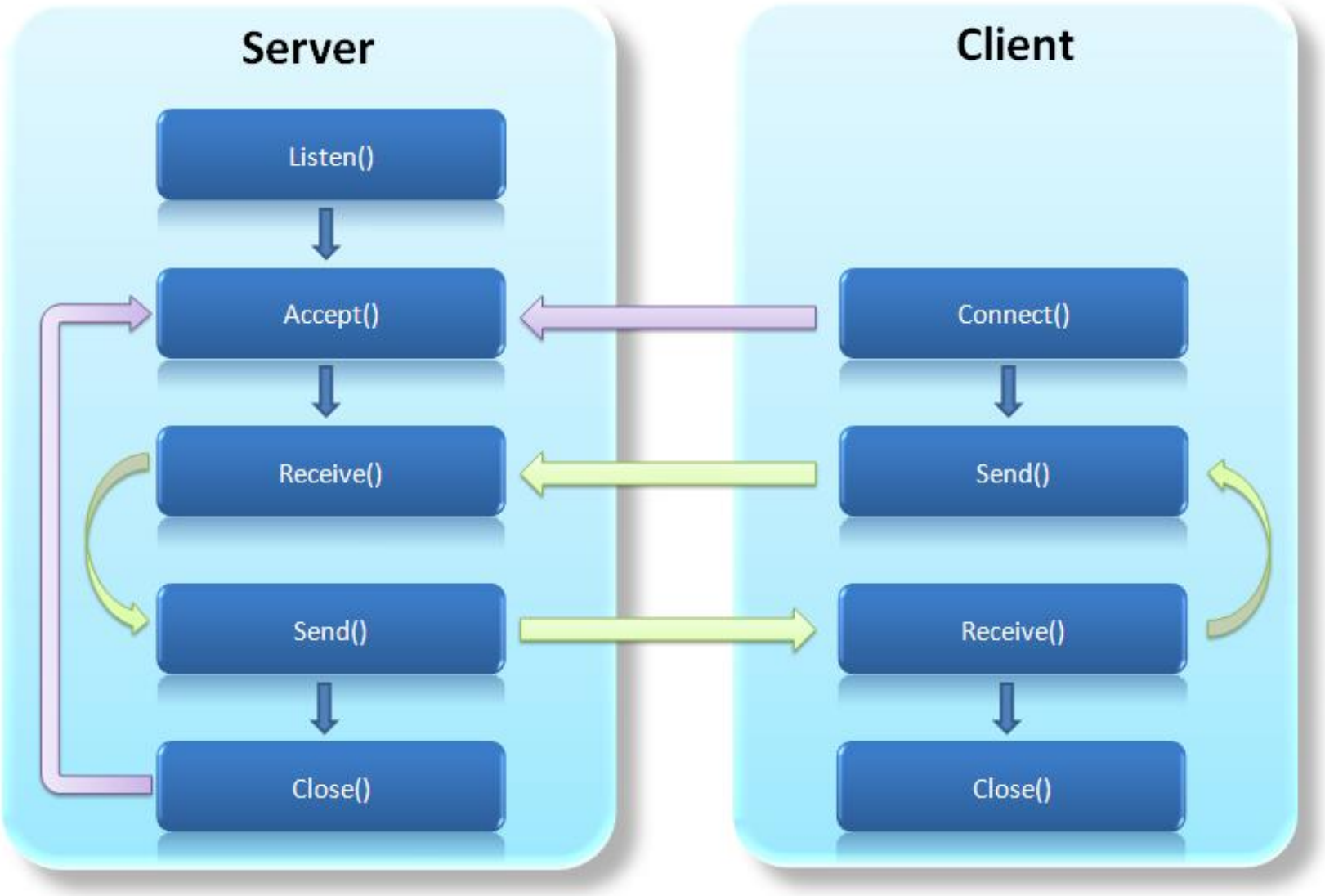
PYTHON – NETWORK PROGRAMMING

Partially from www.py4e.com/

Client Server Connection



Common service (listening) ports

21 FTP

22 SSH

23 Telnet

25 SMTP (Mail)

53 DNS

80 HTTP (Web)

110 POP3 (Mail)

443 HTTPS (web)

Network Socket

An endpoint of an inter-process communication flow across a computer network.

API set to the OS for network services

Since all communications between computers is based on the Internet Protocol most network sockets are internet sockets.

