Lecture 11

Network Programming
JavaScript

Prof. Steven Ludtke
N410.07, sludtke@bcm.edu

The Internet

- When I enter www.google.com into my web browser, what happens?

The Internet

- My machine opens a connection to the server at Google and requests the main HTML page
- Google server sends the page
- Browser displays the page

The Internet

- My machine opens a connection to the server at Google and requests the main HTML page
- Google server asks who I am (cookie)
- My browser sends my credentials
- Google server sends the (personalized) page
- Browser displays the page

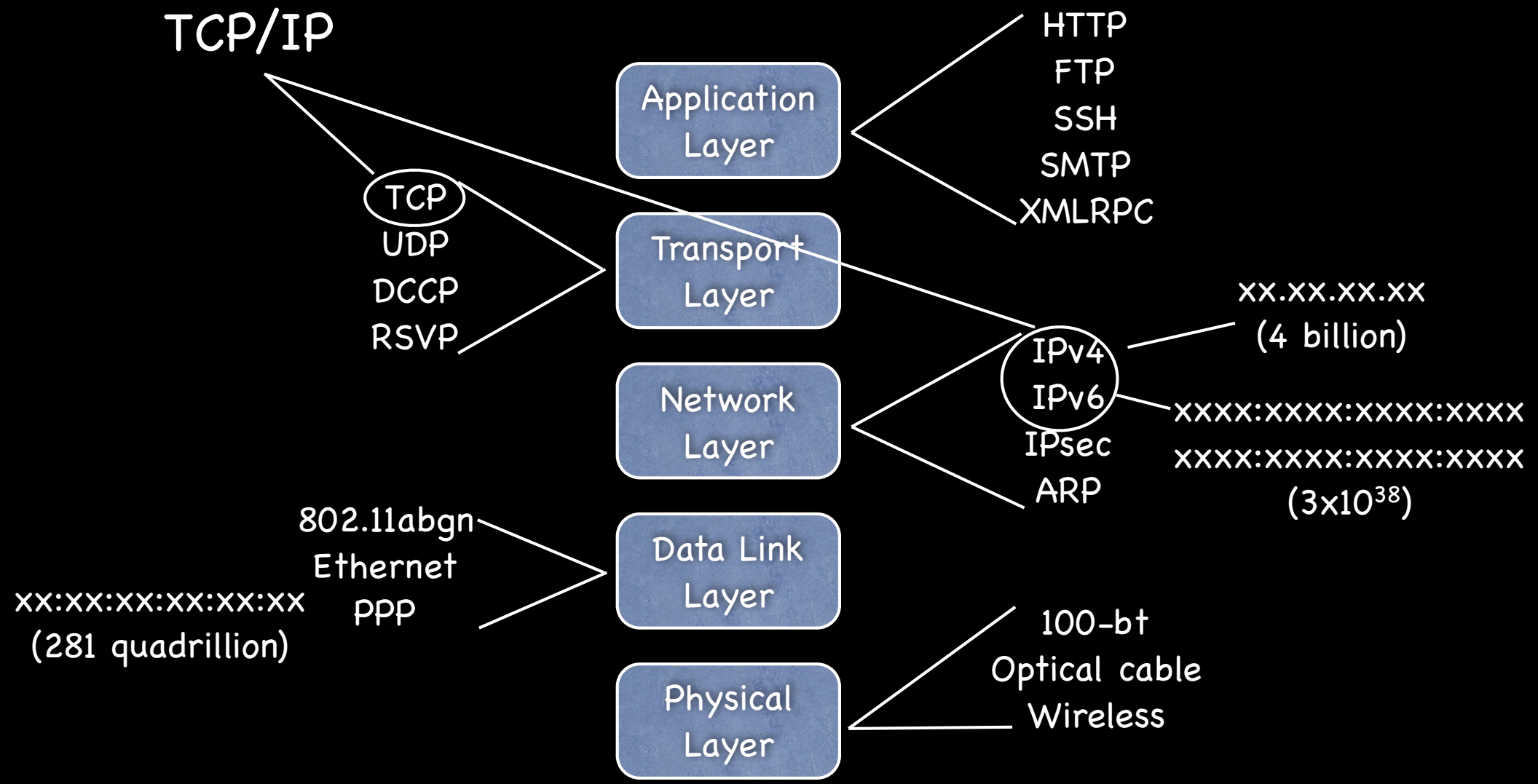
The Internet

- My machine opens a connection to the server at Google and requests the main HTML page
  - How does my machine connect to www.google.com?
  - Where is this machine?
  - How does it connect?
  - What does "connect" mean?

The Internet

- My machine asks my local DNS server for the IP address of the name "www.google.com"
- DNS server either has the information or gets it by asking other DNS servers. Sends my machine the address.
- My machine opens a TCP connection to this address on port 80, and sends a HTTP request for "/" the root page.
- Google looks at the information in my request, including the browser I claim to be using, sends back its request to my browser for a "cookie" it stored in my browser the last time I visited the site.
- My browser returns the "cookie" if it has one, or replies that it doesn't have one.
- Google assembles a customized HTML page based on everything it knows and sends it back to me.
- My browser renders the HTML page, and starts running any embedded JavaScript programs.

