FACULTY OF ENGINEERING,
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE4436A–NETWORKING: PRINCIPLES, PROTOCOLS, AND ARCHITECTURES

Assignment 1

Schedule:
TAs will be available at labs during your assigned lab hours to answer your questions. **Assignment 1 is due Thursday October 29th, 2015 at 11:55 pm.**

Information:
This assignment is to be done individually. Do not show your code to any other student and do not look at any other student's code. Do not put your code in any public domain. However, you may get all the help you need from the instructor or the TAs (Mr. Karim Hammad   khammad@uwo.ca and Mr. Fuad Shamieh fshamieh@uwo.ca). Also you are encouraged to discuss your problems (if any) with other students. This assignment has 3 sections.

Section 1: Report
When you have completed your assignment, submit the source code using OWL. Although this is a programming assignment, you must comment the code so that it can be read and understood by another programmer that is not necessarily experienced with Python. An introduction to Python is given in pages 2-11. The assignment consists of 3 parts:

> Part 1: Develop a Ping program (Socket Programming Assignment UDP) – pages 12-14
> Part 2: Develop a simple mail client (Socket Programming Assignment - SMTP)- pages 15-17
> Part 3: Develop an HTTP Web Proxy (Server Socket Programming Assignment) – pages 18-22

Section 2: Materials
See Sections 2.7 and 2.8 of the textbook for an introduction to socket programming in Python. Furthermore, you can download the source code for the ping server from OWL to test your program. Additional examples (including most of this assignment) can be found at the textbook's website http://www.awl.com/kurose-ross.

**Section 3: Getting Started with Python**

Getting Started with Python

Python is a general purpose, high level programming language that is used in a variety of application domains. The Python language has a very clear and expressive syntax as well as a large and comprehensive library. Although Python is often used as a scripting language, it can also be used in a wide range of non-scripting contexts. It's available for all major Operating Systems: Windows, Linux/Unix, OS/2, Mac, Amiga, among others. Python is free to use, even for commercial products, because of its OSI-approved open source license.

Python 2 or Python 3?

Python has two standard versions, Python 2 and Python 3. The current production versions (July 2012) are Python 2.7.3 and Python 3.2.3. *Python 2.7 is the status quo.* We recommend you use Python 2.7 for completing the assignments.

