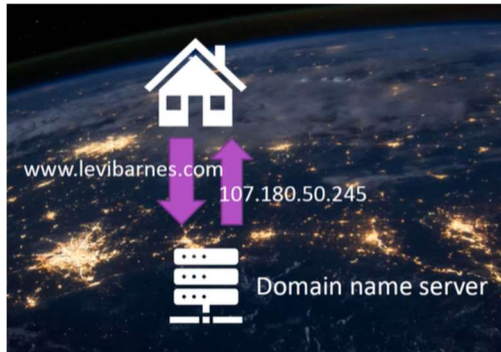


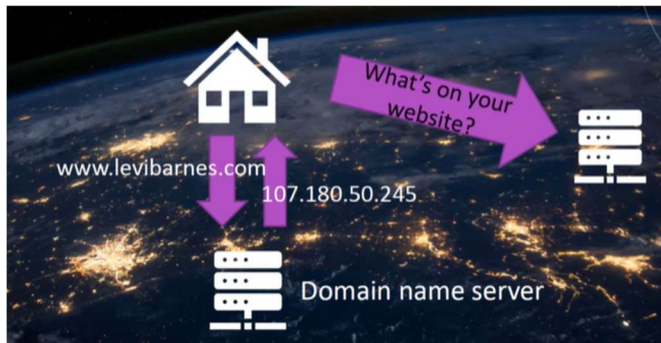
## Vid 45 - Internet Basics

Sockets and Protocols (TCP and IP) that will enable us to build a wallet and a miner

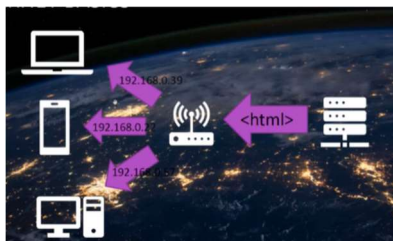


DNS provide IP addresses for anything you want to search??

IP address is a set of 4 #'s between 0 and 255. IP address tells the entire internet where a specific computer is found



When levibarns responds w/ here's the HTML code for my website, it first communicates w/ your gateway → communicates w/ ISP and also w/ devices in your house (phone, laptop, desktop some on ethernet some on wifi) and their all going to communicate thru this one point called the gateway

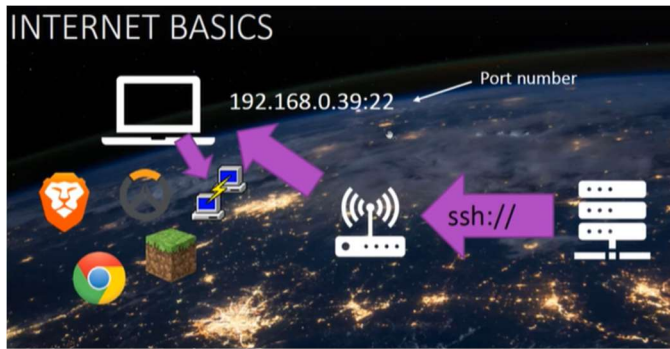


Router assigns all these devices a similar IP address

**See 3:27** in video if need to conceptualize different types of data needing to be recognized for certain applications

### Port numbers for crypto wallet and miner

So, when we create, our app were going to send it over a port number that were going to specify is specifically the port number for our crypto wallet or crypto miner



**Protocol** → set of rules for communicating between computers

- Specify when data packets are sent, there is usually a header to those packets and the contents of that header will be specified by that protocol

**TCP** → Transmission Control Protocol

- Handles the packeting of data
- When you want to send something really big
  - Responsible for breaking that up into little packets and then are sent individually \
  - Also ensuring that they arrive on time and for checking the integrity
- Features confirmed delivery – you know that each of those packets is getting to the right place, if not it gets re-sent or you get an ERROR, so you know it didn't go thru

**IP** → Internet Protocol

- Handles routing – sends the packets to the right place and manages those IP addresses and bounces off the servers on the way that your interested in

