Introduction
to
Network Programming
Part II

Apinun Tunpan, Ph.D.
Internet Education and Research Lab
Asian Institute of Technology
September 2010
Today’s Outline

1. Some useful Python modules
2. Bluetooth programming
3. GPS device interface
4. Further self studies
1. Some useful Python modules
1.1 Date & time module (time)

```python
>>> import time
>>> dir( time )
['__doc__', '__name__', '__package__', 'accept2dyear', 'altzone', 'asctime', 'clock', 'ctime', 'daylight', 'gmtime', ' localtime', 'mktime', 'sleep', 'strftime', 'strptime', 'struct_time', 'time', 'timezone', 'tzset']

>>> time.gmtime()  # This returns a struct_time representing UTC time.
'time.struct_time(tm_year=2010, tm_mon=9, tm_mday=15, tm_hour=8, tm_min=6, tm_sec=3, tm_wday=2, tm_yday=258, tm_isdst=0)

>>> time.localtime()  # This returns a struct_time representing local time.
'time.struct_time(tm_year=2010, tm_mon=9, tm_mday=15, tm_hour=15, tm_min=6, tm_sec=7, tm_wday=2, tm_yday=258, tm_isdst=0)

>>> time.asctime( time.gmtime() )
'Wed Sep 15 08:06:17 2010'

>>> time.asctime( time.localtime() )
'Wed Sep 15 15:06:24 2010'

>>> time.ctime()  # time.ctime() returns a string showing local time.
'Wed Sep 15 15:06:29 2010'
```
time.sleep()

```python
def GoToSleep( n_sec ):
    print 'Going to sleep on ', time.ctime()
    time.sleep( n_sec )
    print 'Waking up on ', time.ctime()
```

```python
>>> GoToSleep( 12 )
Going to sleep on  Wed Sep 15 15:22:50 2010
Waking up on  Wed Sep 15 15:23:02 2010
>>> GoToSleep( 15 )
Going to sleep on  Wed Sep 15 15:23:13 2010
Waking up on  Wed Sep 15 15:23:28 2010
```
1.2 System module (sys)

# SysArgv.py : showing how to access command line parameters

```python
import sys
print sys.argv
```

```
interlab@ubuntu104:~/.PythonNetworking$ python SysArgv.py Network Programming is fun
['SysArgv.py', 'Network', 'Programming', 'is', 'fun']
interlab@ubuntu104:~/.PythonNetworking$
```
1.3 Operating system module (os)

```python
>>> import os
>>> os.getcwd()
'/home/interlab/PythonNetworking'
>>> os.listdir( '.' )
['helloworld.py', 'WebbrowserControl.py', 'SysArgv.py', 'RecvMsgUDP.py', 'RecvMsgTCP.py', 'Part II', 'speak.py', 'SimpleHTMLParser.py', 'SendMsgUDP.py', 'SendMsgTCP.py']
>>> os.stat( 'helloworld.py' )
posix.stat_result(st_mode=33188, st_ino=146737L, st_dev=2049L, st_nlink=1, st_uid=1000, st_gid=1000, st_size=20L, st_atime=1284451149, st_mtime=1284450857, st_ctime=1284450857)
>>> os.chdir( 'Part II' )
>>> os.getcwd()
'/home/interlab/PythonNetworking/Part II'
>>> os.mkdir( 'Section 2.1' )
>>> os.chdir( 'Section 2.1' )
>>> os.getcwd()
'/home/interlab/PythonNetworking/Part II/Section 2.1'
```

- Get the current working directory
- List the contents of the current directory
- Get the information of a specific file
- Change the current working directory
- Create a new directory and change the current directory to it.
1.3 Operating system module (os) continues

```python
>>> import os
>>> p = os.popen('ls')
>>> result = p.read()
>>> result
'helloworld.py\nPart II\nRecvMesgTCP.py\nRecvMesgUDP.py\nSendMesgTCP.py
SendMesgUDP.py
SimpleHTMLParser.py
speak.py
SysArgv.py
WebbrowerControl.py\n'
>>> print(result)
helloworld.py
Part II
RecvMesgTCP.py
RecvMesgUDP.py
SendMesgTCP.py
SendMesgUDP.py
SimpleHTMLParser.py
speak.py
SysArgv.py
WebbrowerControl.py
```

Open a pipe and execute the `/bin/ls` Linux command. Then read the result from the opened pipe.