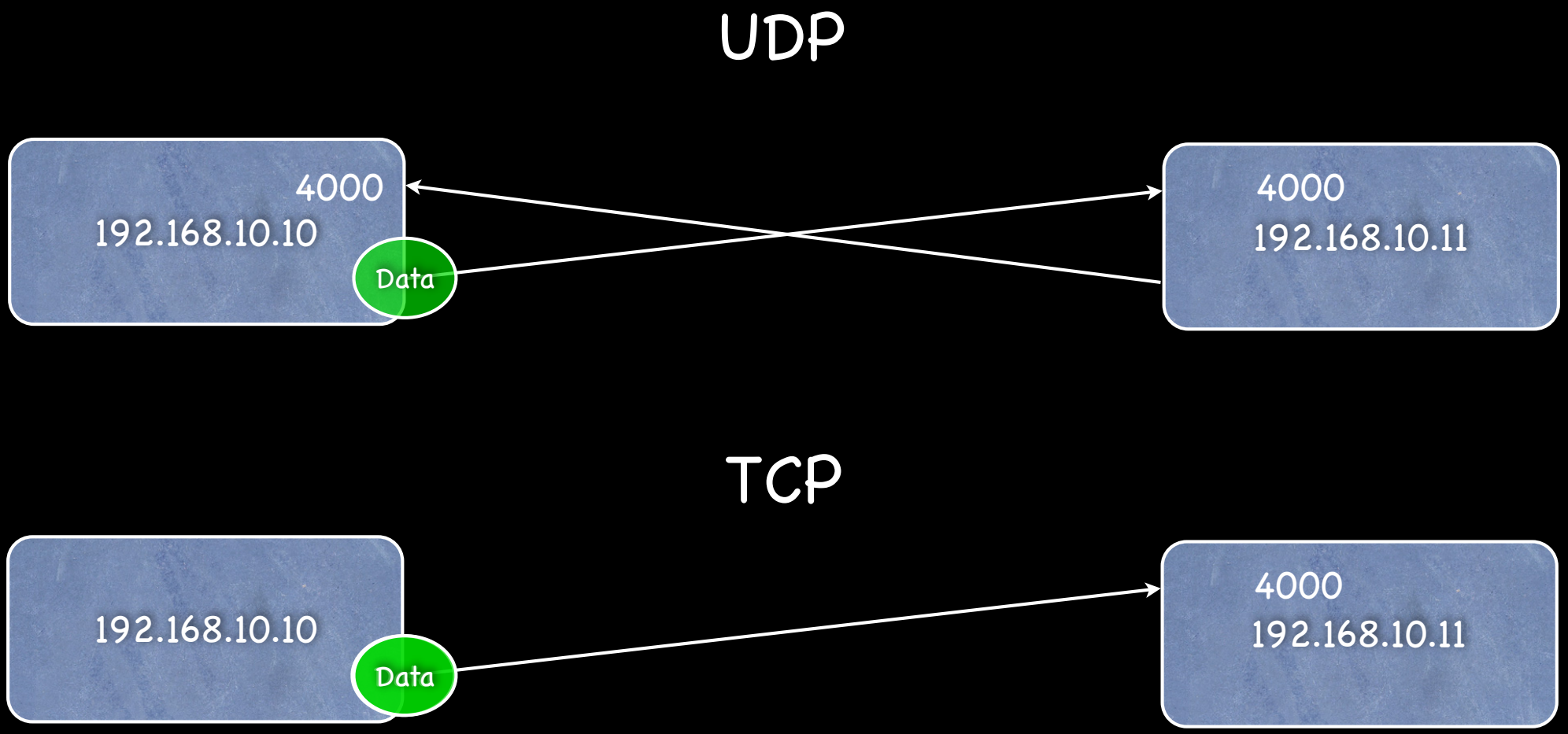
Networking


TCP/IP
Application Layer
HTTP
FTP
SSH
SMTP
XMLRPC
TCP
UDP
DCCP
RSVP
Transport Layer
xx.xx.xx.xx
(4 billion)
IPv4
IPv6
xxxx:xxxx:xxxx:xxxx
xxxx:xxxx:xxxx:xxxx
(3x10^38)
Network Layer
IPsec
ARP
802.11abgn
Ethernet
PPP
xx:xx:xx:xx:xx:xx
(281 quadrillion)
