Hello, Class in Python!

>> Print("Hello, Class")

Interpreted language, no compilation!
Basics of Python Programming

Hands on in the class!

Online resources:
• https://www.programiz.com/python-programming/examples
• https://www.geeksforgeeks.org/python-programming-examples/
Networking Stack

- OS implements the networking stack (software) for the communication.
- The networking stack is exposed using the networking APIs for packet transmission and reception.
- Packet headers are essential as they provide information required for successful communication.
The TCP/IP Layering Model

- The Socket Interface provided by the Berkeley distribution of Unix is a standard now.
- Protocol provides a set of services.
- The API provides a syntax to invoke services in the OS.

Socket API

Application
Host-to-host
Internetwork
Subnetwork
The Networking Protocols

- IP is the narrow waist of the protocol stack
- It is a connectionless protocol which provides best-effort delivery
- The addressing used is logical

<table>
<thead>
<tr>
<th>Telnet</th>
<th>SSH</th>
<th>SMTP</th>
<th>FTP</th>
<th>NFS</th>
<th>DNS</th>
<th>SNMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet, Token Ring, RS232, IEEE 802.3, HDLC, Frame Relay, Satellite, Wireless Links, Wet String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Application
Host-to-host
Internetwork
Subnetwork
Connectionless vs. Connected-oriented

- UDP provides connectionless transport
  - Unreliable
  - Applications handle errors

- TCP provides connection-oriented transport
  - Reliable
  - Retry till the message delivery is successful
Socket

• Socket is a point where local application attaches to the network (application creates a socket)
• It is an interface between an application and the network
• Sockets are:
  • created, opened/closed, attached to the network, and used to send/receive data from the network
• Socket family: AF_INET denotes the Internet family
• Socket type:
  • SOCK_DGRAM (message oriented/connectionless)
  • SOCK_STREAM (streaming/connection-oriented)
Server-Client Basics

- **Server**
  - Opens a specific port for a service
  - Passively waits for the connection requests

- **Client**
  - Unique port > 1024
  - Actively initiates the connections

See: the file `/etc/services`
Connectionless Service using Python

Server

```
from socket import socket, AF_INET, SOCK_DGRAM
s = socket(AF_INET, SOCK_DGRAM)
s.bind(('127.0.0.1', 8888))
while True:
    data, addr = s.recvfrom(1024)
    print("Connection from", addr)
    s.sendto(data.upper(), addr)
```

Client

```
from socket import socket, AF_INET, SOCK_DGRAM
s = socket(AF_INET, SOCK_DGRAM)
s.bind(('127.0.0.1', 0))
server = ('127.0.0.1', 8888)
s.sendto("comnetsii", server)
data, addr = s.recvfrom(1024)
print("received", data, "from", addr)
s.close()
```
Connection Oriented Services

When interaction is over, server loops to accept a new connection
Asynchronous Communication

- Achieved using non-blocking sockets
- `socket.send()` and `socket.receive()` are blocking calls
  - They don’t return until a task completes
- Does not work well with the databases
- There are `select()` and `poll()` methods available
  - They watch for the socket activities (send/receive/error)
Utility Functions

Data Formatting:
• `htonl(i), htons(i)`  
  • 32-bit or 16-bit integer to network format
• `ntohl(i), ntohs(i)`  
  • 32-bit or 16-bit integer to host format
• `inet_aton(ipstr), inet_ntoa(packed)`  
  • Convert addresses between regular strings and 4-byte packed strings

Timeouts:
• Can set a default for all sockets  
  • `socket.setdefaulttimeout(seconds)`  
  • Argument is float # of seconds  
  • Or `None` (indicates no timeout)
• Can set a timeout on an existing socket `s`  
  • `s.settimeout(seconds)`

Note: some behavior may be platform dependent, since calls are made to the operating system socket APIs.
Assignment 2 Objectives

• Explore Python to learn network programming concepts
• Be able to write simple socket programs
Submission Info

• Due:
  • Assignment2: Feb. 7th (before class)

• Submission instructions:
  • Prepare a Word (or Latex) file with the answers. You can write the answers and/or take
    screenshots where appropriate
  • Submit your work to sumitm@winlab.rutgers.edu with the subject “ECE423/544 Assignment2”
  • Submit ONLY a single archive with all your files.
  • The archive/file name should be “your_name_assignment_number”.
  • If you have written additional test configuration file(s), feel free to send
  • Make sure your code is readable – use commenting!
  • Write a README file if there is anything you think we should know
  • Do not submit the old and unmodified files
  • For the coming exercises, explore Networking using Python.
General Info

- Course website: [http://www.winlab.rutgers.edu/comnet2/](http://www.winlab.rutgers.edu/comnet2/)
  - Check the project tab for the project resources and assignments
    [http://www.winlab.rutgers.edu/comnet2/Projects/project_index.html](http://www.winlab.rutgers.edu/comnet2/Projects/project_index.html)

- Again, get comfortable with the VM as soon as possible
  - You do not want to get stuck at the last minute without even having the chance to do the assignments

- Technical questions: use the mailing list. It is better for all of us.
  - [comnet2@winlab.rutgers.edu](mailto:comnet2@winlab.rutgers.edu)
  - **Apply for membership (if not done already):**
    [https://groups.google.com/a/winlab.rutgers.edu/forum/#!forum/comnet2](https://groups.google.com/a/winlab.rutgers.edu/forum/#!forum/comnet2)