Lab10-Python Internet Services

Embedded Systems

TCP/IP Services

- We will cover the following topics:
  - TCP/IP and TCP servers for remote hardware access
  - HTTP servers for remote hardware access

- Software implementing TCP/IP typically follows a server/client model.
- The TCP server process has an open socket listening on a predefined port, and the client processes make connections to the server using its IP address and that port. For example,
  - a TCP server, with IP address 192.168.7.2, listening on port 8080.
  - A client could then open a connection to the server, which would create a new socket on the server computer, characterized by the local address, 192.168.7.2:8080 (it is a common notation to append the port to the address, separated by a colon), as well as the client computers IP address and port.
- If a second client then opens a connection to the server, since the second client has a different address (even if it's the same IP address, it will have a different port number), a new socket will be created on the server computer.

Python TCP Server

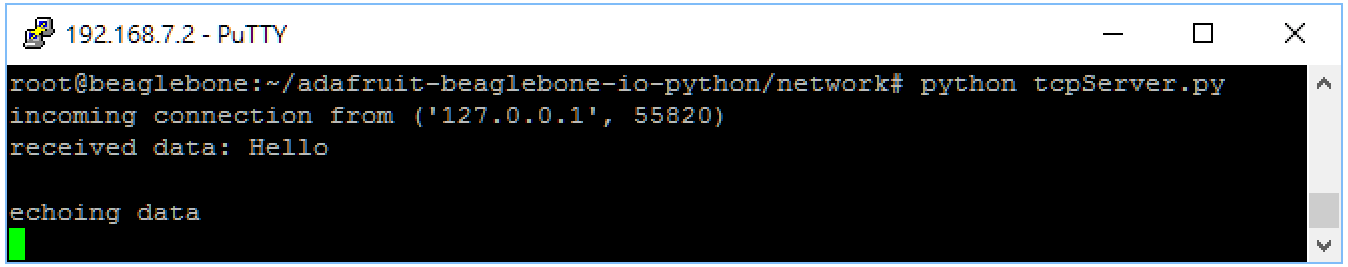
- Python's built-in socket library provides an API for using these socket interfaces. Let's look at an example of a simple TCP server:

```
import socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("", 8080))
server.listen(5)
while True:
    client, address = server.accept()
    print "incoming connection from", address
    data = client.recv(1024)
    if data:
        print "received data: {}".format(data)
        print "echoing data”
        client.send(data)
client.close()
```

Telnet TCP Client

- With your server running, you can use the telnet command line tool to connect to it as a client.

  # netcat localhost 8080
- You can then start typing characters into telent client, and when you press Enter it will send the string in a TCP/IP packet to the server:
- TCP Server side:

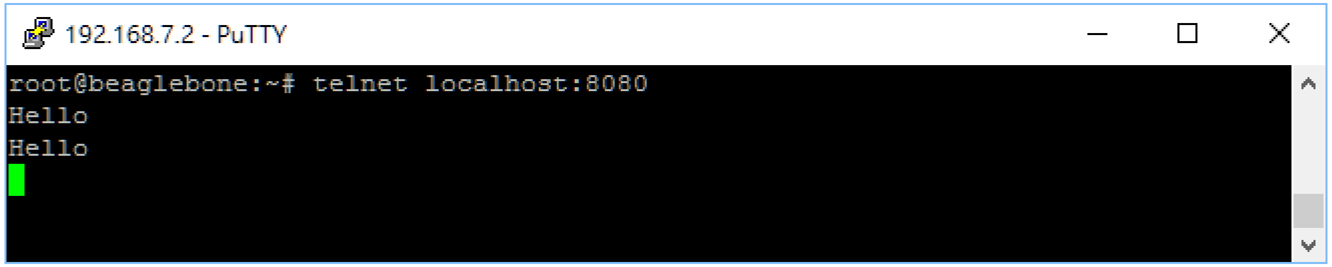


192.168.7.2 - PuTTY

```
root@beaglebone:~/adafruit-beaglebone-io-python/network# python tcpServer.py
incoming connection from ('127.0.0.1', 55820)
received data: Hello

echoing data
```

