offline=dump_file)
		count = 0
		while (count <=2):
			print ("----Dumping pkt:%s----" %count)
			print (hexdump(pkts[count]))
			count += 1
```

```
    else:
        print ("dump fie %s not found." %dump_file)

if __name__ == '__main__':
    print ("Started packet capturing and dumping... Press CTRL+C to exit")
    sniff(prn=write_cap)
    test_dump_file()
```

Scanning the ports of a remote host

If you are trying to connect to a remote host using a particular port, sometimes you get a message saying that Connection is refused. The reason for this is that, most likely, the server is down on the remote host. In such a situation, you can try to see whether the port is open or in the listening state. You can scan multiple ports to identify the available services in a machine.

Sample code:

```
import argparse
import socket
import sys

def scan_ports(host, start_port, end_port):
    """ Scan remote hosts """

    #Create socket
    try:
        sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    except socket.error as err_msg:
        print ('Socket creation failed. Error code: '+ str(err_msg[0]) + ' Error mesage: ' + err_msg[1])
        sys.exit()

    #Get IP of remote host
    try:
        remote_ip = socket.gethostbyname(host)
    except socket.error as error_msg:
        print (error_msg)
        sys.exit()

    #Scan ports
    end_port += 1
    for port in range(start_port,end_port):
        try:
            sock.connect((remote_ip,port))
            print ('Port ' + str(port) + ' is open')
            sock.close()
            sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        except socket.error:
            pass # skip various socket errors
```

```
if __name__ == '__main__':
	# setup commandline arguments
	parser = argparse.ArgumentParser(description='Remote Port Scanner')
	parser.add_argument('--host', action="store", dest="host", default='localhost')
	parser.add_argument('--start-port', action="store", dest="start_port", default=1, type=int)
	parser.add_argument('--end-port', action="store", dest="end_port", default=100, type=int)
	# parse arguments
	given_args = parser.parse_args()
	host, start_port, end_port = given_args.host,given_args.start_port, given_args.end_port
	scan_ports(host, start_port, end_port)


#sample usage
#hostscanner.py --host=localhost --start-port=1 --end-port=100
```

Scanning the broadcast of packets

We can use Scapy to sniff the packets arriving to a network interface. After each packet is captured, they can be processed by a callback function to get the useful information from it.

Sample code:

```
from scapy.all import *
import os
import datetime

captured_data = dict()
END_PORT = 1000

def monitor_packet(pkt):
	if IP in pkt:
		if pkt[IP].src not in captured_data:
			captured_data[pkt[IP].src] = []
	if TCP in pkt:
		if pkt[TCP].sport <= END_PORT:
			if not str(pkt[TCP].sport) in captured_data[pkt[IP].src]:
				captured_data[pkt[IP].src].append(str(pkt[TCP].sport))

	#os.system('cls')
	ip_list = sorted(captured_data.keys())

	for key in ip_list:
		ports=', '.join(captured_data[key])
		if len (captured_data[key]) == 0:
			print ('%s' % key)
		else:
			print ('%s (%s) [%s]' % (key, ports,datetime.datetime.now()))

if __name__ == '__main__':
	sniff(prn=monitor_packet, store=0)
```

Python Server Administration

Server administration often involves a variety of tasks, from file management to system monitoring. Python scripts can greatly simplify and automate these tasks. Here are 10 examples of Python scripts that can be useful in server administration:

Disk Space Monitoring:

Script to check the available disk space on the server and send alerts if it falls below a certain threshold.