The result is in fact a string received from the output of the `/bin/ls` command. Entries of the result are separated by `\n` (the newline character).

If we print this result, it looks nice. But do not forget that it is still a string which can be very long.

```python
>>> lines = result.split(' \n')
>>> lines
['helloworld.py', 'Part II', 'RecvMesgTCP.py', 'RecvMesgUDP.py', 'SendMesgTCP.py', 'SendMesgUDP.py', 'SimpleHTMLParser.py', 'speak.py', 'SysArgv.py', 'WebbrowerControl.py', '']
```

`str.split()` can break down a string into a list. Here we use `\n` as a field separator.
Exercise

• The iwlist command in Linux can scan for WiFi access points and show their characteristics (e.g. channels, cell ids, signal strengths). Write a Python program to read the whole output of the iwlist command (e.g. from “iwlist scan”) and keep it as a (very long) string.
Solving a problem

• We are doing a Vehicle-to-Infrastructure (V-2-I) communication project. Suppose that our first step is to monitor and record WiFi access points, their channels and signal strengths, along the road while we drive. Can we build a simple automated tool for this purpose?

We shall revisit this problem after we learn more about Regular Expressions.
1.4 Regular expression (re) module

• Matching or searching for a specific pattern (specified by a regular expression)
• Substituting one or more occurrences of a pattern (specified by a regular expression)
• Splitting the string based on a regular expression
Common uses of regular expressions

- `Re.match()` = match if the pattern occurs at the beginning of a string
- `Re.search()` = search for the first occurrence of a pattern in the whole string
- `Re.findall()` = search for every occurrence of a pattern in the whole string
- `Re.sub()` = replace all occurrences of the pattern with a different string.
re.match() vs. re.search()

```python
>>> import re
>>> result1 = re.match( 'aa', 'aabbccddeeffaabbccddeeff' )
>>> print result1
<_sre.SRE_Match object at 0x937aa30>
>>> result2 = re.match( 'bb', 'aabbccddeeffaabbccddeeff' )
>>> print result2
None
>>> result3 = re.search( 'bb', 'aabbccddeeffaabbccddeeff' )
>>> print result3
<_sre.SRE_Match object at 0x937ae90>
>>> result1.span()
(0, 2)
>>> result3.span()
(2, 4)
>>> result3.start()
2
>>> result3.end()
4
```

- re.match() sees if the pattern occurs at the beginning of a string.
- Pattern ‘bb’ does not occur at the beginning.
- Re.search() looks for the pattern inside the whole string.

When we look into a search result, we can see the index range where the pattern occurs in the string.

Note that re.search() returns only the first occurrence.
re.findall() vs. re.finditer()

```python
>>> import re
>>> result4 = re.findall( 'cc', 'aabbccddeeffaabbccddeeff' )
>>> print result4
['cc', 'cc']
>>> iterator = re.finditer( 'cc', 'aabbccddeeffaabbccddeeff' )
>>> for i in iterator:
    print i.span()
```

re.findall() returns all occurrences of the matches

re.finditer() returns an iterator on which you can run a loop (e.g. to find the spans of the occurrences)

Here ‘cc’ occurs twice, at index 4 and index 16.
Example

```python
>>> import re
>>> txt = "John actively runs a science project. He probably discovers a new method."
>>> re.findall( '\w+ly', txt )
['actively', 'probably']
```

\w matches any alphanumeric character and the underscore (equivalent to [a-zA-Z0-9_] )

+ means one or more occurrences of the pattern ahead of it.

\w+ly then means one or more of \w instances which end with ly

Adapted from Python documentation: http://docs.python.org/howto/regex.html
Example

```python
>>> import re
>>> txt = "John actively runs a science project. He probably discovers a new method."
>>> re.findall( '^[rp]\w+', txt )
['runs', 'project', 'probably', 'rs']

\[rp\] matches r or p.
\w+ matches one or more of any alphanumeric or the underscore.

>>> re.findall( '\s+[rp]\w+', txt )
['runs', ' project', ' probably']
\s+ matches one or more of whitespaces
```

Adapted from Python documentation: http://docs.python.org/howto/regex.html
re.sub() : pattern substitution

```python
>>> import re
>>> text = 'This\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\...
re.split() : splitting a string using RE