Installing Python

Python can be downloaded directly from the official website http://www.python.org/ . This is the program that is used to write all your python code. On the left side of the website there is a download section
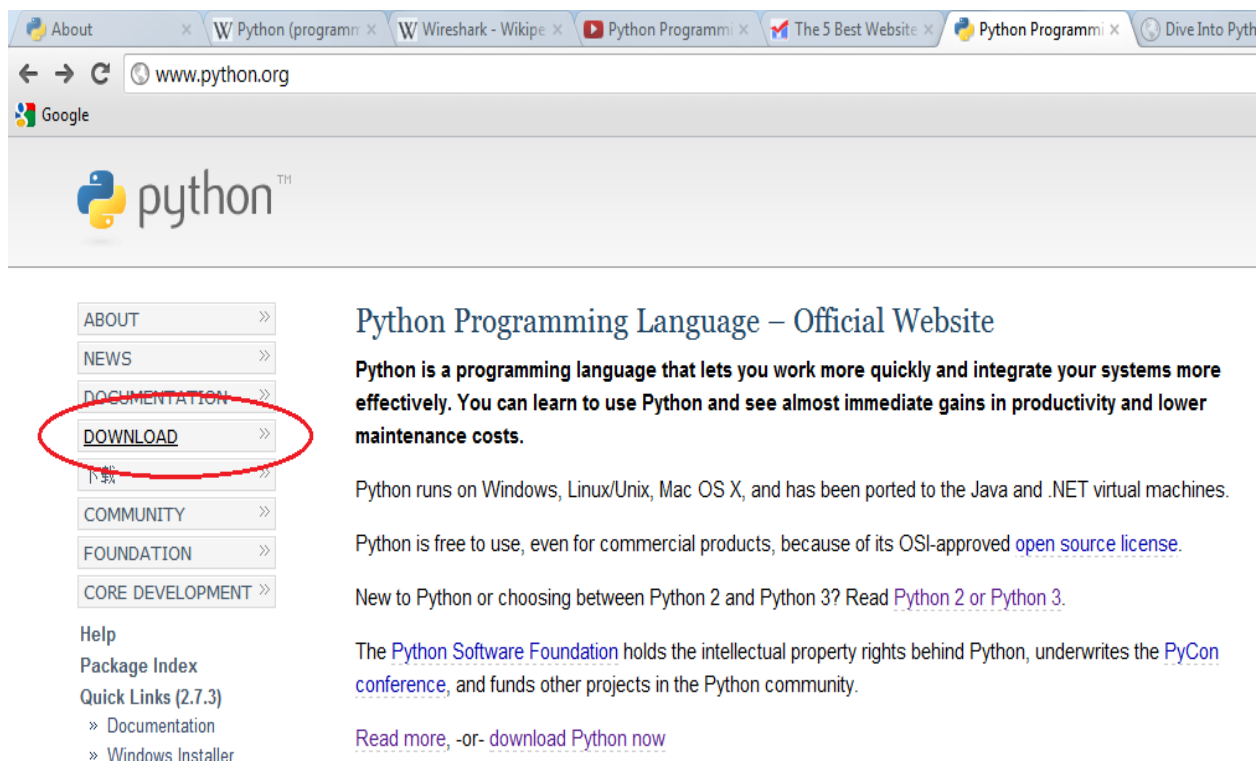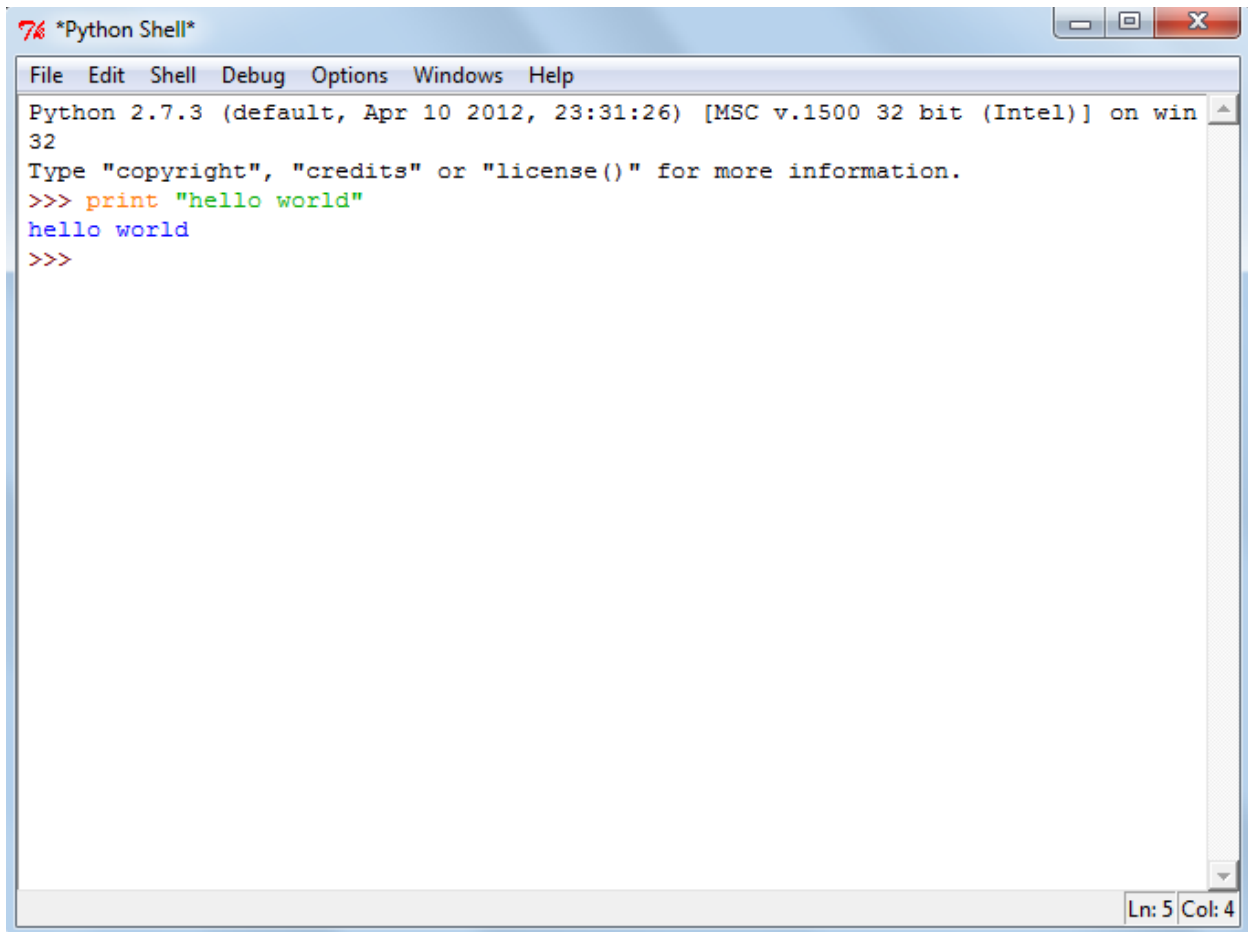
Clicking the download link will present you with two versions of Python, namely *python 2* and *python 3;* you can also chose the version specific to your operating system.   Most popular Linux distributions come with Python in the default installation. Mac OS X 10.2 and later includes a command-line version of Python, although you'll probably want to install a version that includes a more Mac-like graphical interface.

Installing Python on Windowa

1. Double-click the installer, Python-2.xxx.yyy.exe. The name will depend on the version of Python available when you read this.
2. Select run.
3. Step through the installer program.
4. If disk space is tight, you can deselect the HTMLHelp file, the utility scripts (Tools/), and/or the test suite (Lib/test/).
5. If you do not have administrative rights on your machine, you can select Advanced Options, then choose Non-Admin Install. This just affects where Registry entries and Start menu shortcuts are created.
6. If you see the following that means the installation is complete.