```
import psutil

def check_disk_space(disk, threshold):
    """
    Check disk space on a given disk.
    :param disk: Disk to check (e.g., 'C:\\')
    :param threshold: Minimum free space percentage (e.g., 20 for 20%)
    """
    disk_usage = psutil.disk_usage(disk)
    free_percentage = (disk_usage.free / disk_usage.total) * 100

    print(f"Disk {disk} - Total space: {disk_usage.total / (1024**3):.2f} GB")
    print(f"Disk {disk} - Used space: {disk_usage.used / (1024**3):.2f} GB")
    print(f"Disk {disk} - Free space: {disk_usage.free / (1024**3):.2f} GB ({free_percentage:.2f}%)")

    if free_percentage < threshold:
        print(f"Warning: Low disk space on {disk} - only {free_percentage:.2f}% free")

# Example usage
check_disk_space('C:\\', 20)
```

Backup Automation:

Automate the process of backing up server files to a remote location or cloud storage at regular intervals.

Automating the process of backing up server files to a remote location or cloud storage at regular intervals can be accomplished with a Python script using libraries such as paramiko for SSH-based transfers or boto3 for AWS S3 storage.

Script that backs up files to an AWS S3 bucket using boto3. This script will require AWS credentials and the boto3 library.

```
pip install boto3
```

```
import os
import boto3
from datetime import datetime

def backup_files_to_s3(local_directory, bucket, s3_folder):
    """
    Backup files from a local directory to an AWS S3 bucket.

    :param local_directory: Path to the local directory to back up.
    :param bucket: Name of the S3 bucket.
    :param s3_folder: Folder in the S3 bucket to store the backups.
    """
    # Create an S3 client
    s3_client = boto3.client('s3')

    for root, dirs, files in os.walk(local_directory):
        for file in files:
            local_path = os.path.join(root, file)
            relative_path = os.path.relpath(local_path, local_directory)
            s3_path = os.path.join(s3_folder, relative_path)

            print(f"Uploading {local_path} to {bucket}/{s3_path}")
            s3_client.upload_file(local_path, bucket, s3_path)

def main():
    # Define your local directory, S3 bucket, and S3 folder
    local_directory = '/path/to/your/local/directory'
    bucket = 'your-s3-bucket-name'
    s3_folder = 'backup-' + datetime.now().strftime('%Y-%m-%d-%H-%M-%S')

    backup_files_to_s3(local_directory, bucket, s3_folder)

if __name__ == "__main__":
    main()
```

Replace /path/to/your/local/directory with the path of the local directory you want to back up, your-s3-bucket-name with the name of your S3 bucket, and ensure that you have configured AWS credentials that have access to the S3 bucket.

To run this script at regular intervals, you can use a task scheduler:

- On Windows: Use the Task Scheduler to run this script at your desired intervals.
- On Linux: Use cron jobs to schedule the script.

Backing up to a remote shared folder on a local network can be done using Python with the smbprotocol library, which allows you to connect to SMB (Server Message Block) shares, commonly used in Windows networking. The script will copy files from a local directory to a shared network folder.

First, you need to install the smbprotocol library:

```
pip install smbprotocol
```

```
import os
from smbclient import register_session, SambaClient

# Replace these with your network share details
NETWORK_SHARE_PATH = r'\\SERVER\shared_folder'
USERNAME = 'your_username'
PASSWORD = 'your_password'
DOMAIN = 'your_domain'  # Can be an empty string if there's no domain
```

```
# Local directory to back up
LOCAL_DIRECTORY = '/path/to/your/local/directory'

def backup_files_to_network_share(local_directory, network_share):
    # Register the session for the shared network
    register_session(network_share, username=USERNAME, password=PASSWORD, domain=DOMAIN)

    with SambaClient(server=network_share, username=USERNAME, password=PASSWORD, domain=DOMAIN) as client:
        for root, dirs, files in os.walk(local_directory):
            for file in files:
                local_file_path = os.path.join(root, file)
                # Construct the full path where the file will be copied on the network share
                remote_file_path = os.path.join(network_share, os.path.relpath(local_file_path, local_directory))
                print(f"Copying {local_file_path} to {remote_file_path}")
                # Create directories if they don't exist
                remote_dir = os.path.dirname(remote_file_path)
                if not client.exists(remote_dir):
                    client.makedirs(remote_dir)
                # Copy the file
                client.upload(local_file_path, remote_file_path)

def main():
    backup_files_to_network_share(LOCAL_DIRECTORY, NETWORK_SHARE_PATH)

if __name__ == "__main__":
    main()
```

For a backup script on a Windows environment, you can utilize Python's built-in libraries such as shutil for file copying. This example script will copy files from a local directory to a network share or another local directory (which can be a mounted network drive in Windows).

First, make sure the target network share is accessible from your Windows machine, and you have the necessary permissions. If it's a network share, you might need to map it to a drive letter on your Windows machine, or you can access it directly using its UNC path (like \\Server\SharedFolder).