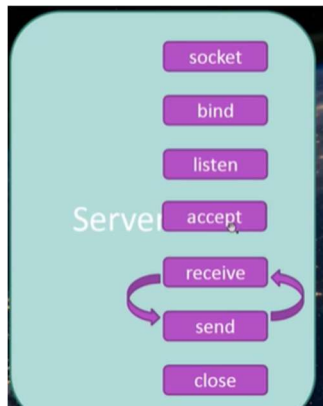
## Vid 46 – Client and Server

Socket → stores the info necessary to communicate between two different computers (where computers located, port #, and the info necessary to facilitate that communication)

**Server Client Model** → a server essentially waits and listens for a connection from a client

Building a simple echo server for the first example → the client is going to send some data and the server is going to echo it back to the client

### Server Side



To start communicating with a client a server needs to create a socket and then bind that socket to an IP address and a Port # (where the data should go once it gets to your computer)

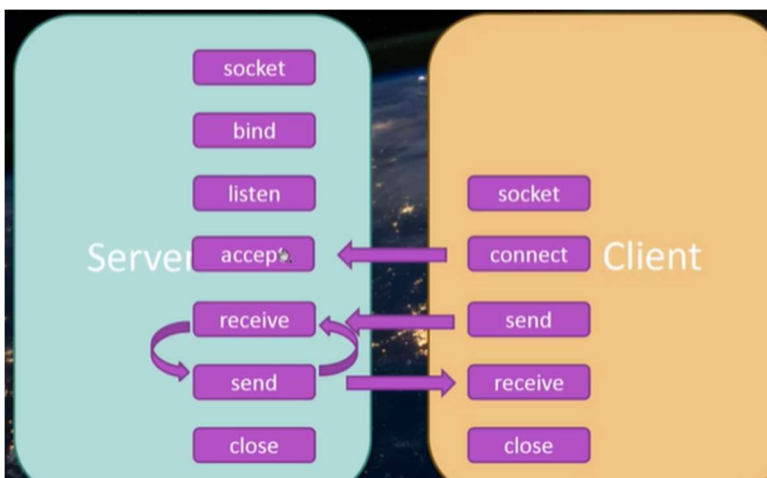
Set our server socket to listen and that socket will listen until a client connects to it – once connected it will **accept** that connection, receive some data from the client, then it can send some data

You can send and receive data from either end however many times until you decide to close the connection

### Client Side

Client side sees the socket and immediately says I know who I want to connect to – so it connects to it with an IP address and a port #

So then the Socket on the other side sees an IP address that its bound to and then accepts the connection



### Vid 47 - Building a server

Local IP address, port number made up sometimes can cause collisions but NBD

socket.AF\_INET → specifies that your using IP version 4

socket.SOCK\_STREAM → specifies that your using TCP

- There are other things out there, but these are good for 99% of the things that were gonna do in this course

```
import socket

ip_addr = '192.168.0.39'
port = 5005

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind((ip_addr, port))

s.listen()
conn, addr = s.accept()
while True:
    data = conn.recv()
    if not data: break
    print ("Rec'd: " + data)
    conn.send(data)
s.close()
```

When we accept a brand-new socket gets created

When the client it done sending data it'll send an empty str and when that happens, we break the loop

- Receive data
- Print it
- Sent it back to the place it came from

### Vid 48 - Building a client

BUFFER\_SIZE specifies how much data is going to be sent

```
ip_addr = '192.168.0.39'
port = 5005
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((ip_addr, port))

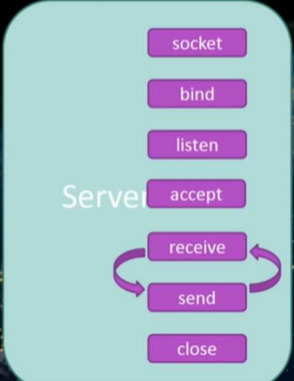
s.send("Hello! World")
data = s.recv(BUFFER_SIZE)
print ("Rec'd: " + data)

s.close()
```