```python
>>> import re
>>> text = 'This	\tis\fa::very\v, weird\n\tstring.'
>>> text
'This\t\tis\x0ca::very\x0b, weird\n\tstring.'
>>> print text
This\sis|a::very|, weird
    string.
>>> re.split( '\[s:, .]+', text )
['This', 'is', 'a', 'very', 'weird', 'string', '']
```
Further references on regular expressions

• http://docs.python.org/howto/regex.html
• http://docs.python.org/library/re.html
Revisiting our problem

• We are doing a Vehicle-to-Infrastructure (V-2-I) communication research. Suppose that our first step is to monitor and record WiFi access points, their channels and signal strengths, along the road while we drive. Can we build a simple automated tool for this purpose?
One problem of RE – we can easily overdo it..

This was supposed to be a string: “Interlab AP-1B” but it got split!!
# ApScanV1.py : access point scanning

```python
import os

txt = os.popen( 'iwlist scan' ).read()
lines = txt.split( '\n' )
xlines = map( lambda i : i.strip(), lines )
print xlines
```
Well, it seems good, but we’ll need some re-formatting effort

Apinun Tunpan, IntERLab
# ApScanV2.py : access point scanning
WLAN = 'ra0'
import os, re, time

def ParseLines( lines ):
    cells = dict()
    c = None
    for l in lines:
        if re.match( '^Cell', l ) is not None:
            c = l.split( ' ' )[4]
            cells[c] = dict()
            continue
        if c is not None and re.match( '^ESSID', l ) is not None:
            essid = l.split( '::' )[1]
            cells[c][ 'ESSID' ] = essid
            continue
        if c is not None and re.match( '^Quality', l ) is not None:
            qsn = re.sub( 'Quality[:=]', 'Q=', l )
            qsn = re.sub( 'Signal Level[:=]', 'S=', qsn )
            qsn = re.sub( 'Noise level[:=]', 'N=', qsn )
            qsn = re.sub( '\s+', ' ', qsn )
            cells[c][ 'QSN' ] = qsn
            continue
        if c is not None and re.match( '^Channel', l ) is not None:
            ch = re.sub( 'Channel[:=]', '', l )
            cells[c][ 'CH' ] = ch
            continue
    return cells
```python
def DoScan():
    txt = os.popen('iwlist ' + WLAN + ' scan').read()
    lines = txt.split('\n')
    xlines = map(lambda i: i.strip(), lines)
    cells = ParseLines(xlines)
    fmt = '%-17s | %-2s | %-30s | %s'

    print '\n' + time.ctime()
    print fmt % ('Cell', 'CH', '(Q)uality & (S)ignal & (N)oise', 'ESSID')

    for c in cells.keys():
        print fmt%(c, cells[c].get('CH'), cells[c].get('QSN'), cells[c].get('ESSID'))

if __name__ == '__main__':
    while True:
        DoScan()
        time.sleep(3)
```

Apinun Tunpan, IntERLab
You can move your laptop around and observe signal quality.
Exercise

• Extend ApScanV2.py so that it can display
  – Network mode (“managed” vs. “ad-hoc”)
  – Whether the network is encrypted
Exercise

• Recall the HTML parser example from last time. We would like to make it smarter.

Say, we would like to have the lists of oil prices, kept and indexed in a dictionary by their dates.
2. Bluetooth Programming
Bluetooth basics

• Personal Area Network (PAN)  2.402-2.480 GHz ISM

• There are 3 classes of Bluetooth devices:
  – Class 1: Max power 100mW  (range ~100 meters)
  – Class 2: Max power  2.5mW    (range ~10 meters)
  – Class 3: Max power   1mW          (range ~1 meter)

• There are several versions of Bluetooth
  – Bluetooth v1.2     (~1Mbps)
  – Bluetooth v2.0+EDR (~3 Mbps)
  – Bluetooth v2.1+EDR (e.g. “secure simple pairing”, SSP)
  – Bluetooth v3.0+HS  ( ~24Mbps)
Bluetooth modules for Python

• Bluez (Pybluez)
  – Homepage: http://www.bluez.org/
  – Ubuntu: sudo apt-get install bluez python-bluez
  – Tutorial:
    • http://people.csail.mit.edu/albert/bluez-intro/

• Lightblue
  – Homepage: http://lightblue.sourceforge.net/
  – Ubuntu: sudo apt-get install python-lightblue
  – Tutorial: See the lightblue’s homepage
lightblue.finddevices() : search for nearby BT devices