```
import os
import shutil
from datetime import datetime

def backup_files_to_network_share(local_directory, network_share):
    """
    Copy files from the local directory to a network share or another directory.

    :param local_directory: Path to the local directory to back up.
    :param network_share: Network share path or path to the target directory.
    """
    for root, dirs, files in os.walk(local_directory):
        for file in files:
            local_file_path = os.path.join(root, file)
            relative_path = os.path.relpath(local_file_path, local_directory)
            network_file_path = os.path.join(network_share, relative_path)

            # Create the target directory if it doesn't exist
            os.makedirs(os.path.dirname(network_file_path), exist_ok=True)

            print(f"Copying {local_file_path} to {network_file_path}")
            shutil.copy2(local_file_path, network_file_path)

def main():
    local_directory = 'C:\\path\\to\\your\\local\\directory'
    network_share = 'Z:\\path\\to\\network\\share'  # Can be a mapped drive or UNC path

    print(f"Starting backup at {datetime.now()}")
    backup_files_to_network_share(local_directory, network_share)
    print(f"Backup completed at {datetime.now()}")

if __name__ == "__main__":
    main()
```

Replace C:\\path\\to\\your\\local\\directory with the path to your local directory and Z:\\path\\to\\network\\share with the path to your network share or another directory.

If you're using a UNC path directly (like \\Server\SharedFolder), make sure your script is running with credentials that have access to the network share.

To automate this script, you can use Windows Task Scheduler

Log File Analysis:

Parse server log files to extract useful information, such as error messages or access statistics, and generate summary reports.

Parsing Windows Event Logs to extract useful information like error messages or access statistics requires using Python's win32evtlog library, which is part of the pywin32 package. This library allows you to interact with the Windows Event Log system.

```
pip install pywin32
```

```
import win32evtlog
import win32evtlogutil
import win32security

def parse_windows_event_logs(server, log_type, query):
    """
    Parses Windows Event Logs.

    :param server: Name of the server/computer.
    :param log_type: Type of log to parse, e.g., 'System', 'Application'.
    :param query: Query string to filter logs, e.g., 'Error', 'Warning'.
    """
    # Connect to the event log
    handle = win32evtlog.OpenEventLog(server, log_type)

    try:
        flags = win32evtlog.EVENTLOG_BACKWARDS_READ | win32evtlog.EVENTLOG_SEQUENTIAL_READ
        total = 0

        while True:
            events = win32evtlog.ReadEventLog(handle, flags, 0)

            if not events:
                break

            for event in events:
                # Check if the event matches the query (e.g., error or warning)
                if query in win32evtlogutil.SafeFormatMessage(event, log_type):
                    total += 1
                    print("Event ID:", event.EventID)
                    print("Event Source:", event.SourceName)
                    print("Event Type:", event.EventType)
                    print("Event Category:", event.EventCategory)
                    print("Event Time:", event.TimeGenerated)
                    print("Event Message:", win32evtlogutil.SafeFormatMessage(event, log_type))
                    print("-" * 50)

        print(f"Total '{query}' events found: {total}")

    finally:
        win32evtlog.CloseEventLog(handle)

# Example usage
parse_windows_event_logs('localhost', 'System', 'Error')
```

Server Health Check:

Periodically check the health of the server by monitoring CPU usage, memory usage, and other vital metrics, and alerting if certain thresholds are exceeded.

Windows/Linux

```
import psutil
import time

# Thresholds
CPU_THRESHOLD = 85  # percent
MEMORY_THRESHOLD = 85  # percent
CHECK_INTERVAL = 60  # seconds

def check_system_health():
    while True:
        # Check CPU usage
        cpu_usage = psutil.cpu_percent(interval=1)
        if cpu_usage > CPU_THRESHOLD:
            print(f"Warning: High CPU usage detected: {cpu_usage}%")