## Vid 49 – Non-Blocking Server

49. Non-blocking server

SOCKET COMMUNICATION



```
import socket

ip_addr = '192.168.0.39'
port = 5005

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.bind((ip_addr, port))

s.listen()
for q in range(10):
    conn, addr = s.accept()
    while True:
        data = conn.recv()
        if not data: break
        print ("Rec'd: " + data)
        conn.send(data)

s.close()
```

```
s.listen()
for q in range(10):
    rd, wt, err = select.select([s], [], [s], 6) #6 sec timeout
    if s in rd:
        conn, addr = s.accept() #Blocking!
        while True:
            data = conn.recv()
            if not data: break
            print ("Rec'd: " + data)
            conn.send(data)
```

[Inputs][Outputs][Error]timeout

Passing list of sockets that can be read | list of sockets that can be written to | a list of sockets that can return an error message | timeout

## Vid 51 – Pickling data for communication

```
server.py - C:/Users/Russ/Python36/server.py (3.6.4)
File Edit Format Run Options Window Help

# Server.py
import TxBlock
import socket
import pickle

TCP_PORT = 5005

def recvObj(ip_addr):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((ip_addr, TCP_PORT))
    s.listen()
    new_sock, addr = s.accept()
    data = new_sock.recv()
    return pickle.loads(data)

    return None

if __name__ == "__main__":
    newB = recvObj('localhost')
    print(newB.data[1])
    print(newB.data[2])
    if (newB.is_valid()):
        print("Success. Tx is valid")
    else:
        print("ERROR. Tx invalid")
    if newB.data[0].inputs[0][1] == 2.3:
        print("Success. Input value matches")
    else:
        print("ERROR. Wrong input value for block 1, tx 1")

client.py - C:/Users/Russ/Python36/client.py (3.6.4)
File Edit Format Run Options Window Help

# Client.py
import TxBlock
import Transactions
import Signatures
import pickle
import socket

TCP_PORT = 5005

def sendBlock(ip_addr, blk):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip_addr, TCP_PORT))
    data = pickle.dumps(blk)
    s.send(data)
    return False

if __name__ == "__main__":
    pr1, pu1 = Signatures.generate_keys()
    pr2, pu2 = Signatures.generate_keys()
    pr3, pu3 = Signatures.generate_keys()
    Tx1 = Transactions.Tx()
    Tx1.add_input(pu1, 2.3)
    Tx1.add_output(pu2, 1.0)
    Tx1.add_output(pu3, 1.1)
    Tx1.sign(pr1)

    Tx2 = Transactions.Tx()
    Tx2.add_input(pu3, 2.3)
    Tx2.add_output(pu2, 1.0)
    Tx2.add_output(pu1, 3.1)
    Tx2.sign(pr2)
    Tx2.sign(pr3)

    B1 = TxBlock.TxBlock(None)
    B1.addTx(Tx1)
    B1.addTx(Tx2)

    sendBlock('localhost', B1)
```

```
Microsoft Windows [Version 10.0.22000.556]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Russ>cd Python36\
C:\Users\Russ\Python36>python server.py
C:\Users\Russ\Python36>python server.py
File "server.py", line 26
    if (newB.data[0].inputs[0][1] == 2.3:
        ^
SyntaxError: invalid syntax

C:\Users\Russ\Python36>python server.py
Traceback (most recent call last):
  File "server.py", line 40, in <module>
    newB = recvObj('localhost')
  File "server.py", line 13, in recvObj
    data = new_sock.recv()
TypeError: recv() takes at least 1 argument (0 given)

C:\Users\Russ\Python36>
```

```
REQD:
SIGS:
b'\xb3p\x99\x98PF\xb8\xedk4\xd2z\xd7\x94W\x17e\xab[y\xff
xc4\x0f\xfk\xcd\x85\xbd\xeeC\x06\x07P\xbf\xe1\x0eC\x
d\xbc\x18\x03\x13\x19\xa5\xa8\n\xfa*\xa7\x92f\xdcB\xb7\
fff^\xdav\xd4s8\xf6\x7f\xd5\xcaGr\x1ec\xd3\x95\xde\xd2QP
6F\x92\x89X\xe6\x05\xe8\x98mx6\xba\xfd0\\\xd8ZH\xe7
c\x0c\x98\xd0\xb5\x1c\x04\xd7\x7f\x06\xe2\xda\xa9a\x00\x
P\x120\xe5\xce\xffMm\xa1+\x8f'
b'\x8b[\xb1\x9b5\x94\xa9_X\x88\xe84\x19\xcb\xdd\x8fr\x0e
1H\x06Z\xdd\x97\x90\x8f\x9a\xe6\xbf\xab\xea\x9e$B\xc8\xa
\xce\x0f\xd3\xbeG$xae\x10s!\xa5\xd3uj\xb1\x95\xd1\xbb\
x85b\xec\x17\x86\x1c\x02,\xad\xaa\xa2q\xe3\xef\xc6\xe8\x
0\xb20\xa6\x0fE\x1a"\xaa\xae\x13\x86\xf1\x1a\x80\x01\xc3
feFSo\xc11G0{s\x92\xa7\x87\xca\xcd\x9dz8\xa8\x05\xb3F\x0
bb4\x8d'
END

Success. Tx is valid
Success. Input value matches
Success. Output value matches
Success. Input value matches
Traceback (most recent call last):
  File "server.py", line 42, in <module>
    if newB.data[1].inputs[1][1] == 1.0:
IndexError: list index out of range

C:\Users\Russ\Python36>
```