```python
>>> import lightblue
>>> devices = lightblue.finddevices()
>>> devices
[('00:1B:C1:02:F0:8B', 'HOLUX_M-1000', 7936), ('00:21:FC:FC:F5:91', 'Good Old Nokia', 5898756)]

>>> for d in devices:
    print(d)

('00:1B:C1:02:F0:8B', 'HOLUX_M-1000', 7936)
('00:21:FC:FC:F5:91', 'Good Old Nokia', 5898756)
```
lightblue.findservices() : search for nearby BT services

```python
>>> import lightblue

>>> services = lightblue.findservices()

>>> services


>>> for s in services:
...    print s

('00:1B:C1:02:F0:8B', 1, 'SPP slave')
('00:21:FC:FC:F5:91', 1, 'Dial-up networking')
('00:21:FC:FC:F5:91', 15, 'Nokia PC Suite')
('00:21:FC:FC:F5:91', 3, 'COM 1')
('00:21:FC:FC:F5:91', 13, 'Voice Gateway')
('00:21:FC:FC:F5:91', 12, 'Audio Gateway')

The SPP slave service is available on Holux M-1000 GPS
These are the services available on my Nokia phone

Apinun Tunpan, IntERLab
lightblue.selectdevice() : device selection GUI

Note: this program may freeze IDLE. You should run it from linux command line.

Apinun Tunpan, IntERLab
lightblue.selectservice() : service selection GUI

Note: this program may freeze IDLE. You should run it from linux command line.
Sending objects via Object Exchange (OBEX)
Object Exchange (OBEX) Push Example

```python
# LightblueSend.py: Apinun Tunpan, intERLab, AIT

import lightblue

print "Finding Devices.."
device = lightblue.selectdevice()

if device is not None:
    print "Device selected was: ", device
else:
    print "No device was selected. Exiting.."
    exit()

obex_push = None
print "Finding OBEX Object Push service..."

for s in lightblue.findservices( addr = device[0] ):
    print s
    if s[2] == 'OBEX Object Push':
        obex_push = s[1]

if obex_push is not None:
    client = lightblue.obex.OBEXClient( str(device[0]), obex_push )
    print client.connect()
    print client.put( {"name":"Mmyfile.jpg"}, file("Mmyfile.jpg","rb" ) )
    client.disconnect()
else:
    print "OBEX Object Push service was not listed on the device. Exiting."```
Exercise

• Implement a program to receive a file from mobile phone via OBEX push
3. GPS device interface
Bluetooth GPS
Ideas for interfacing with GPS

• Connect (e.g. via bluetooth) to the GPS
• Read data lines
• Get to the right line (e.g. $GPGGA)
• Split & Decode the line
  – To get UTC, Lat, Lon, Height, Validity...
• Disconnect

Apinun Tunpan, IntERLab
Connecting with a Bluetooth device

• This time we need to transfer stream of data
• There are two protocols:
  – RFCOMM : reliable, stream-based
  – L2CAP : best-effort, datagram

More information at
http://people.csail.mit.edu/albert/bluez-intro/x95.html
The bluez (bluetooth) module

```python
>>> import bluetooth
>>> services = bluetooth.find_service()
>>> for s in services:
    print s
```
import sys, time, bluetooth

deviceAddress = '00:1B:C1:02:F0:8B' # Change this line to your device's MAC

def main():

    # Find RFCOMM port
    services = bluetooth.find_service(address=deviceAddress)
    port = None
    for svc in services:
        if svc["name"] == "SPP slave":
            port = svc["port"]
            break

    if port is None:
        print "Could not find RFCOMM port for SPP slave service."
        sys.exit(4)

    # Create bluetooth socket
    bluesock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
    bluesock.connect((deviceAddress, port))

    while True:
        gpsdata = bluesock.recv(2048)
        gpslines = gpsdata.splitlines()
        print '*************************** Printing raw GPS data: *******************',
        for l in gpslines:
            print l
        time.sleep( 1 )

if __name__ == '__main__':
    main()
Sample output from GPSBlue1.py