Data Link Layer
100-bt
Optical cable
Wireless
Physical Layer

Common Services

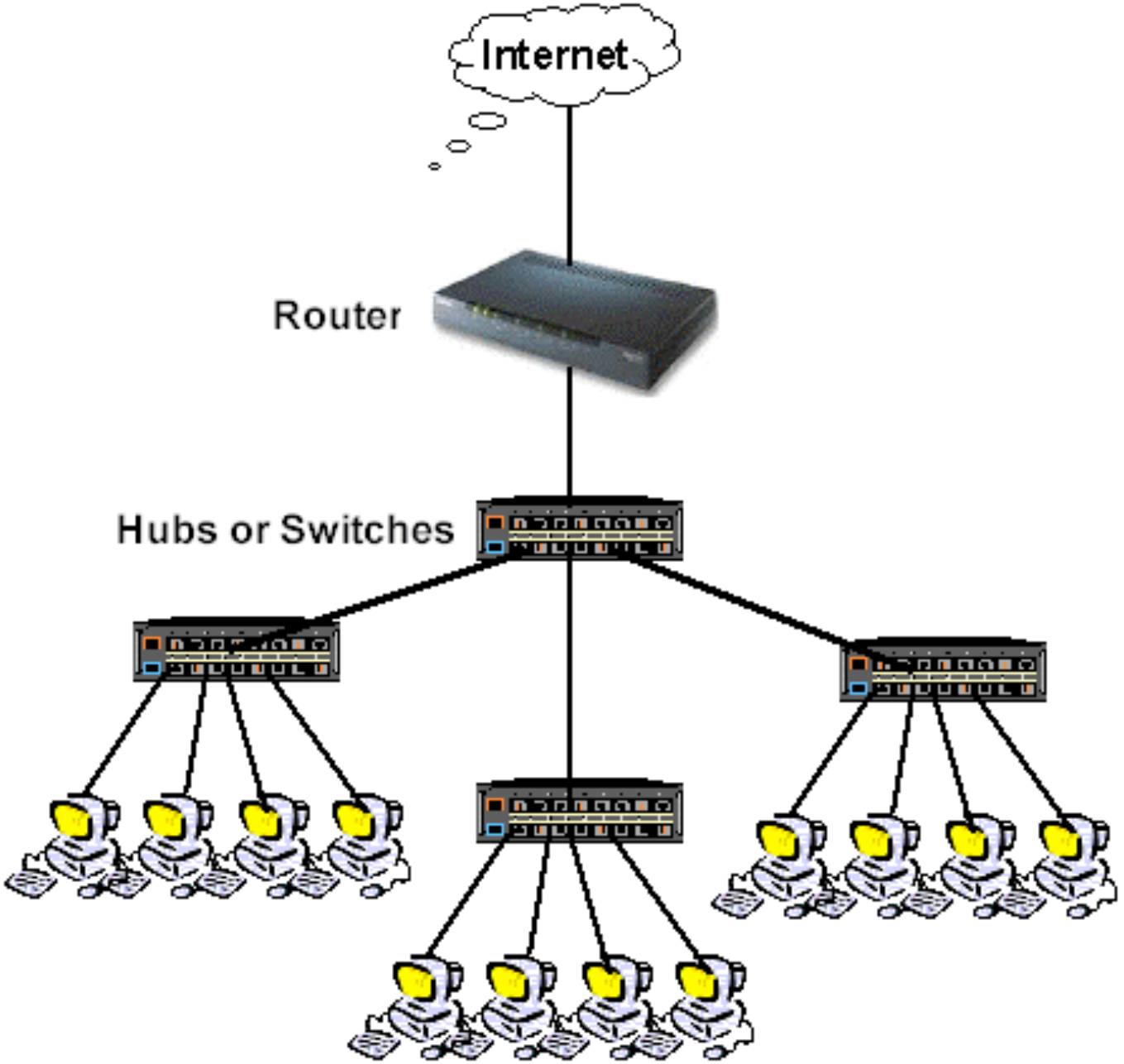
port	service
21	ftp
22	ssh
23	telnet
25	smtp (mail)
79	finger
80	http (web)
110	pop3 (email retrieval)
123	ntp (time)
137-139	Windows file sharing
143	imap (email retrieval)
443	https (secure http)