- Telnet Client side:

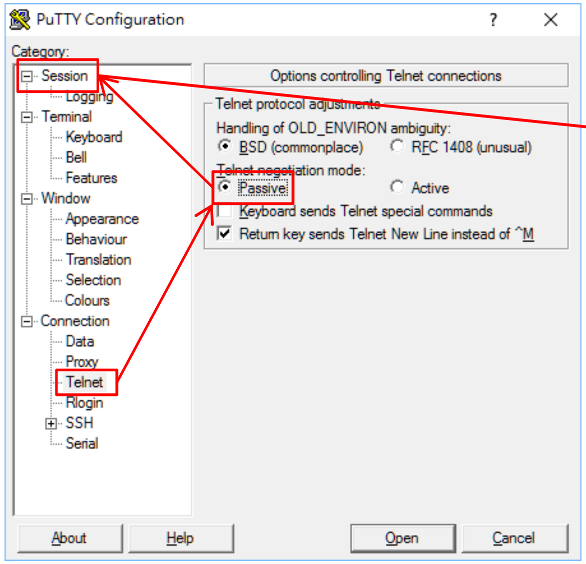


192.168.7.2 - PuTTY

```
root@beaglebone:~# telnet localhost:8080
Hello
Hello
```

Putty TCP Client

- You can also use Telnet client from another Linux machine on the same network by replacing 'localhost' with your BeagleBone's IP address.
- You can use PuTTY to connect from a Windows machine on the same network by setting it to use Telnet in passive mode (meaning it won't send any data until you've pressed Enter).
- Select the Telnet section under Connection and select the Passive option:



PuTTY Configuration

Options controlling Telnet connections

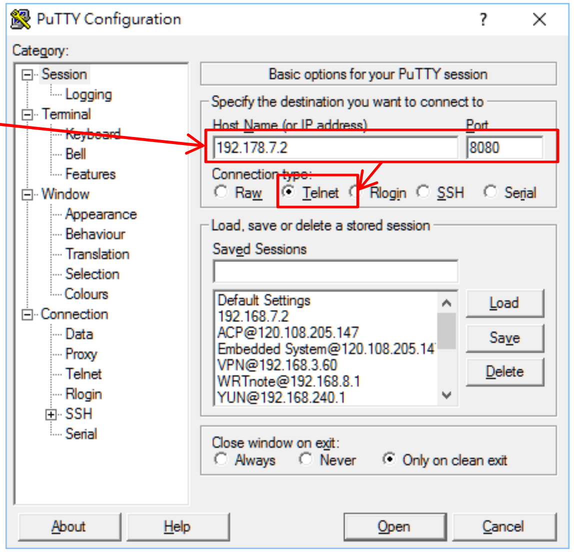
Telnet protocol adjustments

Handling of OLD_ENVIRON ambiguity: ● BSD (commonplace) ○ RFC 1408 (unusual)