PYTHON SOCKET

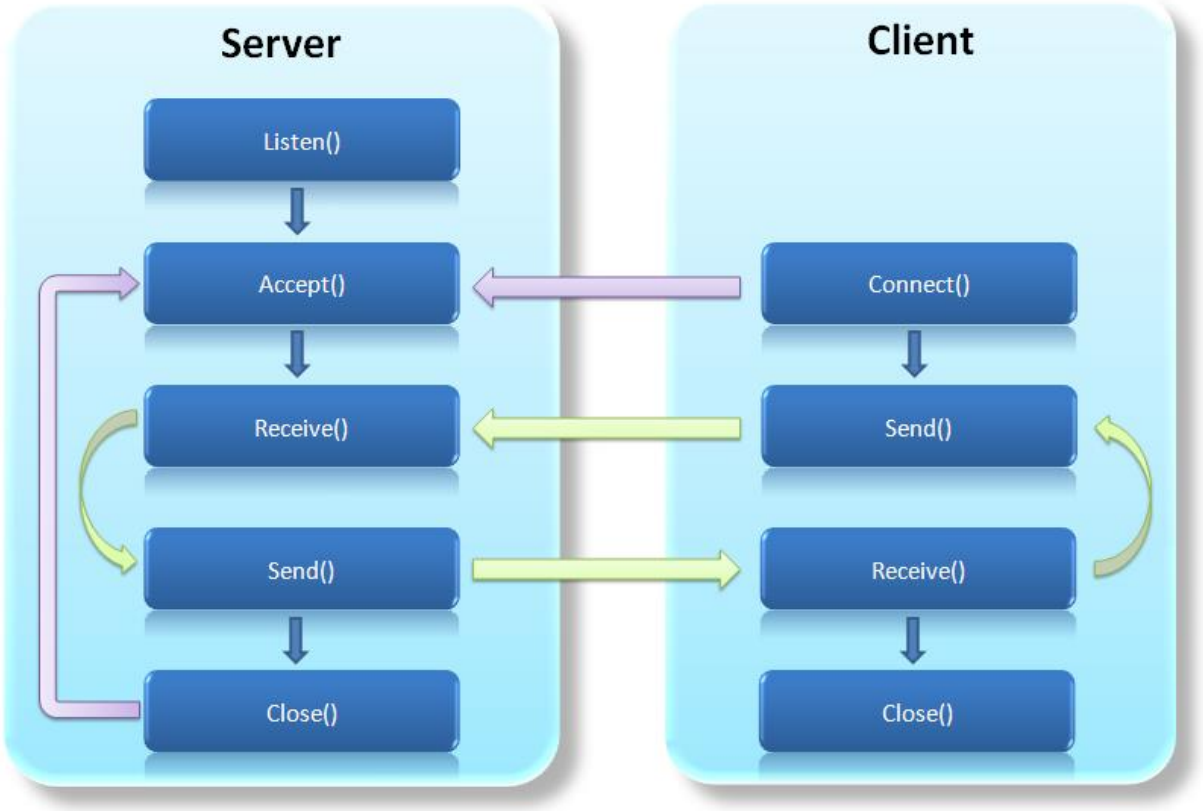
```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
```

- Address family:
  - `socket.AF_UNIX`
  - `socket.AF_INET`
  - `socket.AF_INET6`
- Socket Type:
  - `socket.SOCK_STREAM`
  - `socket.SOCK_DGRAM`
  - `socket.SOCK_RAW`

Client Server Connection

A certain web server has opened a socket and has binded it to port 80 and is listening.

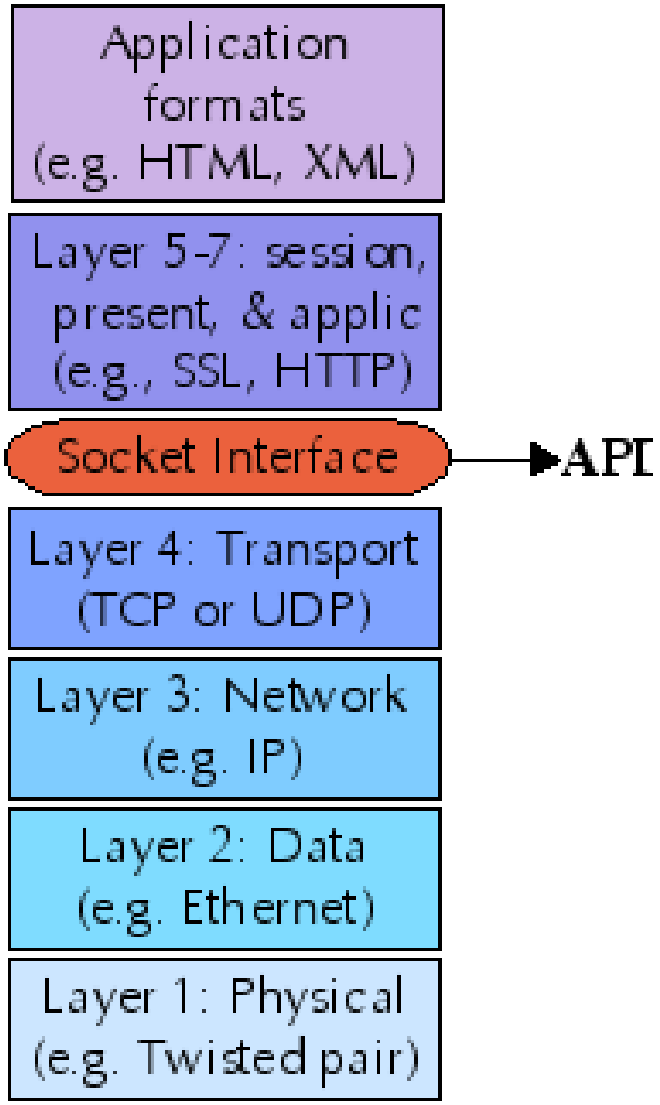
A syn packet arrives.. What happens next??