## Vid 53 – Miner and Wallet

### Wallet

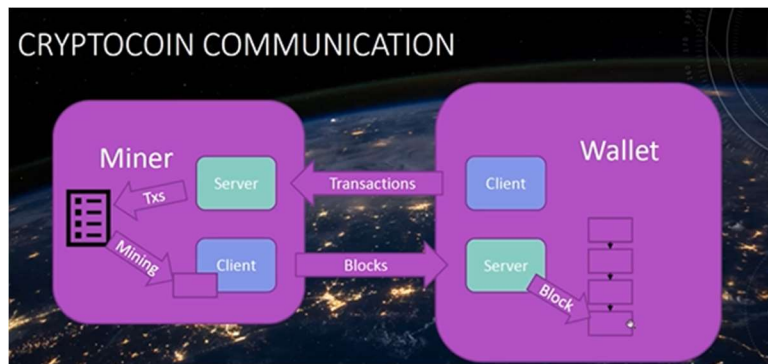
- ➔ manages the coins that belong to one person
- ➔ For each of your crypto currencies the wallet will usually store a number or private keys that control your coins

One of the groups that will pick-up transactions from your wallet is the miners. Miners are set up with a server to receive these transactions where they will be put into a list.

- ➔ They will be able to check to make sure they are valid transaction's when they come in

The miner can assemble a list of transactions into a block and hunt for a nonce. Once it finds a nonce the miner will also need a client to send those brand-new blocks among other people, their wallets out in the world

This means that the wallets also need a server on their side to receive these blocks and then put them into their own block chain



Most crypto wallets don't usually store the entire blockchain that called a full node

But in our coin, we're going to assume that every wallet does store the entire blockchain