Telnet negotiation mode: ● Passive ○ Active

□ Keyboard sends Telnet special commands

☑ Return key sends Telnet New Line instead of ^M



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.178.7.2

Port: 8080

Connection type: ○ Raw ● Telnet ○ Rlogin ○ SSH ○ Serial

Load, save or delete a stored session

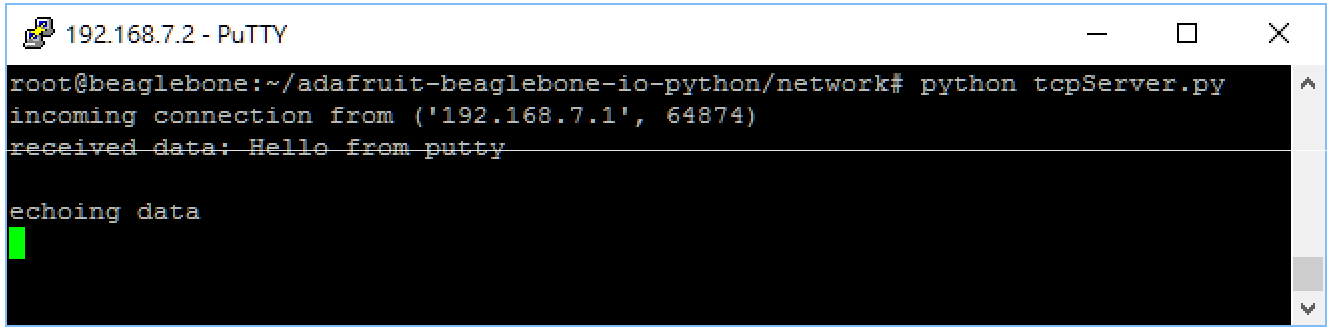
Saved Sessions

Default Settings
192.168.7.2
ACP@120.108.205.147
Embedded System@120.108.205.14
VPN@192.168.3.60
WRTnote@192.168.8.1
YUN@192.168.240.1

Close window on exit: ○ Always ○ Never ● Only on clean exit

Putty TCP Client

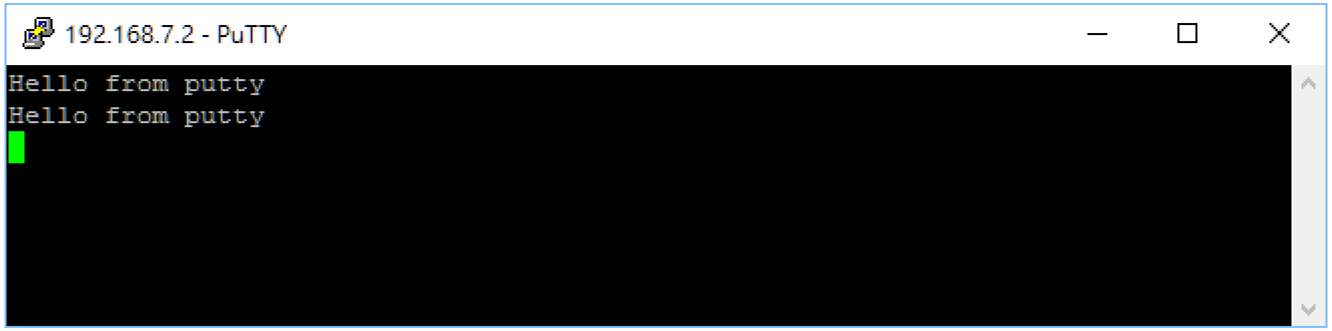
- When you press Open, you'll see a blank terminal window that you can type into, and when you press Enter, it will send the data to the server.
- TCP Server side:



```
root@beaglebone:~/adafruit-beaglebone-io-python/network# python tcpServer.py
incoming connection from ('192.168.7.1', 64874)
received data: Hello from putty

echoing data
```

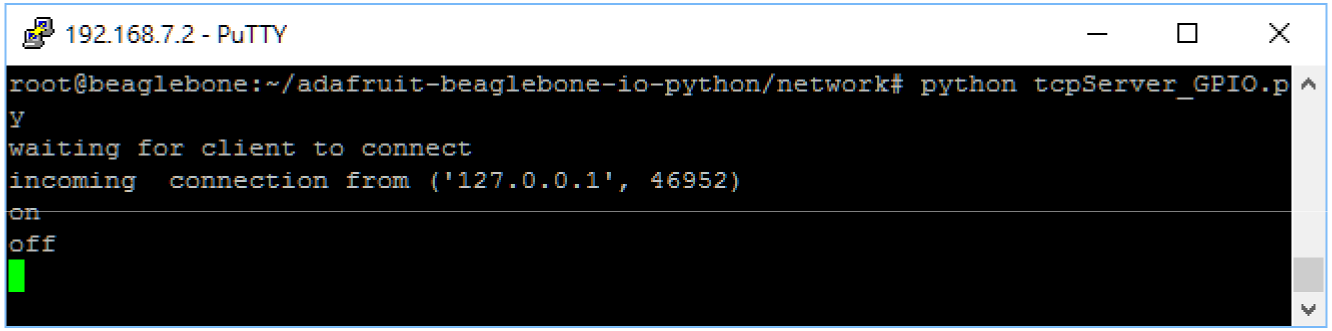
- Putty TCP Client :