Chain of Events

- Each incoming packet causes chain of events
  - HW interrupt -> interrupt handler loads OS device driver which will transfer packet from NIC to RAM.
  - OS driver calls the stack via SW interrupts to process the packet.
  - IP process checks packet address and header CRC.
  - TCP process:
    - Connection Oriented: TCP (L4) will complete 3 way handshake before establishing connection.
    - Once connection complete, listening application is called.
    - All packets pass back and forth between TCP and application processes – TCP guarantees connection – retransmits missing packets etc.
  - Affiliated app process typically spawns a thread or fork to handle each successive concurrent connection.

Sockets in TCP Stack



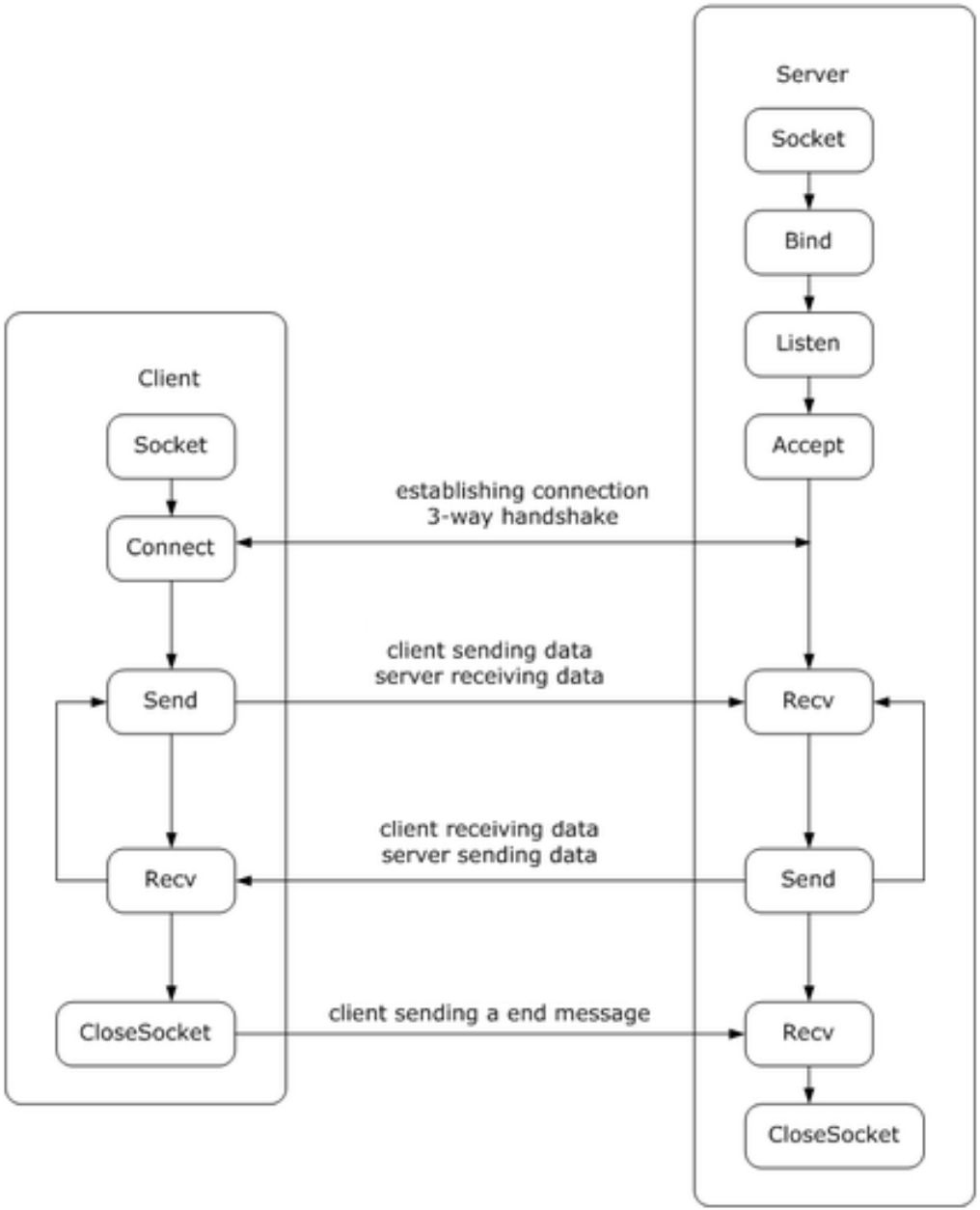
CREATE A SERVER

- To create a server, you need to:
  - create a socket.
  - bind the socket to an address and port.
  - listen for incoming connections & wait for clients.
  - accept a client.
  - send and receive data.

