Prep Learning:-

* Network, types of network (General Awareness)
* Internet concepts (General Awareness)
* Network Models - OSI (or) TCP/IP model
    * Layers - Physical + Link, Network/IP layer, Transport, Application Layer
* IP layer - IP address
* Transport - TCP, UDP protocols, port concept
* Application - HTTP, FTP and many more

Terminology:-

* client - server
  server should be running ahead of client
  client should well aware of server,
  need not be vice versa until request comes from a client

Weel Defined port numbers for popular services:-

* HTTP - 80
* FTP - 21
* HTTPS - 443
* SSH - 22
* TELNET - 23
* Custom - 8080

IP Address format - classes of IP address a.b.c.d   , where a,b,c,d are in range of 0-255 (8 bits each, total 32 bit)

192.168.0.1

human friendly names, web compatible - hostnames

hostname to IP address ==> Name Service, DNS

Physical Address - MAC Address (48 bit number, 6 pairs of hex digits)
Logical Address - IP Address

localhost , loopback address : 127.0.0.1

Public IP Address / Private address (10, 172, 192), in local network

NAT - Network Address Translation

IP Address, Port number
TCP, UDP

TCP vs UDP:-
* Reliable,only By TCP - Acknowledgement, ReTransmission,
  Fragmentation & Arranging Back
  Flow Control

Both TCP & UDP - multiplexing of data, checksum

TCP - connection oriented protocol
UDP - connection less protocol

TCP - segment,
UDP - datagrams

URL format

http://www.example.com/abc/test.html?x=25&y=40
http://www.example.com:8080/abc/test.html?x=25&y=40

protocol/scheme : http
hostname/addr : example.com
port no : 80
path : abc/test.html
query string : x=25&y=40

Python Socket Examples - TCP, UDP
* TCP Server & TCP Client
* UDP Receiver & Sender

AF_INET
 - IPv4
SOCK_STREAM)
 - TCP

Socket - IP Address + Port number

TCP Server :-

Step-1:- Create an empty socket

```
        ssd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Step-2:- Fill IP and Port number (bind)

```
        addr = ("127.0.0.1", 1500)              # tuple
        ssd.bind(addr)
```

Step-3:- use socket for server purpose (Passive socket)

```
        ssd.listen()
```

Step-4:- wait for client connection and accept

```
        clisock , addr = ssd.accept()
```

Step-5:-

```
        clisock.recv(MAX_SIZE)   (or)   clisock.send(data,len)
```

Step-6:-

```
    clisock.close()
    ssd.close()
```

simple client:-

```
        telnet 127.0.0.1 1500
```

Account - id, name, balance

Destructor - Invoked just before object is going out of scope
Not meant for meant releasing memory of regular object data
It's meant for any reversal, for special initialization done
in ctor - file open, socket open, db connect

Checklist:-

* TCP Server & Client
  - simple socket ctor call
  - with statement : socket call
  - own class (OO Approach)
  - __enter__, __exit__, in your call
* Simple UDP sender & receiver