7. After the installation is complete, close the installer and select Start->Programs->Python 2.3->IDLE (Python GUI). You'll see something like the following:

7⅙ *Python Shell*

File   Edit   Shell   Debug   Options   Windows   Help

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> print "hello world"
hello world
>>>
```

Ln: 5 Col: 4

Other Window Installation Options

ActiveState makes a Windows installer for Python called ActivePython, which includes a complete version of Python, an IDE with a Python-aware code editor, plus some Windows extensions for Python that allow complete access to Windows-specific services, APIs, and the Windows Registry. ActivePython is freely downloadable, although it is not open source. You recommend you use this for writing more complicated programs.

Download ActivePython from http://www.activestate.com/Products/ActivePython/.If you are using Windows 95, Windows 98, or Windows ME, you will also need to download and install Windows Installer 2.0 before installing ActivePython.

Installing Python On Mac

The latest version of Mac OS X, Lion, comes with a command line version preinstalled. This version is great for learning but is not good for development. The preinstalled version may be slightly out of date, it does not come with an XML parser, also Apple has made significant changes that can cause hidden bugs.

Rather than using the preinstalled version, you'll probably want to install the latest version, which also comes with a graphical interactive shell.

Running the Preinstalled Mac Version

Follow these steps in order to use the preinstalled version.

1. Go to Finder->Applications->Utilities.
2. Double click Terminal to get a command line.
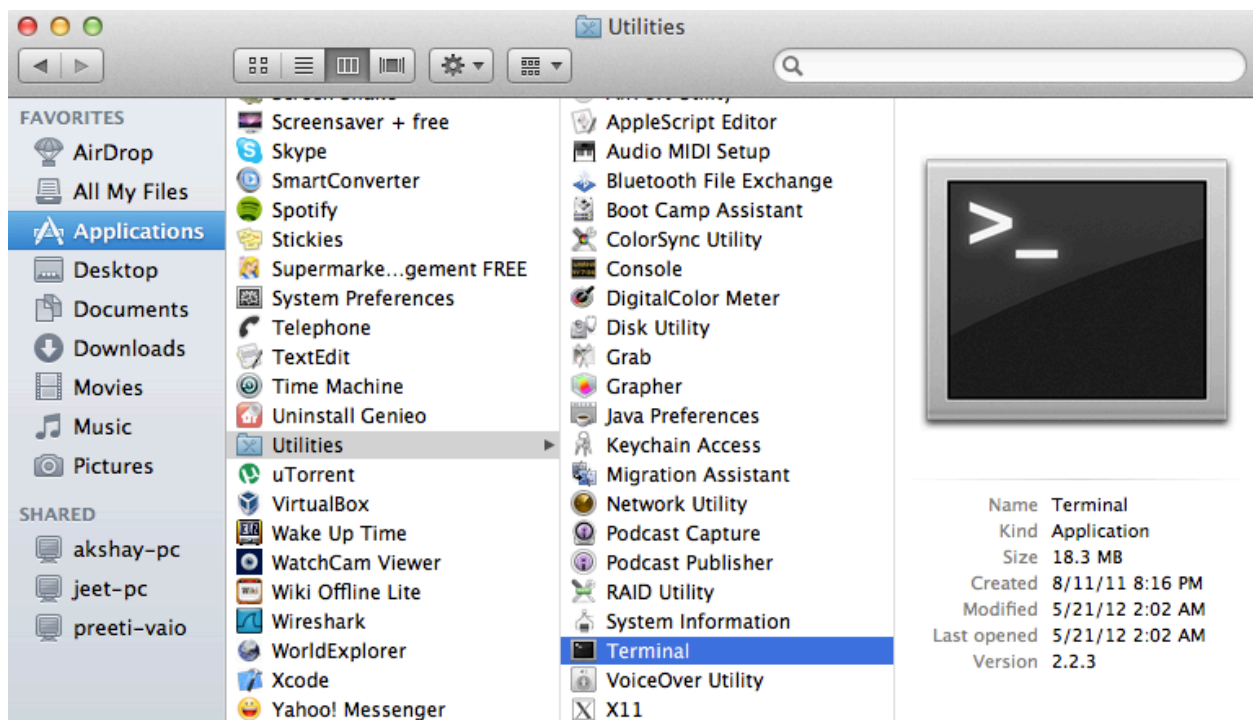
3. Type **python** at the command prompt

4. Now you can try out some basic codes here