CREATE A CLIENT

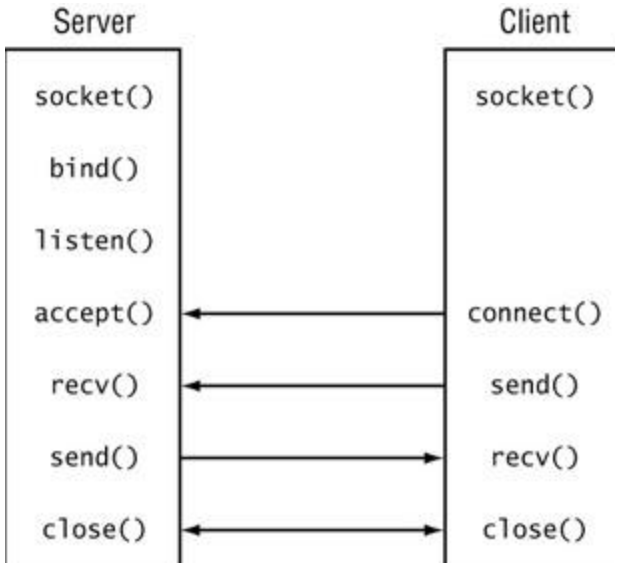
- To create a client, you need to:
  - create a socket
  - connect to the server
  - send and receive data

BIG PICTURE


Server
Socket
Bind
Listen
Accept
Client
Socket
Connect
establishing connection
3-way handshake
Send
client sending data
server receiving data
Recv
Recv
client receiving data
server sending data
Send
CloseSocket
client sending a end message
Recv
CloseSocket

Client Server Terminology

- Each endpoint is called a “host”.
- Each host is running an application which is using the sockets API.
- Servers use socket methods to:
  - “bind to ports” so they can “listen” for incoming connections.
  - “accept” tcp connections provided the 3-way handshake was acceptable
- Clients use socket methods to “open connections” to servers on service port
  - Clients use an “ephemeral port” as their own source.
- Both server and client use recv and send methods.



BASIC SOCKET MODULE

- The Python socket module provides direct access to the standard BSD socket interface.
- Socket addressing is simpler than more primitive means and much of the dirty work (buffer allocation) is done for you.
- But there are easier ways to open connection (e.g. netcat or nc)
  - `nc -l 2389`
  - `nc 192.168.100.2 2389`
  - `nc -k -l 2389`

Socket Method

```
s=socket(family, type[protocol])
```

socket family:

- `AF_UNIX`**:** A Unix socket allows two processes running on the same machine to communicate with each other through the socket interface.

- `AF_INET`**:** An IPv4 socket is a socket between two processes, potentially running on different machines