```plaintext
$GPGGA,042230.000,1404.6657,N,10036.7734,E,1,10,0.88,34.5,M,-28.0,M,,*7F
$GPGSA,A,3,23,17,03,19,07,20,13,28,11,08,,1.51,0.88,1.23*0E
$GPGSV,4,1,14,13,56,236,33,19,51,024,28,11,50,169,34,07,44,343,36*75
$GPGSV,4,2,14,23,41,192,38,24,30,078,,03,20,035,35,28,13,290,25*70
$GPGSV,4,3,14,08,12,326,18,17,11,226,23,20,08,171,24,06,07,039,*7F
$GPGSV,4,4,14,32,02,154,,45,,,*4E
$GPRMC,042230.000,A,1404.6657,N,10036.7734,E,0.00,268.33,210910,,A*6E
$GPZDA,042230.000,21,09,2010,,*58

$GPGGA,042231.000,1404.6657,N,10036.7734,E,1,10,0.88,34.5,M,-28.0,M,,*7E
$GPGSA,A,3,23,17,03,19,07,20,13,28,11,08,,1.51,0.88,1.23*0E
$GPRMC,042231.000,A,1404.6657,N,10036.7734,E,0.01,268.33,210910,,A*6E
$GPZDA,042231.000,21,09,2010,,*59

$GPGGA,042232.000,1404.6657,N,10036.7734,E,1,10,0.88,34.5,M,-28.0,M,,*7D
$GPGSA,A,3,23,17,03,19,07,20,13,28,11,08,,1.51,0.88,1.23*0E
$GPRMC,042232.000,A,1404.6657,N,10036.7734,E,0.01,268.33,210910,,A*6D
$GPZDA,042232.000,21,09,2010,,*5A