```
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:32:06)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> print "hello world"
hello world
>>> _
```

Installing the Latest Version on the Mac

As said earlier Python comes preinstalled on Mac OS X , but due to Apple's release cycle, its often a year or two old. The "MacPython" community highly recommends you to upgrade your Python by downloading and installing a newer version.

Go to http://www.python.org/download/ and download the version suitable for your system from among a list of options.
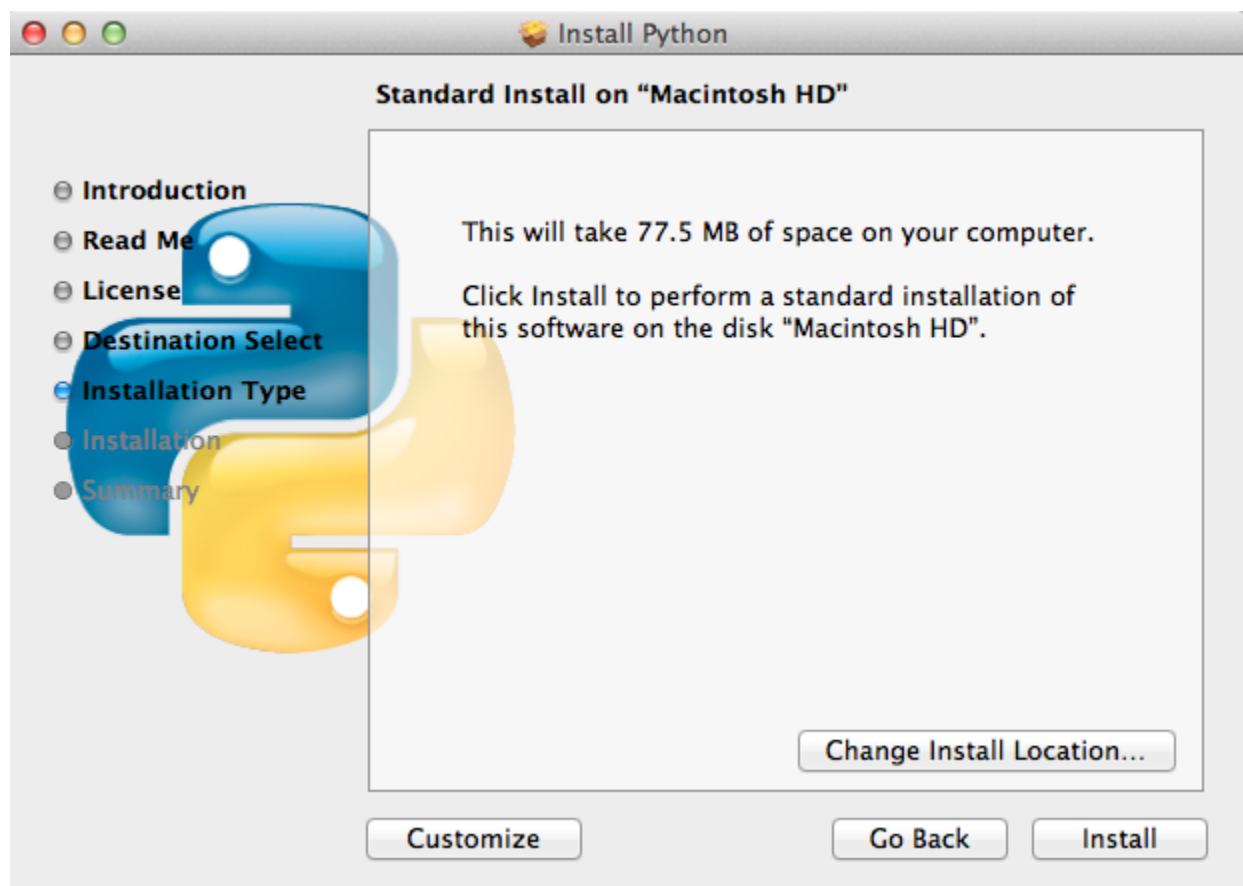
The downloaded file should look like this
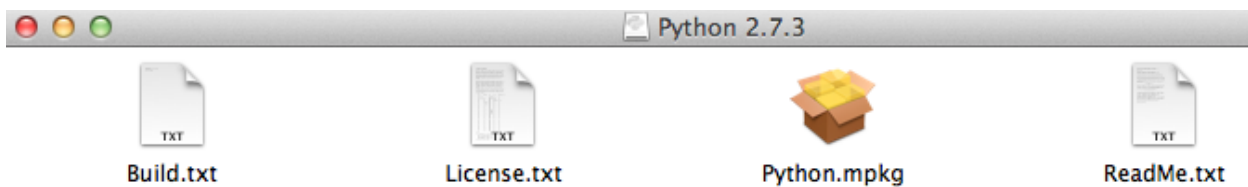
Double click the "Python.mpkg" file. The installer may prompt you for your administrative username and password.

Step through the installer program.
You can choose the location at which it is to be installed.

Install Python

Standard Install on "Macintosh HD"

- ⊖ **Introduction**
- ⊖ **Read Me**
- ⊖ **License**
- ⊖ **Destination Select**
- ⊖ **Installation Type**
- ⊙ Installation
- ⊙ Summary

This will take 77.5 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

Change Install Location...

Customize   Go Back   Install
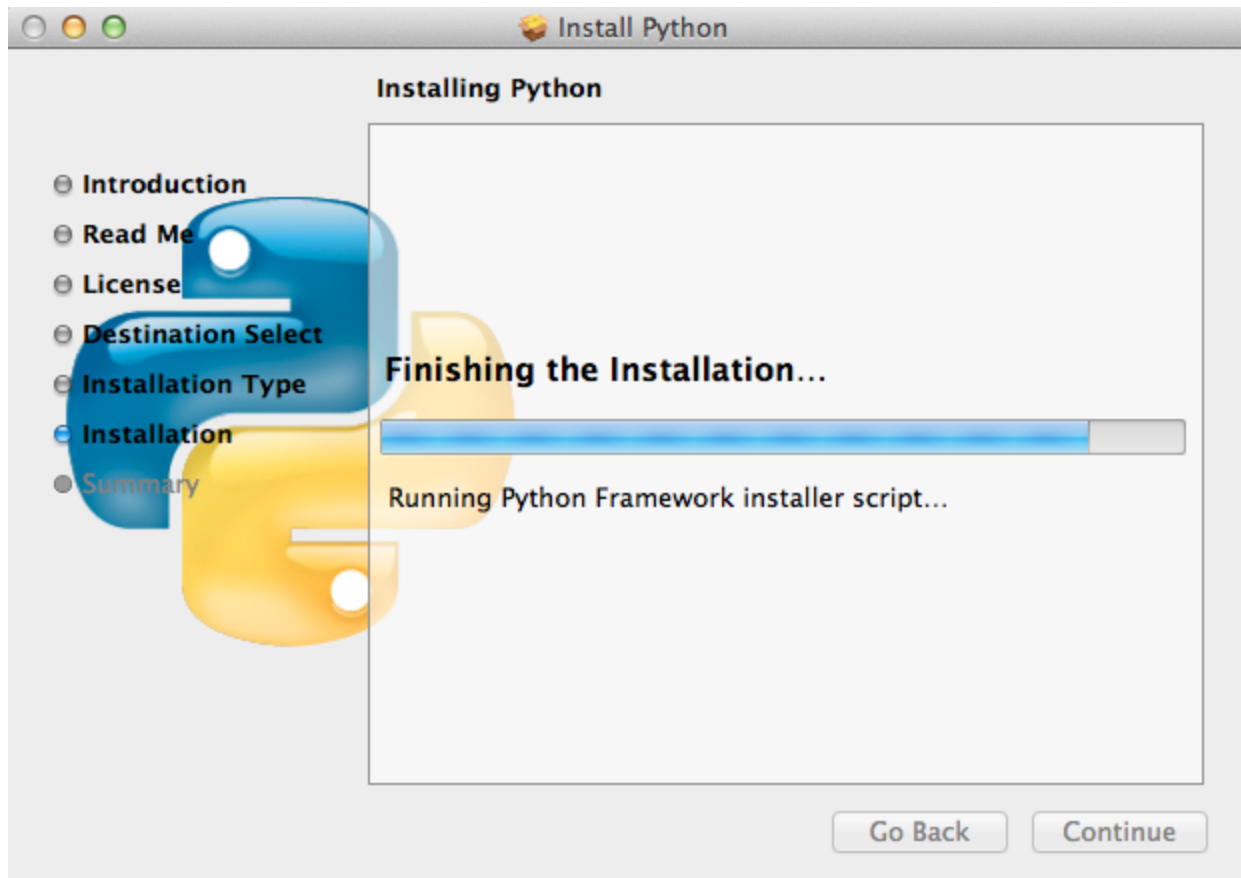
After the installation is complete, close the installer and open the Applications folder , search for Python and you'll see the Python IDLE i.e. the standard GUI that comes with the package.

Alternative Packages for Mac OS X