Socket Method

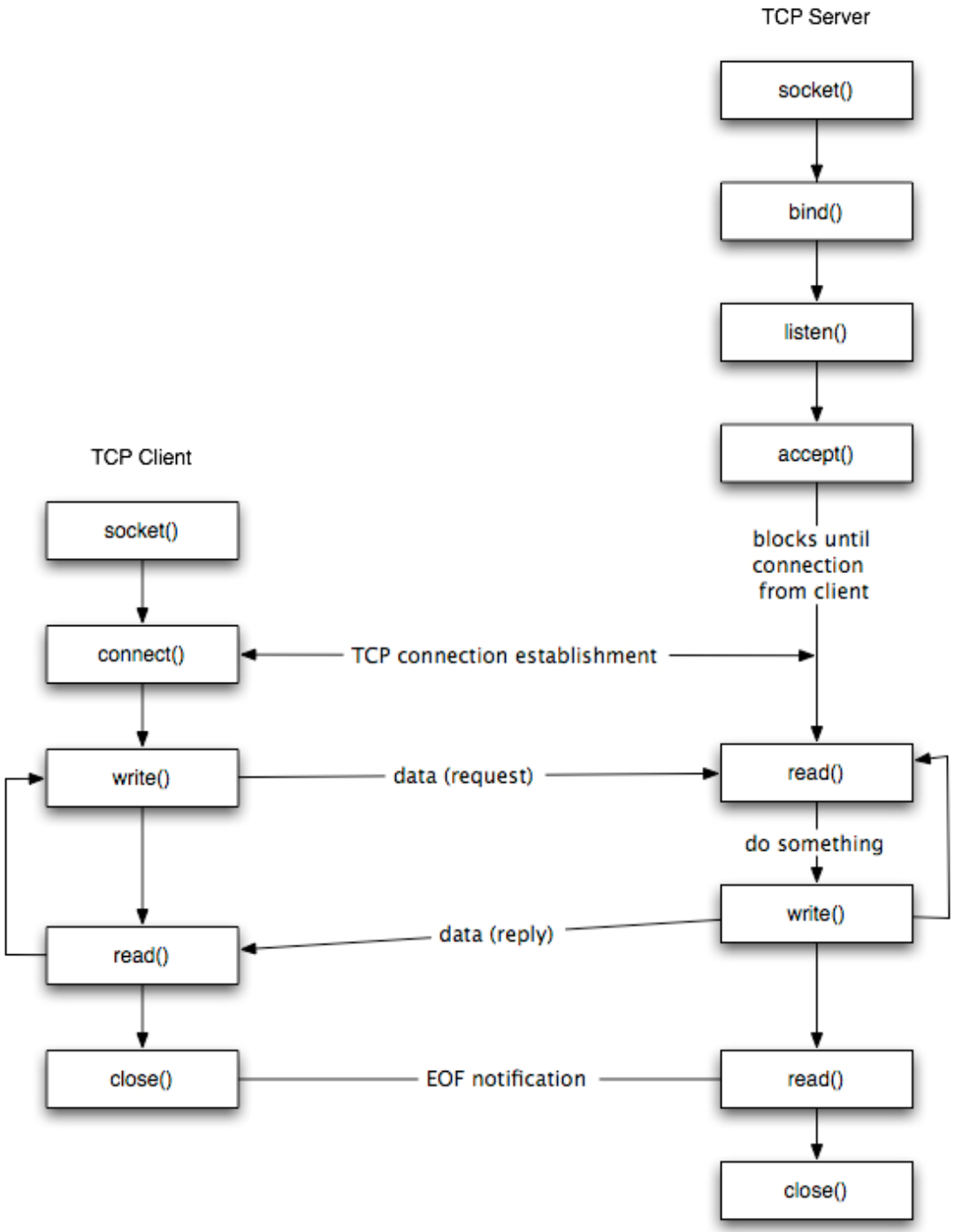
```
s=socket(family, type[protocol])
```

socket types :

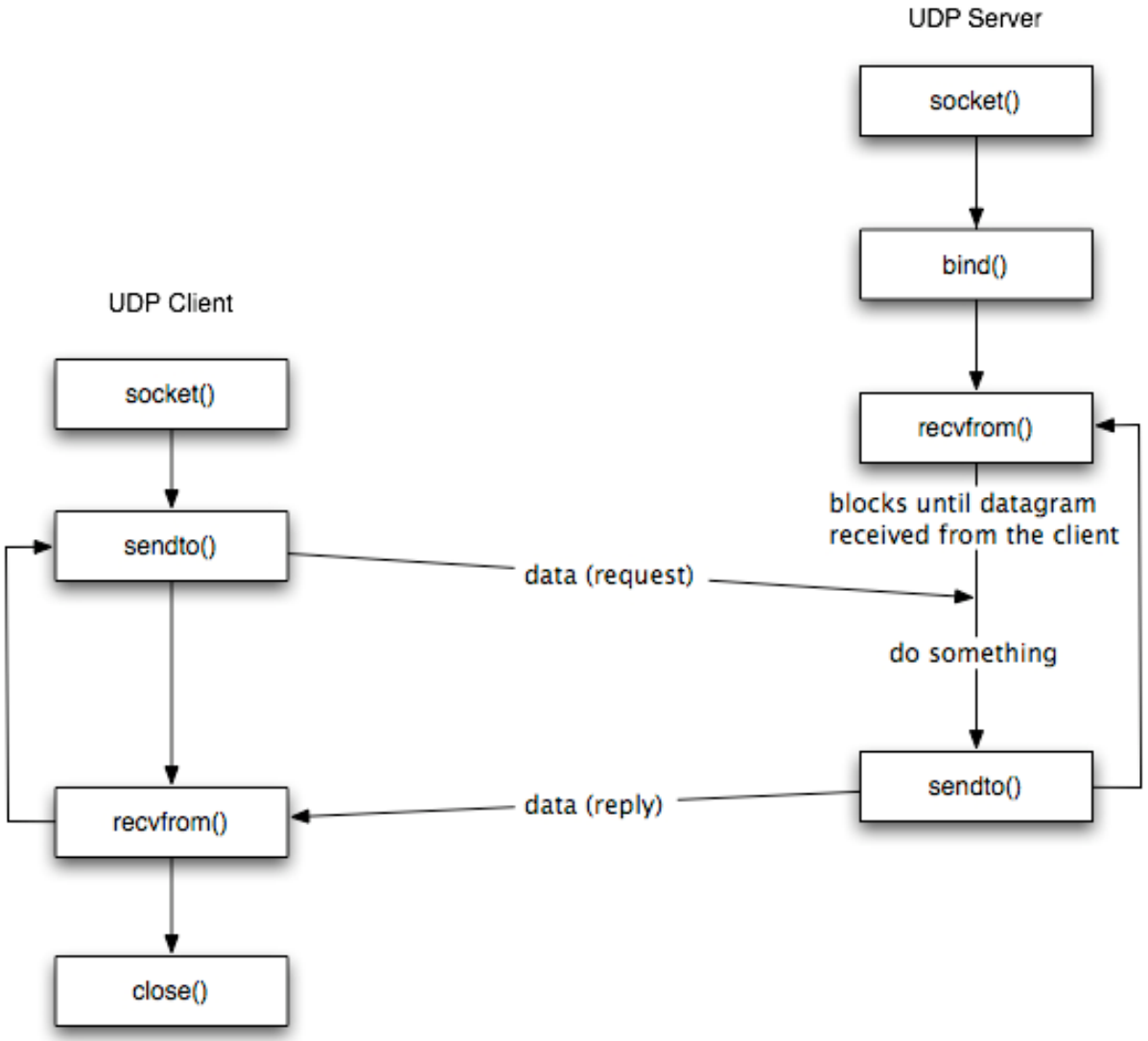
- `SOCK_STREAM` for TCP sockets
- `SOCK_DGRAM` for UDP sockets