$GPGGA,042233.000,1404.6657,N,10036.7733,E,1,10,0.88,34.5,M,-28.0,M,,*7B
$GPGSA,A,3,23,17,03,19,07,20,13,28,11,08,,1.51,0.88,1.23*0E
$GPRMC,042233.000,A,1404.6657,N,10036.7733,E,0.01,268.33,210910,,A*6B
$GPZDA,042233.000,21,09,2010,,*5B
```
Some of NMEA GPS data types

- `$GPGAA` – Fix information (2D or 3D)
- `$GPGSA` – Satellite status
- `$GPGSV` – Satellites in view
- `$GPRMC` – Recommended minimum
- `$GPZDA` -- UTC date and time

For more information, search Google™, or visit: http://www.gpsinformation.org/dale/nmea.htm
Understanding the lat/lon formats

• Common formats
  – Decimal degrees (e.g. 100.12345)
  – Degrees, decimal minutes (e.g. 100 7.407’)
  – Degrees, minutes, decimal seconds (e.g. 100 7’ 24.42”)

With a straightforward conversion:

Decimal degrees

  = Degrees + minutes/60 + seconds/3600
import sys, time, bluetooth
deviceAddress = '00:1B:C1:02:F0:8B'  # Change this line to your device's MAC

def DecodeGPS(line):
    'From a $GPGGA sentence, decode lat, lon, alt and utc'
    fields = line.split(',,')
    if fields[0] != '$GPGGA' or len(fields) < 6 or int(fields[6]) < 1:
        return None
    else:
        utc = float(fields[1])

    lat_deg = int(float(fields[2]))/100
    lat_min = (float(fields[2]) - lat_deg*100)/60.0
    lat = lat_deg + lat_min

    lon_deg = int(float(fields[4]))/100
    lon_min = (float(fields[4]) - lon_deg*100)/60.0
    lon = lon_deg + lon_min

    if fields[3] != 'N': lat = lat * -1  # Adjust for the sign
    if fields[5] != 'E': lon = lon * -1  # Adjust for the sign

    altitude = float(fields[9])
    quality = float(fields[6])
    return (utc, lat, lon, altitude, quality)

Transforming the format of lat, lon from DDMM.MMMM to DD.DDDDD
Altitude
Quality of the GPS fix.
def main():
    # Find RFCOMM port
    services = bluetooth.find_service(address=deviceAddress)
    port = None
    for svc in services:
        if svc["name"] == "SPP slave."
            port = svc["port"]
            break

    if port is None:
        print "Could not find RFCOMM port for SPP slave service."
        sys.exit(4)

    # Create bluetooth socket
    bluesock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
    bluesock.connect((deviceAddress, port))

    while True:
        gpsdata = bluesock.recv(2048)
        gpslines = gpsdata.splitlines()
        for l in gpslines:
            data = DecodeGPS(l)
            if data is not None:
                msg="utc= %s lat= %f lon= %f alt= %s q= %s" % data
                print msg
                time.sleep(1)

if __name__ == '__main__':
    main()
<table>
<thead>
<tr>
<th>utc</th>
<th>lat</th>
<th>lon</th>
<th>alt</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>080557.000</td>
<td>14.077713</td>
<td>100.612975</td>
<td>13.5</td>
<td>1</td>
</tr>
<tr>
<td>080558.000</td>
<td>14.077715</td>
<td>100.612975</td>
<td>13.4</td>
<td>1</td>
</tr>
<tr>
<td>080559.000</td>
<td>14.077717</td>
<td>100.612970</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080600.000</td>
<td>14.077720</td>
<td>100.612972</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080601.000</td>
<td>14.077725</td>
<td>100.612963</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080602.000</td>
<td>14.077725</td>
<td>100.612965</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080603.000</td>
<td>14.077727</td>
<td>100.612968</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080604.000</td>
<td>14.077728</td>
<td>100.612968</td>
<td>13.2</td>
<td>1</td>
</tr>
<tr>
<td>080605.000</td>
<td>14.077732</td>
<td>100.612958</td>
<td>13.1</td>
<td>1</td>
</tr>
<tr>
<td>080606.000</td>
<td>14.077730</td>
<td>100.612947</td>
<td>13.0</td>
<td>1</td>
</tr>
<tr>
<td>080607.000</td>
<td>14.077730</td>
<td>100.612940</td>
<td>12.9</td>
<td>1</td>
</tr>
<tr>
<td>080608.000</td>
<td>14.077730</td>
<td>100.612935</td>
<td>12.9</td>
<td>1</td>
</tr>
<tr>
<td>080609.000</td>
<td>14.077732</td>
<td>100.612923</td>
<td>12.8</td>
<td>1</td>
</tr>
<tr>
<td>080610.000</td>
<td>14.077732</td>
<td>100.612918</td>
<td>12.7</td>
<td>1</td>
</tr>
<tr>
<td>080611.000</td>
<td>14.077733</td>
<td>100.612910</td>
<td>12.7</td>
<td>1</td>
</tr>
<tr>
<td>080612.000</td>
<td>14.077735</td>
<td>100.612903</td>
<td>12.4</td>
<td>1</td>
</tr>
<tr>
<td>080613.000</td>
<td>14.077735</td>
<td>100.612905</td>
<td>12.4</td>
<td>1</td>
</tr>
<tr>
<td>080614.000</td>
<td>14.077735</td>
<td>100.612907</td>
<td>12.3</td>
<td>1</td>
</tr>
<tr>
<td>080615.000</td>
<td>14.077735</td>
<td>100.612908</td>
<td>12.3</td>
<td>1</td>
</tr>
<tr>
<td>080616.000</td>
<td>14.077732</td>
<td>100.612910</td>
<td>12.2</td>
<td>1</td>
</tr>
<tr>
<td>080617.000</td>
<td>14.077730</td>
<td>100.612910</td>
<td>12.2</td>
<td>1</td>
</tr>
<tr>
<td>080618.000</td>
<td>14.077728</td>
<td>100.612912</td>
<td>12.1</td>
<td>1</td>
</tr>
<tr>
<td>080619.000</td>
<td>14.077713</td>
<td>100.612913</td>
<td>12.1</td>
<td>1</td>
</tr>
</tbody>
</table>
Exercises

• Extend GPSBlue.py so that it displays the current location of your PC + GPS on Google Maps™

• Add GPS coordinate reading to the Access Point scan (ApScanV2.py) so that it displays the lat/lon of the reading.
4. Further self studies
What makes Python full of useful modules and libraries?

• The answer: language wrappers and bindings.

• [http://www.swig.org/exec.html](http://www.swig.org/exec.html)
For your research

Python may already have the modules that make your life easier:

• Graph Theory
  – http://networkx.lanl.gov/
  – http://code.google.com/p/python-graph/

• Scientific Computing & Optimization
  – http://www.scipy.org/
  – http://mdp-toolkit.sourceforge.net/
  – http://cvxmod.net/
For your research

• GIS
  – http://trac.gispython.org/lab
  – http://trac.gispython.org/lab/wiki/OwsLib
  – http://trac.osgeo.org/gdal/wiki/GdalOgrInPython

• Social Network
  – http://code.google.com/p/python-twitter/
  – http://code.google.com/p/pyfacebook/
Thank you for your attention

Hope you have fun and learn a lot from this class.