## Vid 54 – Building a socket and communication utility

Utility for socket communications

Test case for SocketUtils is freestanding meaning that it doesn't have a client on the other end. We're going to test whether it can return w/out blocking.

```
*SocketUtils.py - C:/Users/Russ/Python36/SocketUtils.py (3.6.4)*
File Edit Format Run Options Window Help

# SocketUtils
import socket
import pickle
import select

TCP_PORT = 5005

def newServerConnection(ip_addr):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((ip_addr, TCP_PORT))
    s.listen()
    return s

def recvObj(socket):
    inputs, outputs, errs = select.select([socket], [], [socket], 6)
    if socket in inputs:
        new_sock, addr = socket.accept()
        all_data = b''
        while True:
            data = new_sock.recv(BUFFER_SIZE)
            if not data: break
            all_data = all_data + data
        return pickle.loads(all_data)
    return None

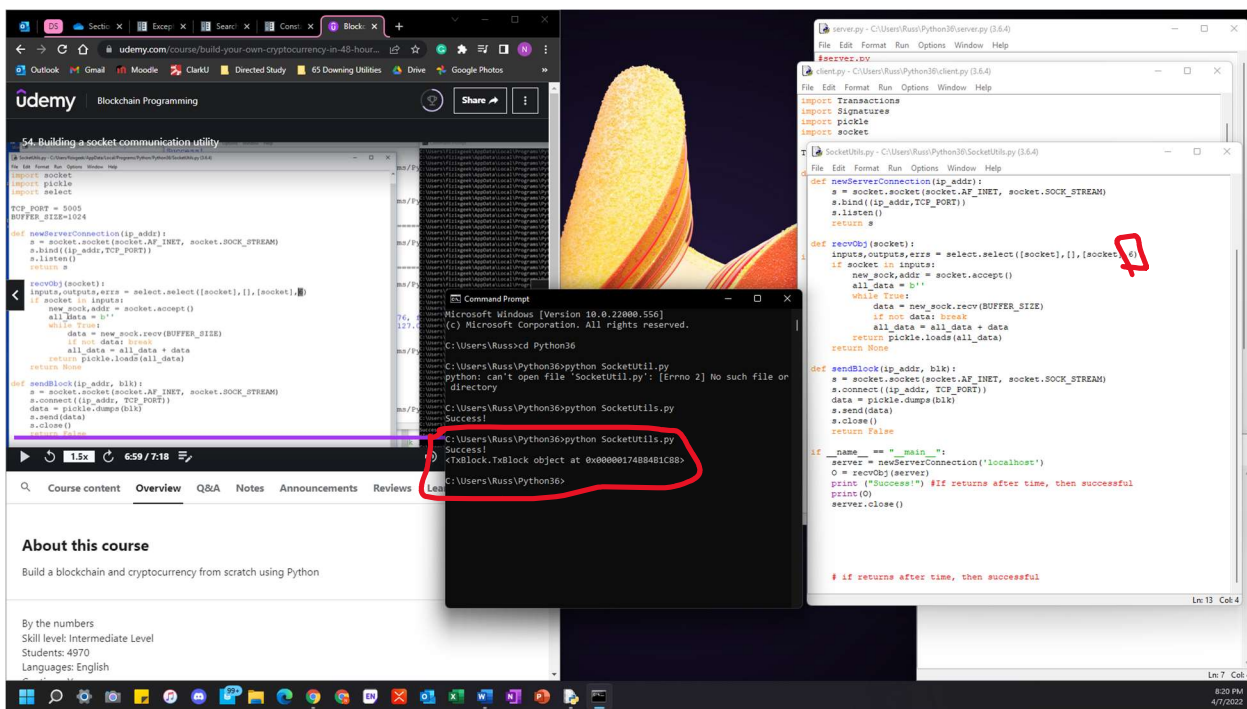
def sendBlock(ip_addr, blk):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip_addr, TCP_PORT))
    data = pickle.dumps(blk)
    s.send(data)
    s.close()
    return False

if __name__ == "__main__":
    server = newServerConnection('localhost')
    O = recvObj(server)
    print("Success!") # if returns after time, then successful
    server.close()
```