- Once you create the socket, you can then use the object to call each of its methods:
  - `bind, listen, accept, connect`, etc.

TCP Connections



UDP Connection– Symmetrical!



Server Socket Methods

`s.bind()`

- Binds address (hostname, port number) to socket.

`s.listen()`

- Sets up and start TCP listener.

`s.accept()`

- Passively accepts TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

`s.connect()`

- This method actively initiates TCP server connection.

Basic Client-Server Socket Methods

`s.recv()`

- This method receives TCP message

`s.send()`

- This method transmits TCP message

`s.recvfrom()`

- This method receives UDP message

`s.sendto()`

- This method transmits UDP message

`s.close()`

- This method closes socket

`socket.gethostname()`

- Returns the hostname.

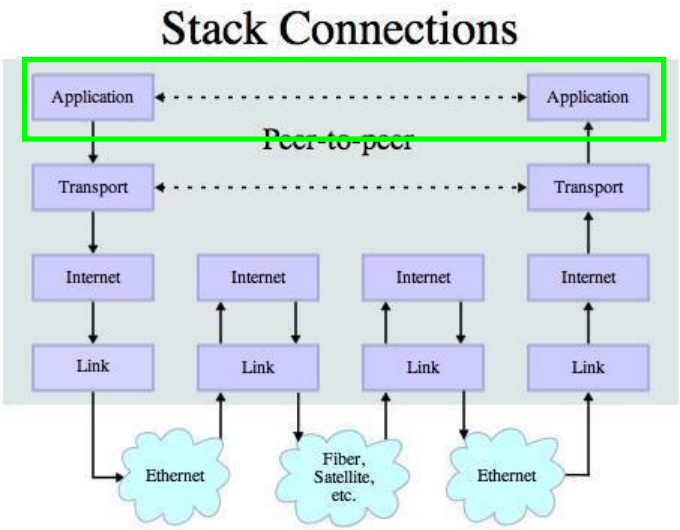
Exercise

A simple client connection to the Internet:

```
import socket
s = socket.socket(AF_INET,SOCK_STREAM)
s.connect(("www.google.ca",80))
request=bytes('GET /index.html
HTTP/1.0\n\n',"ascii")
s.send(request)
data = s.recv(10000)
print(data)
s.close()
```

Application Protocol

- Since TCP (and Python) gives us a reliable socket, what do we want to do with the socket? What problem do we want to solve?
- Application Protocols
  - Mail
  - World Wide Web



Source: http://en.wikipedia.org/wiki/Internet_Protocol_Suite

http://www.dr-chuck.com/page1.htm

protocol host document

GETTING DATA FROM THE SERVER

- Each time the user clicks on an anchor tag with an href= value to switch to a new page, the browser makes a connection to the web server and issues a “GET” request - to GET the content of the page at the specified URL
- The server returns the HTML document to the browser, which formats and displays the document to the user

Web Server

80

Browser

Web Server

80

Browser

Request

Web Server

80

GET http://www.dr-chuck.com/page2.htm

Browser

Request

Web Server

80

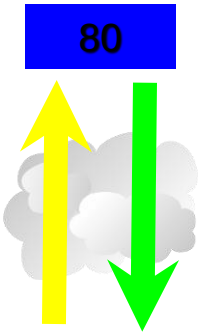
GET http://www.dr-chuck.com/page2.htm

Browser

Request

Web Server

Response



GET http://www.dr-chuck.com/page2.htm

<h1>The Second Page</h1> If you like, you can switch back to the First Page.