        # Check memory usage
        memory_usage = psutil.virtual_memory().percent
        if memory_usage > MEMORY_THRESHOLD:
            print(f"Warning: High memory usage detected: {memory_usage}%")

        # Sleep for a while before checking again
        time.sleep(CHECK_INTERVAL)

if __name__ == "__main__":
    print("Starting system health monitoring...")
    check_system_health()
```

Batch File Renaming or Organization:

Organize and rename files in bulk on the server, especially useful for managing large numbers of files or directories.

For Windows:

```
import os

def bulk_rename_files(directory, prefix):
    """
    Rename files in the specified directory by adding a prefix to each file name.

    :param directory: Path to the directory containing the files to be renamed.
    :param prefix: The prefix to add to each file name.
    """
    for filename in os.listdir(directory):
        old_file_path = os.path.join(directory, filename)

        # Skip directories
        if os.path.isdir(old_file_path):
            continue

        new_filename = prefix + filename
        new_file_path = os.path.join(directory, new_filename)

        # Rename the file
        os.rename(old_file_path, new_file_path)
        print(f"Renamed '{old_file_path}' to '{new_file_path}'")

def main():
    directory = 'C:\\path\\to\\your\\directory'  # Update this path
    prefix = 'new_'  # Change this to your desired prefix

    bulk_rename_files(directory, prefix)
```

```
if __name__ == "__main__":
    main()
```

User Account Management:

Automate the process of creating, deleting, or modifying user accounts and permissions on the server.

Sample using pyad module (but you can also use powershell commands)

```
pip install pyad
```

```
from pyad import pyad, pyadset, pyaduser

# Configure your domain
pyad.set_defaults(ldap_server="ldap://your-ldap-server", username="your-username", password="your-password")

def create_user(username, password, ou_path):
    """
    Create a new user in Active Directory.
    """
    ou = pyad.adcontainer.ADContainer.from_dn(ou_path)
    new_user = pyad.aduser.ADUser.create(username, ou, password)
    print(f"User {username} created successfully.")

def delete_user(username):
    """
    Delete a user from Active Directory.
    """
    user = pyad.aduser.ADUser.from_cn(username)
    user.delete()
    print(f"User {username} deleted successfully.")

def modify_user(username, attribute, value):
    """
    Modify an attribute of a user in Active Directory.
    """
    user = pyad.aduser.ADUser.from_cn(username)
    user.update_attribute(attribute, value)
    print(f"User {username}'s {attribute} updated to {value}.")

# Example usage
create_user('newuser', 'Password123', 'ou=users,dc=example,dc=com')
modify_user('newuser', 'description', 'New user account')
delete_user('newuser')
```

Writing Python script output to file:

Solution 1:

```
output_data = "This is the output data."

# Using 'write'
with open('output.txt', 'w') as file:
    file.write(output_data)

# If you have multiple lines in a list
lines = ["First line\n", "Second line\n", "Third line\n"]

# Using 'writelines'
with open('output.txt', 'w') as file:
    file.writelines(lines)
```

Solution 2: Redirecting print()

```
with open('output.txt', 'w') as file:
    print("This will go to the file.", file=file)
```

Solution 3: Logging outputs

```
import logging

logging.basicConfig(filename='app.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s')
logging.warning('This will get logged to a file')
```

Solution 4: Using csv module

```
import csv

rows = [['Name', 'Age'], ['Alice', 24], ['Bob', 19]]

with open('output.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(rows)
```

Other Recommendations:

- **Automated Database Backups:**
  Create scripts to backup databases regularly, which can be critical for recovery in case of data loss.

- **Network Monitoring and Reporting:**
  Monitor network traffic and bandwidth usage, and generate reports to identify any unusual traffic patterns or potential threats.

- **Automated Security Scanning:**
  Regularly scan the server for vulnerabilities using tools like OpenVAS or Nessus, and report the findings for further action.

- **Automated Script for SSL/TLS Certificate Renewal:**
  Monitor SSL/TLS certificates on web servers and automatically renew them before they expire.