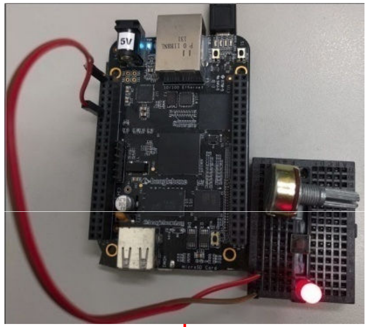
```
Hello from putty
Hello from putty
```

TCP server with remote hardware control

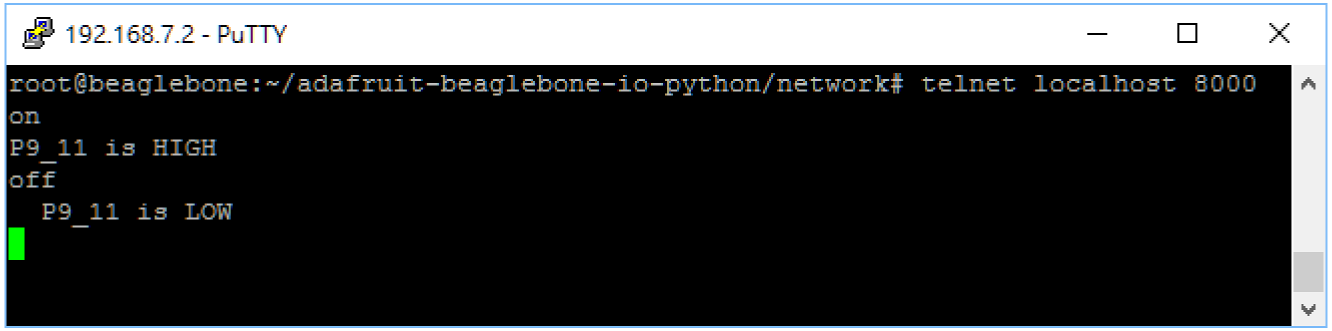
- Now, let's extend the TCP server to allow remote hardware control.
- If you connect to this server remotely using Telnet client or PuTTY, you can send the string on/off to on/off an LED.
- TCP server side:



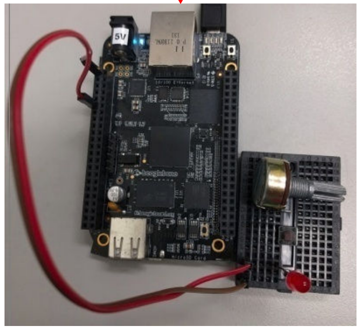
```
root@beaglebone:~/adafruit-beaglebone-io-python/network# python tcpServer_GPIO.p
y
waiting for client to connect
incoming  connection from ('127.0.0.1', 46952)
on
off
```



- Telnet client side:



```
root@beaglebone:~/adafruit-beaglebone-io-python/network# telnet localhost 8000
on
P9_11 is HIGH
off
  P9_11 is LOW
```



TCP server with remote hardware control

```
import socket
import Adafruit_BBIO.GPIO as GPIO
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("", 8000))
server.listen(5)
GPIO.setup("P9_11", GPIO.OUT)
while True:
    print "waiting for client to connect"
    client, address = server.accept()
    print "incoming  connection from", address
    connected = True
    while connected:
        data = client.recv(1024)
        if data:
            data = data.strip()
            print(data)
            if data == "":
                # empty string sent, close connection
                print "closing connection"
                client.close()
                break
```

TCP server with remote hardware control

```
elif data == "on":
                        GPIO.output("P9_11", GPIO.HIGH)
                        client.send("P9_11 is HIGH\r\n")
                elif data == "off":
                        GPIO.output("P9_11", GPIO.LOW)
                        client.send("  P9_11 is LOW\r\n")
                else:   # client no longer connected
                        connected = False
  client.close()
  print "connection closed"
```

Exercise

- Please modify the example codeyou can send the string getA0 string to get the voltage on the AIN0 pin when you connect to Beaglebone Black TCP server remotely using netcat or PuTTY.