Browser

Request

Web Server

Response

80

GET http://www.dr-chuck.com/page2.htm

<h1>The Second Page</h1> If you like, you can switch back to the First Page.

Browser

Parse/
Render

MAKING AN HTTP REQUEST

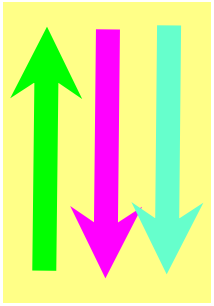
- Connect to the server like www.dr-chuck.com"
- Request a document (or the default document)
  - GET http://www.dr-chuck.com/page1.htm HTTP/1.0
  - GET http://www.mlive.com/ann-arbor/ HTTP/1.0
  - GET http://www.facebook.com HTTP/1.0

```
$ telnet www.dr-chuck.com 80
Trying 74.208.28.177...
Connected to www.dr-chuck.com.Escape character is '^]'.
GET http://www.dr-chuck.com/page1.htm HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 08 Jan 2015 01:57:52 GMT
Last-Modified: Sun, 19 Jan 2014 14:25:43 GMT
Connection: close
Content-Type: text/html

<h1>The First Page</h1>
<p>If you like, you can switch to
the <a href="http://www.dr-chuck.com/page2.htm">Second
Page</a>.</p>
```

Web Server



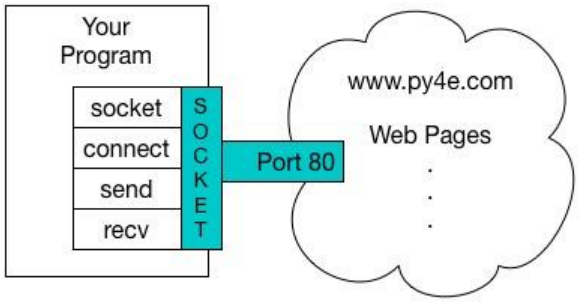
Browser

An HTTP Request in Python

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\r\n\r\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode(),end='')
mysock.close()
```



```
HTTP/1.1 200 OK
Date: Sun, 14 Mar 2010 23:52:41 GMT
Server: Apache
Last-Modified: Tue, 29 Dec 2009 01:31:22 GMT
ETag: "143c1b33-a7-4b395bea"
Accept-Ranges: bytes
Content-Length: 167
Connection: close
Content-Type: text/plain

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

HTTP Header

```
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    print(data.decode())
```

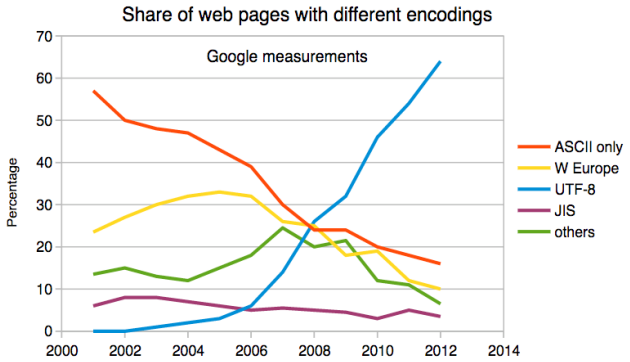
HTTP Body

MULTI-BYTE CHARACTERS

To represent the wide range of characters computers must handle we represent characters with more than one byte

- UTF-16 – Fixed length - Two bytes
- UTF-32 – Fixed Length - Four Bytes
- UTF-8 – 1-4 bytes
  - Upwards compatible with ASCII
  - Automatic detection between ASCII and UTF-8
  - UTF-8 is recommended practice for encoding data to be exchanged between systems

https://en.wikipedia.org/wiki/UTF-8



TWO KINDS OF STRINGS IN PYTHON

```
Python 2.7.10
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3.5.1
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

In Python 3, all strings are Unicode

PYTHON 2 VERSUS PYTHON 3

```
Python 2.7.10
>>> x = b'abc'
>>> type(x)
<type 'str'>
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
```

```
Python 3.5.1
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
```

PYTHON 3 AND UNICODE

- In Python 3, all strings internally are UNICODE
- Working with string variables in Python programs and reading data from files usually "just works"
- When we talk to a network resource using sockets or talk to a database we have to encode and decode data (usually to UTF-8)

```
Python 3.5.1
>>> x = b'abc'
>>> type(x)
<class 'bytes'>
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
```

PYTHON STRINGS TO BYTES

- When we talk to an external resource like a network socket we send bytes, so we need to encode Python 3 strings into a given character encoding
- When we read data from an external resource, we must decode it based on the character set so it is properly represented in Python 3 as a string

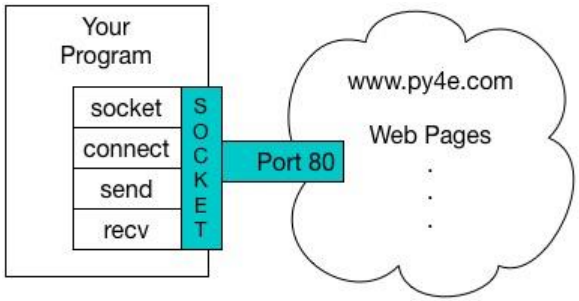
```
while True:
    data = mysock.recv(512)
    if ( len(data) < 1 ) :
        break
    mystring = data.decode()
    print(mystring)
```

An HTTP Request in Python