ActiveState ActivePython (commercial and community versions, including scientific computing modules). ActivePython also includes a variety of modules that build on the solid core.
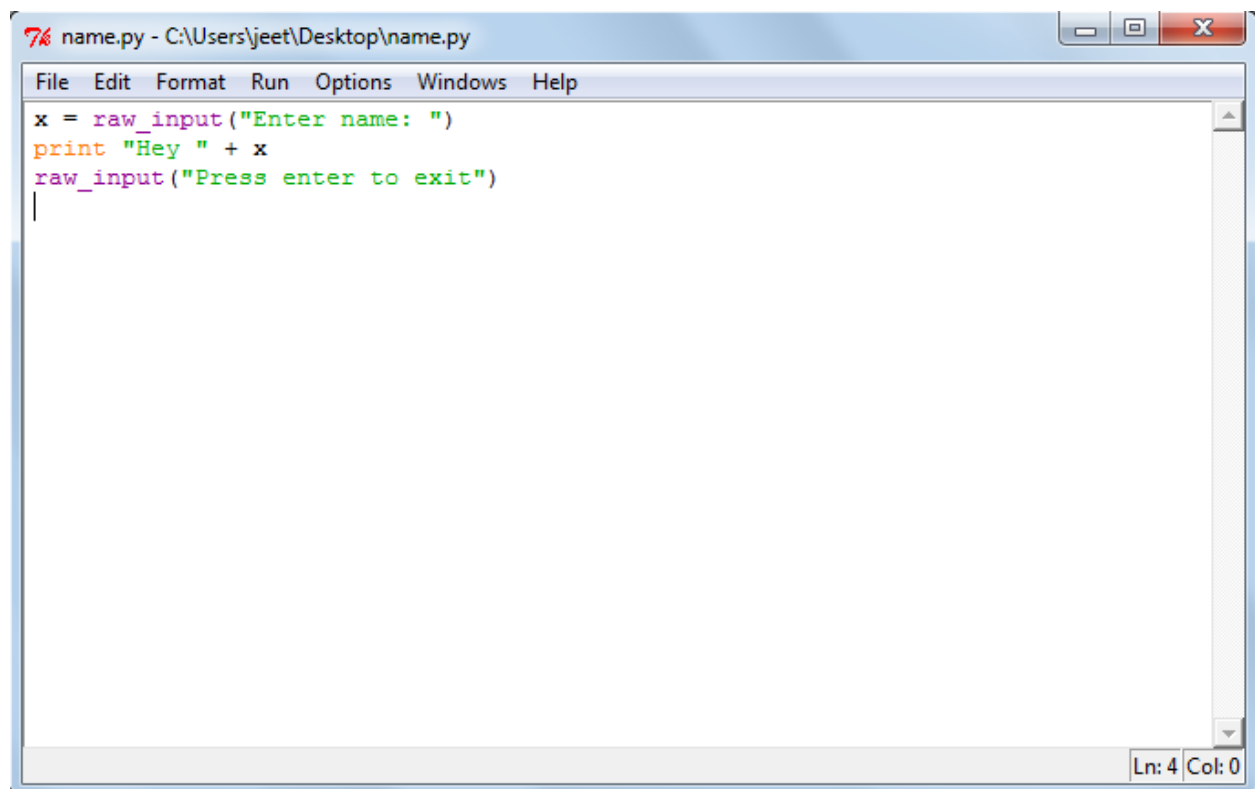
Enthought Python Distribution The Enthought Python Distribution provides scientists with a comprehensive set of tools to perform rigorous data analysis and visualization.

Example of a Basic Python Program

The interface "Idle" that we opened so far is only useful for testing out basic python commands or can otherwise be used as a calculator, it basically means the program cannot be saved this way.

To save a program and execute it we need to follow the following instruction:
On the top left corner of "Idle" select File -> New Window.

The new window that pops out will allow you to save and execute your python programs. You can write your python code in this window. Try the following:

We cannot run this program without saving it. A saved python file has an icon that looks like this

You run the program by pressing F5 or Run-> Run Module. You can also run the program by simply double clicking the file icon. When you open a saved file to run the program, you should see:
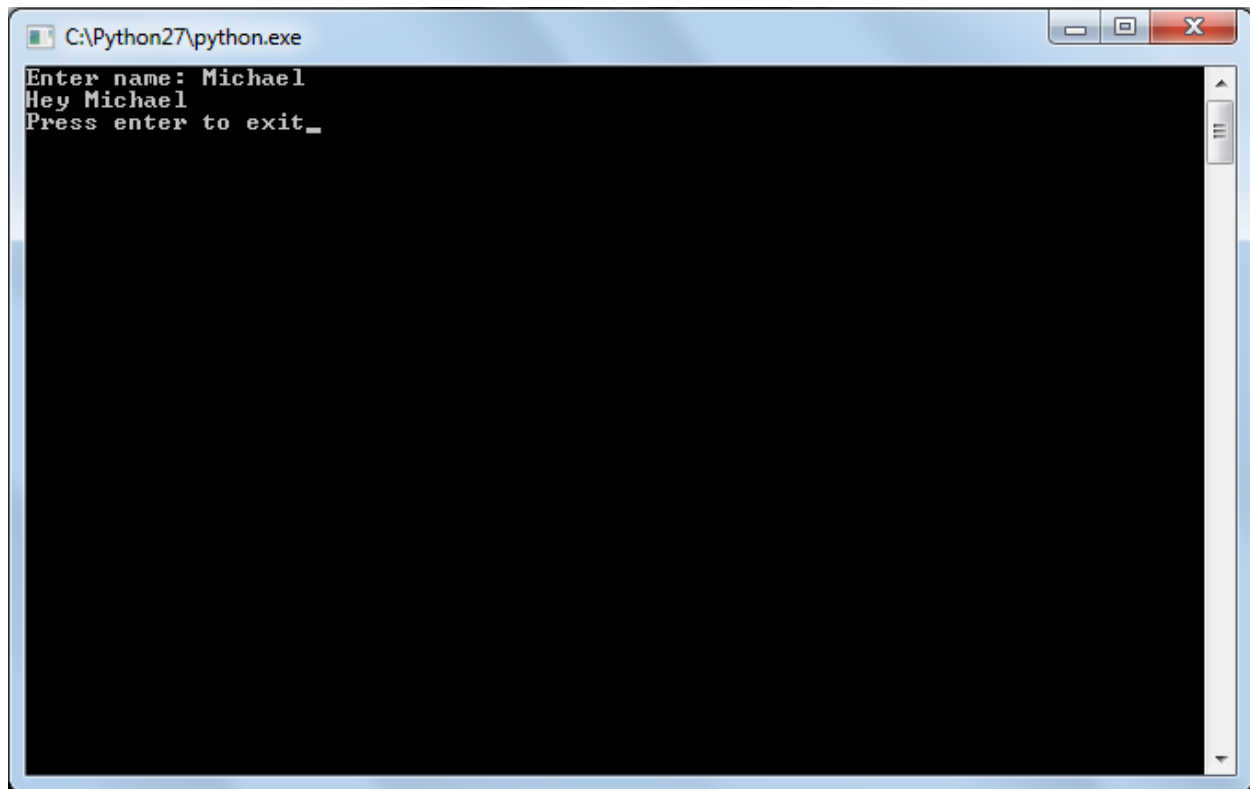
Learning Python Programming

Python is easy to learn, easy to use and very powerful. There are a lot of web resources for learning the language, most of which are entirely free. We recommend *Sthurlow.com's* A Beginner's Python Tutorial:

http://www.sthurlow.com/python/

Part.1:Socket Programming Assignment UDP

In this lab, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout. Throughout the lab, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You will first study a simple Internet ping server written in the Python, and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a simpler protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given the complete code for the Ping server below. Your task is to write the Ping client.

Server Code

The following code fully implements a ping server. You need to compile and run this code before running your client program. *You do not need to modify this code.*

In this server code, 30% of the client's packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets
import random
from socket import *

# Create a UDP socket
# Notice the use of SOCK_DGRAM for UDP packets
serverSocket = socket(AF_INET, SOCK_DGRAM)
# Assign IP address and port number to socket
serverSocket.bind(('', 12000))

while True:
    # Generate random number in the range of 0 to 10
    rand = random.randint(0, 10)
    # Receive the client packet along with the address it is coming from
    message, address = serverSocket.recvfrom(1024)
    # Capitalize the message from the client
    message = message.upper()
```

```
# If rand is less is than 4, we consider the packet lost and do not respond
if rand < 4:
    continue
# Otherwise, the server responds
serverSocket.sendto(message, address)
```