HTTP Server in Python

- Now, we are going to create a simple Python CGI Server.
- To create a directory for our HTTP server and CGI scripts to reside in.
- It simply created a directory named ‘http’ in the root directory, but you may name it anything you wish and place it anywhere in your file system.
- The next step is to create a simple CGI server. We can create the script by WinSCP text editor and write the program below.

http.py

```
#!/usr/bin/env python

import BaseHTTPServer
import CGIHTTPServer
import cgitb; cgitb.enable()  ## This line enables CGI error reporting

server = BaseHTTPServer.HTTPServer
handler = CGIHTTPServer.CGIHTTPRequestHandler
server_address = ("", 8888)
handler.cgi_directories = ["/"]

httpd = server(server_address, handler)
httpd.serve_forever()
```

- Save the python script in the ‘http’ directory as ‘http.py’, and give it executable permissions. On Linux systems this is achieved with the command:
  - $ cd ~/http
  - $ chmod +x http.py
- Now that we have a server ready to handle the HTTP request from user. Next, we will create a simple “Hello, I am the homepage!” CGI script.
- We can create the script by WinSCP text editor and write the program below.

index.py

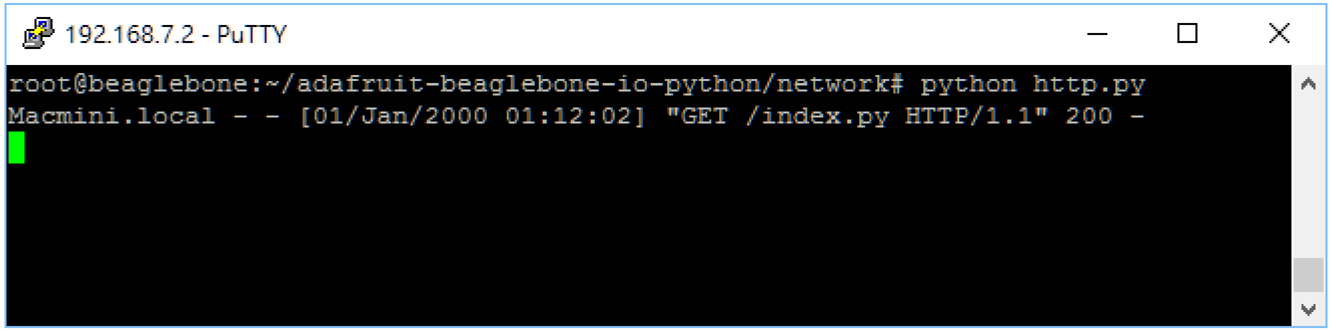
```
#!/usr/bin/env python
print "Content-type: text/html"
print
print "<title>Test CGI</title>"
print "<p>Hello, I am the homepage!</p>"
```

Starting the HTTP Server in BeagleBone Black

- Save this file in the ‘http’ directory as ‘index.py’ and give it executable permissions, just like the http.py file.
  - $ cd ~/http
  - $ chmod +x index.py
- To start your Python CGI server simply open up a terminal and cd into your ‘http’ directory. When you are there simply type the following command.
  - $ ./http.py
- Now, your HTTP server is now fully operational. To see your first page fire in the PC browser and type the following into the location bar.
  - http://192.168.7.2:88888/index.py