Sockets (TCP/UDP)



IPv4 Network Parameters

- IP Address - Computer's unique* address (x.x.x.x)
- Netmask - defines local 'subnet', machines the computer can speak to 'directly'
- Router - Address used to contact machines outside subnet
- DNS Server - Address where names can be mapped to addresses
- Port - For a specific connection 0-65535, 0-1023 reserved for system services



http://www.practicallynetworked.com/networking/port_expand.htm

On your laptop

- blake.bcm.edu/IP16
- download udp_chat.py
- Launch Spyder
- connect your laptop to network called "IP"
- note: you will not have other internet access after doing this.

Socket Module

- gethostname() - name of the current machine
- gethostbyname() - given a name, returns an IP address
- gethostbyaddr() - given an address, returns names and other info
- socket() - create a new socket
  - listen(n) - waits for incoming connections on a socket n>1
  - accept() - accepts an incoming connection, returns tuple with socket
  - connect((host,port)) - connects to a remote socket
  - send(), recv() - send and receive data
  - sendall() - send until success or error
  - makefile() - make a file-like object for the socket
- bind() - bind a socket to an address
- select() - from select module, lets you wait for activity on set of sockets

Making Connections

```
# Receiver/server
import socket
sock=socket.socket()  # default is to make a normal internet socket
sock.bind(("",40000)) # Nothing magic about 40000
sock.listen(1)        # Wait for 'connect' requests
sock2=sock.accept()   # accept the connection (new socket)
print sock2[0].recv(256)  # receive 256 bytes of data

# Sender
import socket
sock=socket.socket()          # default socket
sock.connect((target,40000))  # connect to someone listen()ing
sock.send("Hello there")      # send a string
```

UDP

```
#receiver
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind(("",40000))
print s.recv(1000) # up to 1000 bytes

#sender
s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
s.bind((“”,40000))
s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
s.sendto(“Hello there”,(“<broadcast>”,40000))
```

Socket vs File Obj.