```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```



`bytes.`**`decode`**`(`*`encoding="utf-8", errors="strict"`*`)`

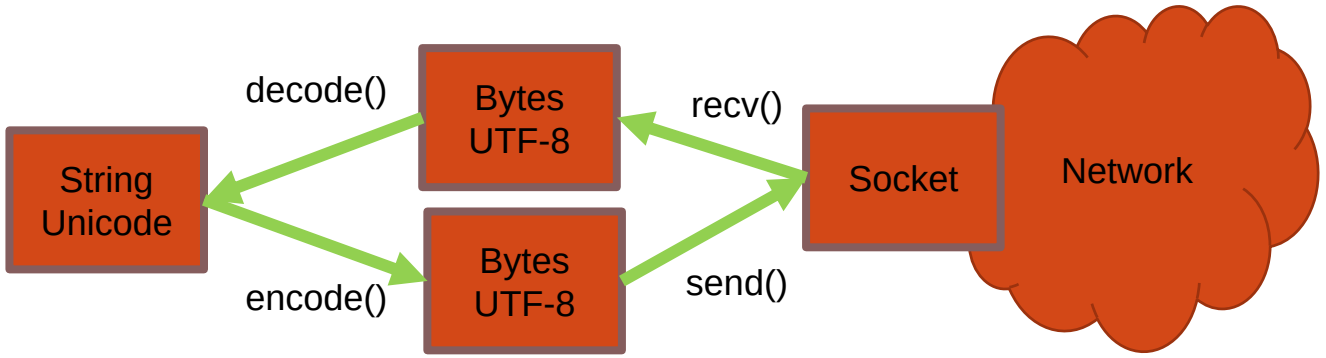
`bytearray.`**`decode`**`(`*`encoding="utf-8", errors="strict"`*`)`

Return a string decoded from the given bytes. Default encoding is `'utf-8'`. *errors* may be given to set a different error handling scheme. The default for *errors* is `'strict'`, meaning that encoding errors raise a `UnicodeError`. Other possible values are `'ignore'`, `'replace'` and any other name registered via `codecs.register_error()`, see section Error Handlers. For a list of possible encodings, see section Standard Encodings.

`str.`**`encode`**`(`*`encoding="utf-8", errors="strict"`*`)`

Return an encoded version of the string as a bytes object. Default encoding is `'utf-8'`. *errors* may be given to set a different error handling scheme. The default for *errors* is `'strict'`, meaning that encoding errors raise a `UnicodeError`. Other possible values are `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'` and any other name registered via `codecs.register_error()`, see section Error Handlers. For a list of possible encodings, see section Standard Encodings.

https://docs.python.org/3/library/stdtypes.html#bytes.decode
https://docs.python.org/3/library/stdtypes.html#str.encode




```
import socket

mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('data.pr4e.org', 80))
cmd = 'GET http://data.pr4e.org/romeo.txt HTTP/1.0\n\n'.encode()
mysock.send(cmd)

while True:
    data = mysock.recv(512)
    if (len(data) < 1):
        break
    print(data.decode())
mysock.close()
```

Using urllib in Python

Since HTTP is so common, we have a library that does all the socket work for us and makes web pages look like a file

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())
```

urllib1.py

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')
for line in fhand:
    print(line.decode().strip())
```

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief

urllib1.py

Like a File...

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://data.pr4e.org/romeo.txt')

counts = dict()
for line in fhand:
    words = line.decode().split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1
print(counts)
```

urlwords.py

Reading Web Pages

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://www.dr-chuck.com/page1.htm')
for line in fhand:
    print(line.decode().strip())
```

```
<h1>The First Page</h1>
<p>If you like, you can switch to the <a
href="http://www.dr-chuck.com/page2.htm">Second
Page</a>.
</p>
```

urllib2.py

Following Links

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://www.dr-chuck.com/page1.htm')
for line in fhand:
    print(line.decode().strip())
```

```
<h1>The First Page</h1>
<p>If you like, you can switch to the <a
href="http://www.dr-chuck.com/page2.htm">Second
Page</a>.
</p>
```

urllib2.py

The First Lines of Code @ Google?

```
import urllib.request, urllib.parse, urllib.error

fhand = urllib.request.urlopen('http://www.dr-chuck.com/page1.htm')
for line in fhand:
    print(line.decode().strip())
```