The server sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply capitalizes the encapsulated data and sends it back to the client.

Packet Loss

UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server in this lab injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

Client Code

You need to implement the following client program.
The client should send 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client wait up to one second for a reply; if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. You will need to look up the Python documentation to find out how to set the timeout value on a datagram socket.

Specifically, your client program should
(1) send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)
(2) print the response message from server, if any
(3) calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
(4) otherwise, print "Request timed out"

During development, you should run the UDPPingerServer.py on your machine, and test your client by sending packets to *localhost* (or, 127.0.0.1). After you have fully debugged your code, you should see how your application communicates across the network with the ping server and ping client running on different machines.

Message Format

The ping messages in this lab are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

Ping *sequence_number time*

where *sequence_number* starts at 1 and progresses to 10 for each successive ping message sent by the

client, and *time* is the time when the client sends the message.

What to Hand in

You will hand in the complete client code and screenshots at the client verifying that your ping program works as required.

 Along with the above specifications, your program should also do the following:

1. Currently, the program calculates the round-trip time for each packet and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage).

2. Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped. Implement the UDP Heartbeat (both client and server). You will need to modify the given `UDPPingerServer.py,` and your UDP ping client.

Part.2:Socket Progrmming Assignment - SMTP

By the end of this lab, you will have acquired a better understanding of SMTP protocol. You will also gain experience in implementing a standard protocol using Python.

Your task is to develop a simple mail client that sends email to any recipient. Your client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Python provides a module, called `smtplib`, which has built in methods to send mail using SMTP protocol. However, we will not be using this module in this lab, because it hide the details of SMTP and socket programming.

In order to limit spam, some mail servers do not accept TCP connection from arbitrary sources. For the experiment described below, you may want to try connecting both to your university mail server and to a popular Webmail server, such as a AOL mail server. You may also try making your connection both from your home and from your university campus.

Code
Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with **#Fill in start** and **#Fill in end**. Each place may require one or more lines of code.

Additional Notes
In some cases, the receiving mail server might classify your e-mail as junk. Make sure you check the junk/spam folder when you look for the e-mail sent from your client.

What to Hand in
In your submission, you are to provide the complete code for your SMTP mail client as well as a screenshot showing that you indeed receive the e-mail message.

Skeleton Python Code for the Mail Client

```python
from socket import *

msg = "\r\n I love computer networks!"

endmsg = "\r\n.\r\n"

# Choose a mail server (e.g. Google mail server) and call it mailserver

mailserver = #Fill in start    #Fill in end

# Create socket called clientSocket and establish a TCP connection with mailserver

#Fill in start


#Fill in end

recv = clientSocket.recv(1024)

print recv

if recv[:3] != '220':

    print '220 reply not received from server.'


# Send HELO command and print server response.

heloCommand = 'HELO Alice\r\n'

clientSocket.send(heloCommand)

recv1 = clientSocket.recv(1024)

print recv1

if recv1[:3] != '250':

    print '250 reply not received from server.'


# Send MAIL FROM command and print server response.

# Fill in start


# Fill in end


# Send RCPT TO command and print server response.

# Fill in start


# Fill in end


# Send DATA command and print server response.

# Fill in start


# Fill in end


# Send message data.

# Fill in start


# Fill in end
```

```
# Message ends with a single period.
# Fill in start


# Fill in end


# Send QUIT command and get server response.
# Fill in start


# Fill in end
```

Along with the above specifications, your program should also do the following:

1. Mail servers like Google mail (address: smtp.gmail.com, port: 587) requires your client to add a Transport Layer Security (TLS) or Secure Sockets Layer (SSL) for authentication and security reasons, before you send MAIL FROM command. Add TLS/SSL commands to your existing ones and implement your client using Google mail server at above address and port.

Part.3:Socket Programming Assignment - HTTP Web Proxy Server

In this lab, you will learn how web proxy servers work and one of their basic functionalities – caching.

Your task is to develop a small web proxy server which is able to cache web pages. It is a very simple proxy server which only understands simple GET-requests, but is able to handle all kinds of objects - not just HTML pages, but also images.

Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.
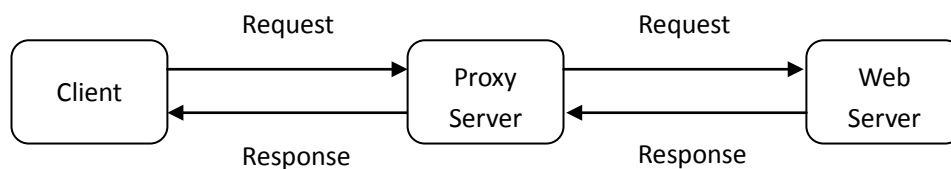
Code

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with **#Fill in start** and **#Fill in end**. Each place may require one or more lines of code.

Running the Proxy Server

Run the proxy server program using your command prompt and then request a web page from your browser. Direct the requests to the proxy server using your IP address and port number.

For e.g. http://localhost:8888/www.google.com

To use the proxy server with browser and proxy on separate computers, you will need the IP address on which your proxy server is running. In this case, while running the proxy, you will have to replace the "localhost" with the IP address of the computer where the proxy server is running. Also note the port number used. You will replace the port number used here "8888" with the port number you have used in your server code at which your proxy server is listening.

Configuring your Browser

You can also directly configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in Tools > Internet Options > Connections tab > LAN Settings. In Netscape (and derived browsers such as Mozilla), you can set the proxy in Tools > Options > Advanced tab > Network tab > Connection Settings. In both cases you need to give the address of the proxy and the port number that you gave when you ran the proxy server. You should be

able to run the proxy and the browser on the same computer without any problem. With this approach, to get a web page using the proxy server, you simply provide the URL of the page you want.

For e.g. http://www.google.com

What to Hand in

You will hand in the complete proxy server code and screenshots at the client side verifying that you indeed get the web page via the proxy server.

Skeleton Python Code for the Proxy Server

```python
from socket import *
import sys


if len(sys.argv) <= 1:
        print 'Usage : "python ProxyServer.py server_ip"\n[server_ip : It is the IP
                Address Of Proxy Server'
        sys.exit(2)


# Create a server socket, bind it to a port and start listening
tcpSerSock = socket(AF_INET, SOCK_STREAM)
# Fill in start.
# Fill in end.
while 1:
        # Strat receiving data from the client
        print 'Ready to serve...'
        tcpCliSock, addr = tcpSerSock.accept()
        print 'Received a connection from:', addr
        message = # Fill in start.        # Fill in end.
        print message
        # Extract the filename from the given message
        print message.split()[1]
        filename = message.split()[1].partition("/")[2]
        print filename
        fileExist = "false"
        filetouse = "/" + filename
        print filetouse
        try:
                # Check wether the file exist in the cache
                f = open(filetouse[1:], "r")
                outputdata = f.readlines()
                fileExist = "true"
                # ProxyServer finds a cache hit and generates a response message
                tcpCliSock.send("HTTP/1.0 200 OK\r\n")
                tcpCliSock.send("Content-Type:text/html\r\n")
                # Fill in start.
                # Fill in end.
                        print 'Read from cache'
        # Error handling for file not found in cache
        except IOError:
                if fileExist == "false":
                        # Create a socket on the proxyserver
                        c = # Fill in start.              # Fill in end.
                        hostn = filename.replace("www.","",1)
```

```
                print hostn
                try:
                        # Connect to the socket to port 80
                        # Fill in start.
                        # Fill in end.
                        # Create a temporary file on this socket and ask port 80
                        for the file requested by the client
                        fileobj = c.makefile('r', 0)
                        fileobj.write("GET "+"http://" + filename + "
                                HTTP/1.0\n\n")
                        # Read the response into buffer
                        # Fill in start.
                        # Fill in end.
                        # Create a new file in the cache for the requested file.
                        # Also send the response in the buffer to client socket
                        and the corresponding file in the cache
                        tmpFile = open("./" + filename,"wb")
                        # Fill in start.
                        # Fill in end.
                except:
                        print "Illegal request"
        else:
                # HTTP response message for file not found
                # Fill in start.
                # Fill in end.
    # Close the client and the server sockets
    tcpCliSock.close()
# Fill in start.
# Fill in end.
```

Optional Exercises

1. Currently the proxy server does no error handling. This can be a problem especially when the client requests an object which is not available, since the "404 Not found" response usually has no response body and the proxy assumes there is a body and tries to read it.

2. The simple proxy server supports only HTTP GET method. Add support for POST, by including the request body sent in the POST-request.

3. *Caching*: A typical proxy server will cache the web pages each time the client makes a particular request for the first time. The basic functionality of caching works as follows. When the proxy gets a request, it checks if the requested object is cached, and if yes, it returns the object from the cache, without contacting the server. If the object is not cached, the proxy retrieves the object from the server, returns it to the client and caches a copy for future requests. In practice, the proxy server must verify that the cached responses are still valid and that they are the correct responses to the client's requests. You can read more about caching and how it is handled in HTTP in RFC 2068. Add the simple caching functionality described above. You do not need to implement any replacement or validation policies. Your implementation, however, will need to be able to write

responses to the disk (i.e., the cache) and fetch them from the disk when you get a cache hit. For this you need to implement some internal data structure in the proxy to keep track of which objects are cached and where they are on the disk. You can keep this data structure in main memory; there is no need to make it persist across shutdowns.