- Socket:
  - send - will transmit data immediately
  - recv - specifies maximum amount to receive. May not get all sent data in one call. Call multiple times until you have what you expected
- File:
  - write - buffers. Need to call flush() for immediate transmission.
  - read/readline - Reads the full amount expected. Multiple calls not required.

Chat Program

Write a multiuser chat program, where anything typed by one user is displayed on all other users's displays

client <-> server

Chat Program

- Server
  - Make a socket, bind to port
  - Loop forever, listening for connections
    - Add the new socket from the connection to a list
  - Loop forever to all of the sockets in the list
    - When text comes in on a socket, send it back out to others
- Client
  - Make a socket, connect to server
  - Loop forever
    - Read input from user
    - Send to server on socket
    - Quit if correct string is typed
  - Loop forever
    - Read from server socket
    - print received string
    - Quit if correct string is received

Threads

How to do more than one thing at a time:

```
from threading import Thread
import time

def func():                    # This is the thread function
  for i in range(10):
    time.sleep(1.2)
    print("\tThread 2: ",i)

thread=Thread(target=func) # create a new thread
thread.start()          # start the thread running, returns immediately

for i in range(8):             # do something in the main program
  time.sleep(1.7)
  print("Thread 1: ",i)

thread.join()                  # wait for the second thread to finish
```

```
#!/usr/bin/env python
import socket
from threading import Thread
import time

global doexit,sock
doexit=False
sock=None

def receive():
    """This will listen for messages until the global 'doexit' is set"""
    global doexit,sock

    sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)   # create socket
    sock.bind(("",40000))                  # attach to port 40000
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    while True:
        if doexit : break
        msg=sock.recvfrom(1000)                # receive up to 1000 characters
        print(msg[1][0],": ",str(msg[0],'utf-8'))

# This starts a separate thread for listening for messages
thread=Thread(target=receive)
thread.start()

# sender

while True:
    txt=input()
    sock.sendto(bytes(txt,'utf-8'),("<broadcast>",40000))      # broadcast on port 40000
    if txt=="quit" or txt=="exit" : break      # if the user wants to quit

doexit=True
time.sleep(1)
```

The Internet

- What happens when I go to http://maps.google.com ?

Simple Python Webserver

```
# This will serve files from the current directory
# we use port 8080 because port 80 is restricted

from http.server import *

httpd = HTTPServer(("",8080),SimpleHTTPRequestHandler)
httpd.serve_forever()
```

Scripting, Server vs. Client

- Serverside scripting depends on the webserver you use
  - Many choices
  - May put load on server
- Clientside
  - Java - often available, but many issues
  - Flash - Almost ubiquitous, but rapidly fading
  - HTML5 - provides many dynamic capabilities
  - Javascript built in to most browsers
    - AJAX - Asynchronous Javascript and XML
    - AJAJ - Asynchronous Javascript and JSON

Javascript - Button

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Here is a title</h3>
And some text
<p>
<input type="button" value="Push Me" onclick="alert('You pushed me too far')">
</p>
</body>
```

Javascript - mouseover

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Here is a title</h3>
And some text
<p>
<a href="index3.html" onmouseover="window.document.bgColor='red'">Red</a>
<a href="index3.html" onmouseover="window.document.bgColor='green'">Green</
a>
<a href="index3.html" onmouseover="window.document.bgColor='blue'">Blue</a>
<a href="index3.html" onmouseover="window.document.bgColor='white'">White</
a>
</p>
</body>
```

Javascript Calculator

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Calculator</h3>
<input type=text name='data' onkeypress='compute(event)' />
<br><input type=text name='result' readonly=true />
<script>
function compute(event) {
if (event.keyCode!=13) { return; }
data=document.getElementsByName('data')[0];
result=document.getElementsByName('result')[0];
result.value=eval(data.value);
}
</script>
</body>
```

Javascript - Calculator #2