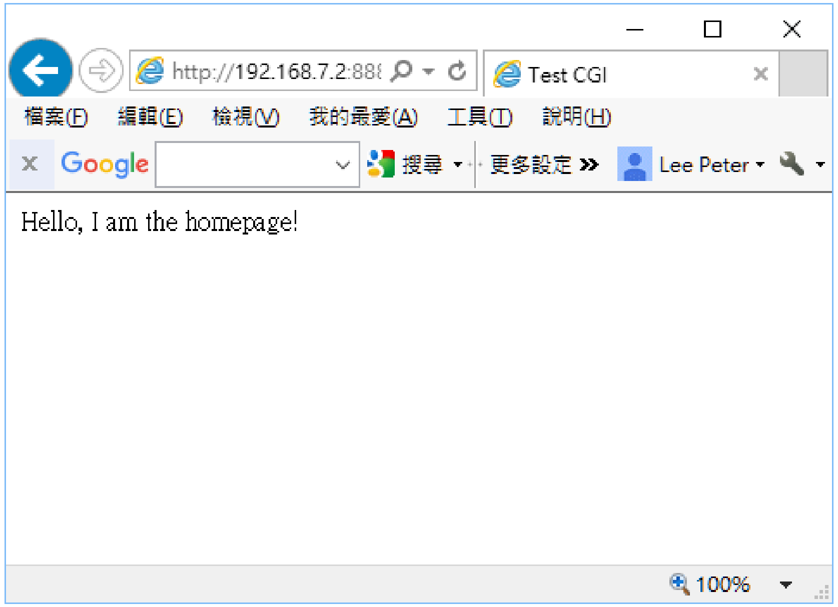
Access the HTTP Server from PC

- HTTP Server side:



```
root@beaglebone:~/adafruit-beaglebone-io-python/network# python http.py
Macmini.local - - [01/Jan/2000 01:12:02] "GET /index.py HTTP/1.1" 200 -
```

- HTTP Client Browser



Hello, I am the homepage!

- Adafruit_BBIO does not includes a library for HTTP services.
- However, we can write an API for creating simple HTML pages for web based user interfaces using HTTP Libraries in Python. Let's run a simple example:

ADC_output.py

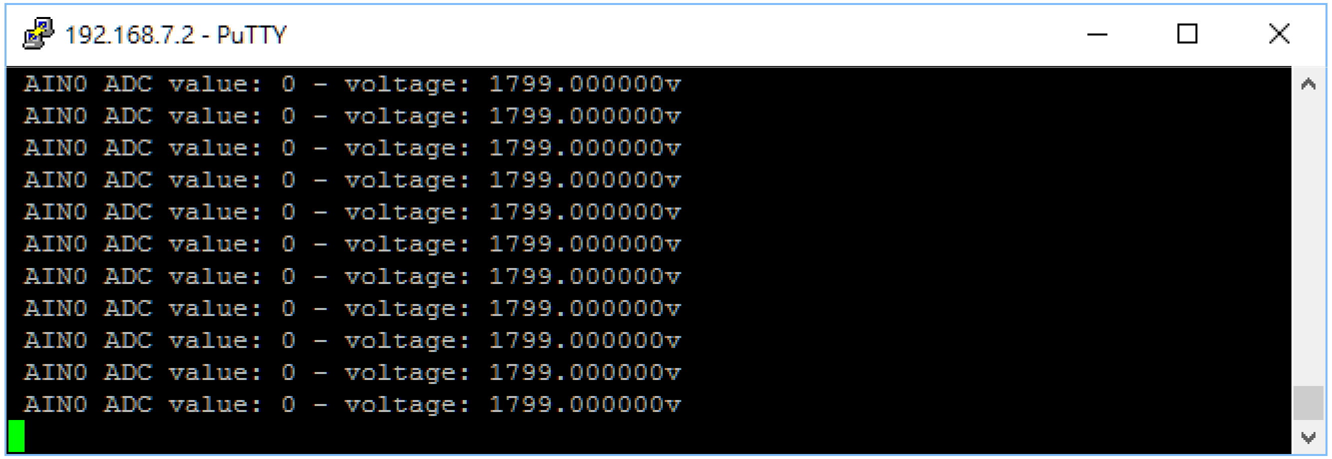
- We can create the script by WinSCP text editor and write the program below.

```
#!/usr/bin/python
import Adafruit_BBIO.ADC as ADC
import time
ADC.setup()
while True:
        #read returns values 0-1.0
        val1 = ADC.read("P9_39")
        raw1 = ADC.read_raw("P9_39")
        str1 = " AIN0 ADC value: %i - voltage: %fv" % (val1, raw1)
        print str1
        file_out = open("ADC.txt", "w")
        file_out.write(str1)
        file_out.close()
        time.sleep(5)
```

- Save the python script in the ‘http’ directory as ‘ADC_output.py’, and give it executable permissions.