Problem here is that accept () call is blocking → need to use select

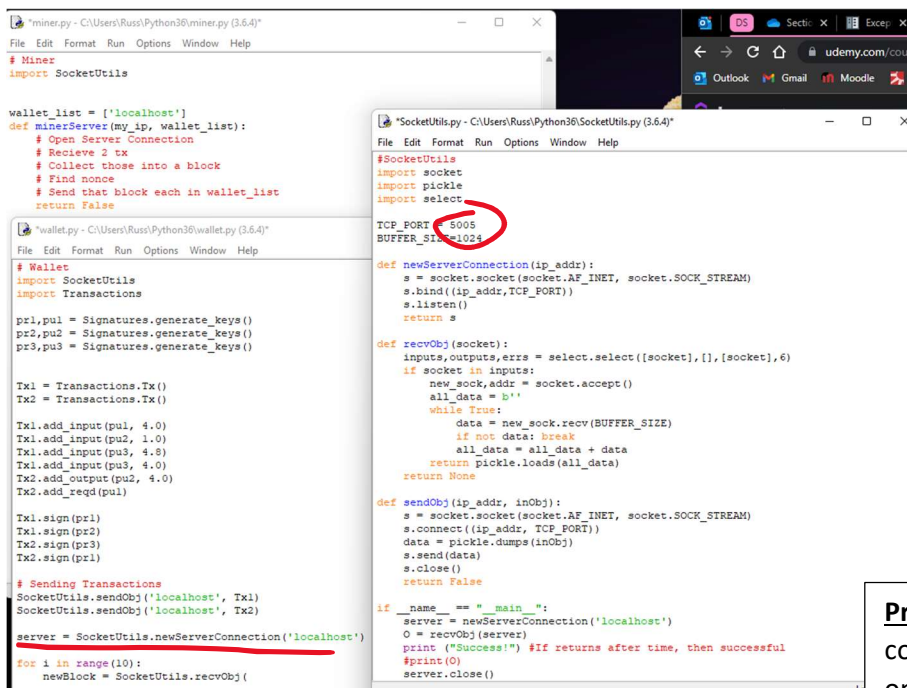
Also must close the server otherwise we'll get an error when we try to open something on the same IP address w/ the same port





Socket gives us 6 seconds of listening w/ out blocking and then successfully receives things at the other end.

## Vid 55 – Build a Miner



Right now, we have 1 TCP port and its communicating everything over the port.

Possibly could use one port to go client to server and the other to go server to client

**Problem:** We're going to try to open a server connection on the **miner side**, then we're going to try to open a server connection on the **wallet side (client side)** as well. → This is an ERROR b/c its going to say you can't open 2 different connections on the same IP address from the same machine.

Our client server program tries to mimic 2 machines even though its only one. (9:47)

## Vid 56 – Solution

```
SocketUtils.py - C:\Users\Russ\Python36\SocketUtils.py (3.6.4)
File Edit Format Run Options Window Help

import socket
import pickle
import select

TCP_PORT = 5005
BUFFER_SIZE=1024

def newServerConnection(ip_addr, port=TCP_PORT):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((ip_addr, port))
    s.listen()
    return s

def recvObj(socket):
    inputs, outputs, errs = select.select([socket], [], [socket], 6)
    if socket in inputs:
        new_sock, addr = socket.accept()
        all_data = b''
        while True:
            data = new_sock.recv(BUFFER_SIZE)
            if not data: break
            all_data = all_data + data
        return pickle.loads(all_data)
    return None

def sendObj(ip_addr, inObj, port=TCP_PORT):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ip_addr, port))
    data = pickle.dumps(inObj)
    s.send(data)
    s.close()
    return False

if __name__ == "__main__":
    server = newServerConnection('localhost')
    O = recvObj(server)
    print("Success!") #if returns after time, then successful
    #print(O)
    server.close()
```

```
Wallet.py - C:\Users\Russ\Python36\Wallet.py (3.6.4)
File Edit Format Run Options Window Help

tx1 = Transactions.Tx()
tx2 = Transactions.Tx()

tx1.add_input(pu1, 4.0)
tx1.add_input(pu2, 1.0)
tx1.add_input(pu3, 4.0)
tx2.add_input(pu3, 4.0)
tx2.add_output(pu2, 4.0)
tx2.add_reqd(pu1)

tx1.sign(pr1)
tx1.sign(pr2)
tx2.sign(pr3)
tx2.sign(pr1)

try:
    SocketUtils.sendObj('localhost', tx1)
    SocketUtils.sendObj('localhost', tx2)
except:
    print("Error! Connection unsuccessful")

server = SocketUtils.newServerConnection('localhost', 5006)
for i in range(10):
    newBlock = SocketUtils.recvObj(server)
    if newBlock:
        break
server.close()

if newBlock.is_valid():
    print("Success! Block is valid")
if newBlock.good_nonce():
    print("Success! Nonce is valid")
for tx in newBlock.data:
    if tx == tx1:
        print("Tx1 is present")
    if tx == tx2:
        print("Tx2 is present")
```