```
<HTML><HEAD><TITLE>Hi there</TITLE></HEAD>
<BODY>
<h3>Calculator</h3>
<form name=calc onsubmit=compute()>
<input type=text name=data value="0"></input>
<table><tr>
<td><input type="button" value="7" onclick="num('7')"></td>
<td><input type="button" value="8" onclick="num('8')"></td>
<td><input type="button" value="9" onclick="num('9')"></td>
<td><input type="button" value="X" onclick="fn('*')"></td></tr><tr>
<td><input type="button" value="4" onclick="num('4')"></td>
<td><input type="button" value="5" onclick="num('5')"></td>
<td><input type="button" value="6" onclick="num('6')"></td>
<td><input type="button" value="-" onclick="fn('-')"></td></tr><tr>
<td><input type="button" value="1" onclick="num('1')"></td>
<td><input type="button" value="2" onclick="num('2')"></td>
<td><input type="button" value="3" onclick="num('3')"></td>
<td><input type="button" value="+" onclick="fn('+')"></td></tr><tr>
<td colspan=3><input type="button" value="0" onclick="num('0')"></td>
<td><input type="button" value="=" onclick="eql()"></td>
</tr> </table> </form>
```

Javascript - Calculator #2

```
<script>
xpr=""
rst=1
function num(val) {
	xpr+=val
	if (rst) {
		rst=0
		document.calc.data.value=""
	}
	document.calc.data.value+=val
}

function fn(val) {
	xpr+=val
	rst=1
}

function eql() {
	document.calc.data.value=eval(xpr)
	xpr=""
	rst=1
}
</script>
</body>
```

Javascript - Statements

```
var name[=value],name[=value]
function f(x,y) statement
if (expression) statement; else statement;
do statement while (expression)
while (expression) statement
for ( var in array ) statement
for (init; update; test) statement
switch (expr) {
case const:
        statements
        break
default:
        statements
}
```

Javascript - Events

- onclick
- onfocus, onblur
- onmousedown, up, move, over,out
- onkeydown, up, press
- onreset
- onsubmit
- onload, unload

References

- http://www.w3.org/TR/html4/
- http://www.w3.org/TR/html4/index/elements.html
- http://htmlhelp.com/reference/html40/olist.html

- http://www.javascriptkit.com/jsref
- http://www.w3schools.com/jsref/default.asp

CLASS PROJECT PRESENTATIONS

- Monday, Feb 29
- 9 AM (usual class time & location)
- We have the room until 11:30, but shouldn't need it
- You will have 10 minutes total:
  - Set up your presentation (1 minute) - TEST LAPTOP BEFORE FEB 29!!!
  - Give your talk (7 minutes)
    - What does your software do, and why did you write it
    - Inputs and outputs
    - Demonstration
  - Questions (2 minutes)
- 1/3 of your grade will be for the presentation, and 2/3 for the program itself. Combined this is 1/2 of your final grade in the class.
- The program MUST WORK to get a good grade. Better to turn in something that doesn't do everything you wanted, but works, than something broken

CLASS PROJECTS

- Must do something useful in some specific context
- Not be trivial
- If you have past programming experience I will expect more

- **Please follow these instructions exactly:**
- Your class project MUST be submitted by 11:59 PM on Sat, Feb 27. No revisions will be accepted after this time. You can use Sunday to prepare your oral presentation
- Your submission should consist of:
- one or more .py files (should have sufficient comments to figure out how they work)
- any necessary additional files to demonstrate that the program works
- A PDF file with a brief description of your program, what inputs the program takes, what outputs the program produces, and what it is supposed to do.
- The final item in the PDF should be a command-line to use in running the program, and any necessary instructions to demonstrate that it works.
- Combine all files into a .zip file named: Familyname_Givenname_project_2016.zip
- Email sludtke@bcm.edu with the subject "Class project submission", and attach the .zip file