ADC_output.py

- To start your Python ADC API in the server side simply open up a terminal and cd into your ‘http’ directory. When you are there simply type the following command:
  - $ python ADC_output.py &



```
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
AIN0 ADC value: 0 - voltage: 1799.000000v
```

http_ADC.py

```
#!/usr/bin/python
import Adafruit_BBIO.ADC as ADC
print "Content-type: text/html"
print
print "<title>Test HTTP ADC</title>"
try:
        file_in = open("ADC.txt", "r")
        ADC = file_in.readline()     # it is a text file
        print "<p>%s</p>" % (ADC)
        file_in.close()
except:
        print "<p>Can't open ADC data!</p>"
```

http_ADC.py

- We can create the script by WinSCP text editor and write the program below.

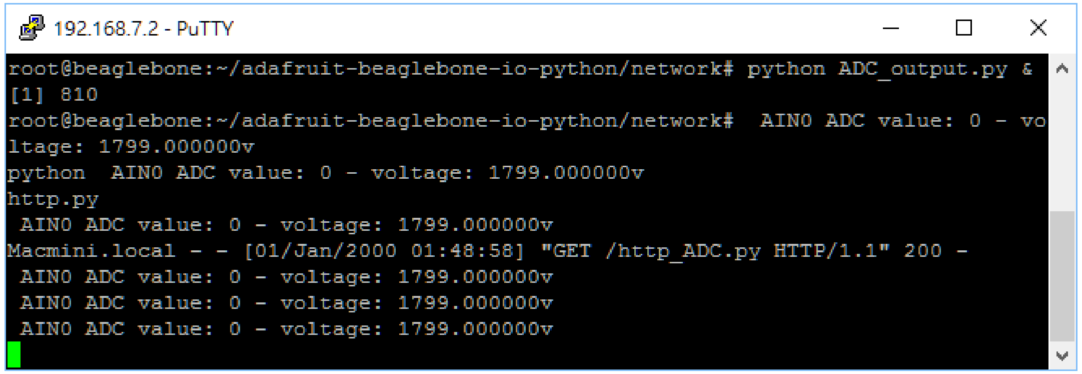
```
#!/usr/bin/python
import Adafruit_BBIO.ADC as ADC
print "Content-type: text/html"
print
print "<title>Test HTTP ADC</title>"
try:
      file_in = open("ADC.txt", "r")
      ADC = file_in.readline()     # it is a text file
      print "<p>%s</p>" % (ADC)
      file_in.close()
except:
      print "<p>Can't open ADC data!</p>“
```

HTTP ADC Services

- Now, your HTTP ADC server is operational. To see your first page fire in the PC browser and type the following into the location bar.
  - http://192.168.7.2:88888/http_ADC.py