Solves the issue with the same port

Ln: 15 Col: 0

What if we wait for our objects and they don't come?? (3:54)

```
Miner.py - C:\Users\Russ\Python36\Miner.py (3.6.4)
File Edit Format Run Options Window Help

#Miner
import SocketUtils
import Transactions
import TxBlock
import Signatures

wallets = ['localhost']
tx_list = []
head_block = None
def findLongestBlockchain():
    longest = -1
    long_head = None
    for b in head_blocks:
        current = b
        this_len = 0
        while current != None:
            this_len = this_len + 1
            current = current.previousBlock
        if this_len > longest:
            long_head = b
            longest = this_len
    return long_head

def minerServer(my_ip, wallet_list, my_public):
    server = SocketUtils.newServerConnection(my_ip)
    # Get 2 Txs from wallets
    for i in range(10):
        newTx = SocketUtils.recvObj(server)
        if isinstance(newTx, Transactions.Tx):
            tx_list.append(newTx)
            print("Recd tx")
            if len(tx_list) >= 2:
                break
    # add Txs to new block
    newBlock = TxBlock.TxBlock(findLongestBlockchain())
    newBlock.addTx(tx_list[0])
    newBlock.addTx(tx_list[1])
    # Compute and add mining reward
    total_in, total_out = newBlock.count_totals()
    mine_reward = Transactions.Tx()
```

```
Wallet.py - C:\Users\Russ\Python36\Wallet.py (3.6.4)
File Edit Format Run Options Window Help

tx1.add_input(pu2, 1.0)
tx1.add_output(pu3, 4.0)
tx2.add_input(pu3, 4.0)
tx2.add_output(pu2, 4.0)
tx2.add_reqd(pu1)

tx1.sign(pr1)
tx1.sign(pr2)
tx2.sign(pr3)
tx2.sign(pr1)

try:
    SocketUtils.sendObj('localhost', tx1)
    print("Sent Tx1")
    SocketUtils.sendObj('localhost', tx2)
    print("Sent Tx2")
except:
    print("Error! Connection unsuccessful")

server = SocketUtils.newServerConnection('localhost', 5006)
for i in range(30):
    newBlock = SocketUtils.recvObj(server)
    if newBlock:
        break
server.close()

if newBlock.is_valid():
    print("Success! Block is valid")
if newBlock.good_nonce():
    print("Success! Nonce is valid")
for tx in newBlock.data:
    try:
        if tx.inputs[0][0] == pu1 and tx.inputs[0][1] == 4.0:
            print("Tx1 is present")
    except:
        if tx.inputs[0][0] == pu3 and tx.inputs[0][1] == 4.0:
            print("Tx2 is present")

for b in head_blocks:
    if newBlock.previousHash == b.computeHash():
```

## **Vid 57 – Placing the New Block**

Miners going to need to store a decent blockchain and its gonna need to listen for new blocks

Going to explore concurrency, computer has several cores in its CPU and we need to use those cores to do things like:

Were going to have a separate thread that's running a miner on the miner side, the client and possibly the server, all together so that we can continue receiving transactions and putting them in the transaction list and then mining them at the same time.