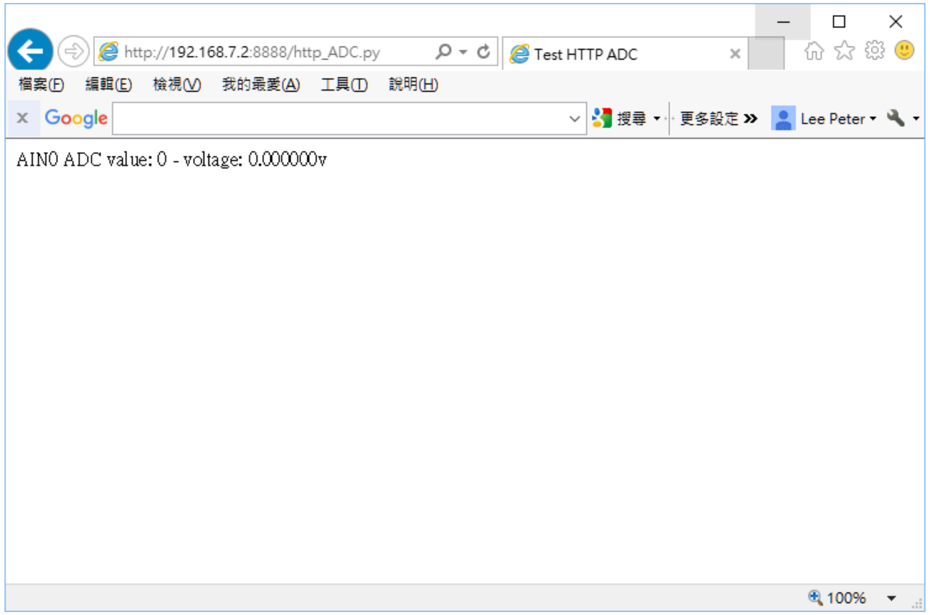
HTTP ADC Services

- HTTP Server side:



```
root@beaglebone:~/adafruit-beaglebone-io-python/network# python ADC_output.py &
[1] 810
root@beaglebone:~/adafruit-beaglebone-io-python/network#  AIN0 ADC value: 0 - vo
ltage: 1799.000000v
python  AIN0 ADC value: 0 - voltage: 1799.000000v
http.py
 AIN0 ADC value: 0 - voltage: 1799.000000v
Macmini.local - - [01/Jan/2000 01:48:58] "GET /http_ADC.py HTTP/1.1" 200 -
 AIN0 ADC value: 0 - voltage: 1799.000000v
 AIN0 ADC value: 0 - voltage: 1799.000000v
 AIN0 ADC value: 0 - voltage: 1799.000000v
```

- PC Client Browser:



AIN0 ADC value: 0 - voltage: 0.000000v

Python Email Service

- Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.
- Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.
- Here is the detail of the parameters:
  - **host:** This is the host running your SMTP server. You can specifiy IP address of the host or a domain name like tutorialspoint.com. This is optional argument.
  - **port:** If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
  - **local_hostname**: If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

- An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters:
  - The *sender* - A string with the address of the sender.
  - The *receivers* - A list of strings, one for each recipient.
  - The *message* - A message as a string formatted as specified in the various RFCs.
- Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail

email.py

```
#!/usr/bin/python

import smtplib

sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']

message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
Subject: SMTP e-mail test

This is a test e-mail message.
"""


try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print "Successfully sent email"
except SMTPException:
```

ADCEmail.py

```
#!/usr/bin/python

import Adafruit_BBIO.ADC as ADC
import time
import math
import smtplib
from email.mime.text import MIMEText

sensor = "P9_39" #or AIN0

ADC.setup()

def read_temperature():
   reading = ADC.read(sensor) # values from 0 to 1
   voltage = reading * 1.8 #values from 0 to 1.8V

   # the voltage/temperature relationship is as follows:
   # Vo = 1/100 * Temperature + 0.5
   temperature_c = (voltage - 0.5) * 100
   temperature_f = (temperature_c * 9/5) + 32
   return "the temperature in Celsius is" + temperature_c
```

ADCEmail.py

```
def send_email(message)
    my_email = raw_input("Insert your e-mail ")
    my_password = raw_input("Insert your e-mail's password ")
    subject = raw_input("Insert the subject ")
    destination = raw_input("Insert the destination e-mail ")
    text = message
    msg = MIMEText(text)
    msg['Subject'] = subject
    msg['From'] = my_email
    msg['Reply-To'] = my_email
    msg['To'] = destination
    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(my_email, my_password)
    server.sendmail(my_email, destination, msg.as_string())
    server.quit()
    print("Your e-mail has been sent!")
while True:
    temperature = read_temperature();
    send_email(temperature);
```

Assignment

- Please re-write the DIY Voltmeter (電壓表).
- You can write a TCP client to remote start sensing the voltage from AIN2 by sending a “getAIN2” string request and displays the value of voltage via the HTTP Service port 10000.
- Also, you will receive an email notification when the AIN2 value higher than 1000.