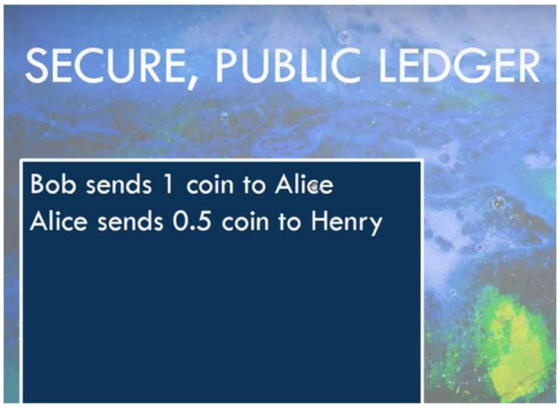
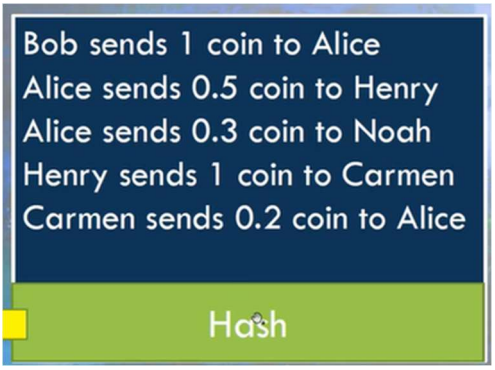
Vid 29 – Securing a Public Transaction Ledger

Ledger should be secured in 2 ways

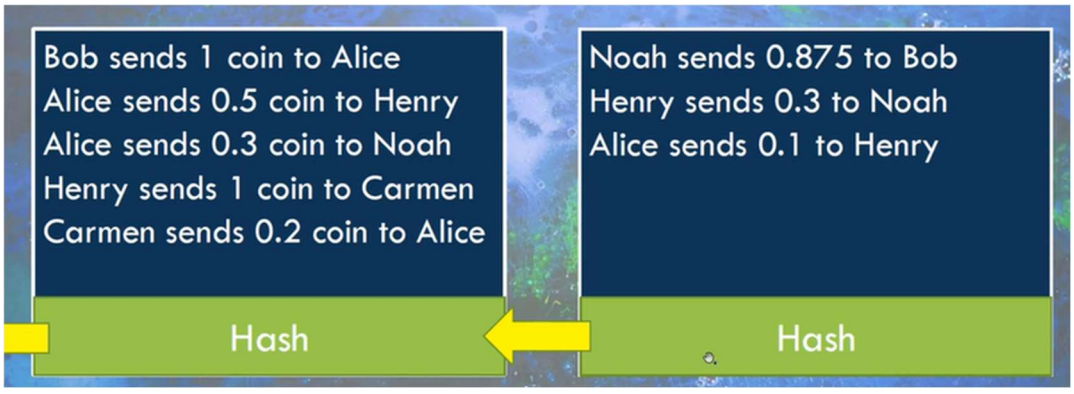
1. Anyone should be able to read this ledger and quickly verify the digital signatures on each of these transactions and in this way be reassured that all of these transactions were added to the ledger with the permission of the coin owner.
    a. When bob sends coins to Alice the only way for those coins to move out of Alice's account is for her to sign using her secret private key – a transaction sending them somewhere else.
2. Should also be tamper proof – once a transaction is committed to the ledger, it should be very hard for bob to go back and delete this transaction.



In a crypto blockchain transactions are collected into a block then the hash of the previous block is computed and added to the current block, then the entire block, including the hash of the previous block, is published to the network.



Each new block will include new transaction and the hash of the block before it making it impossible to change the contents of any block without changing the contents of every block following



Most crypto blockchains begin with the first block called the Genesis block. Both the contents of this block and its hash are typically stored in the software itself

In this section well create a genesis block for our coin and will declare it valid by def and create some coins in it out of thin air. Other blocks will not be allowed to do this until we add whats called a block reward in the next section.

Vid 30 – Review of Previous Work

Mostly about assembling the pieces that we've already created into a tamper-proof transaction ledger.

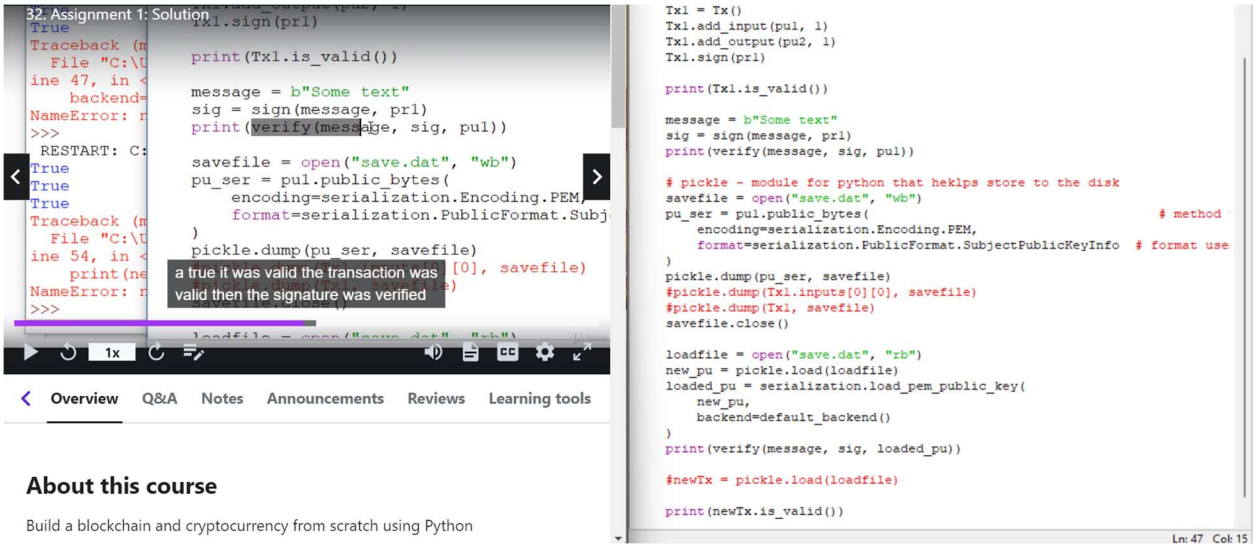
Key loading and serialization → how well be able to save and load our private key. Key serialization takes our private or public key and then takes your private and public key and creates some text out of it that can be then rolled back into a public and private key object using the functionality in key loading

https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/

passwords → should it ever require a password a public key is known (everybody should know it) but a private key is very different, and storing it on your hard drive is a very bad idea so the private key is usually serialized with a password

loading SSH public keys

https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#cryptography.hazmat.primitives.serialization.load_ssh_public_key



3 trues

- ➔ the transaction was valid,
- ➔ sig valid before the saving and loading
- ➔ sig valid after the verifying and loading

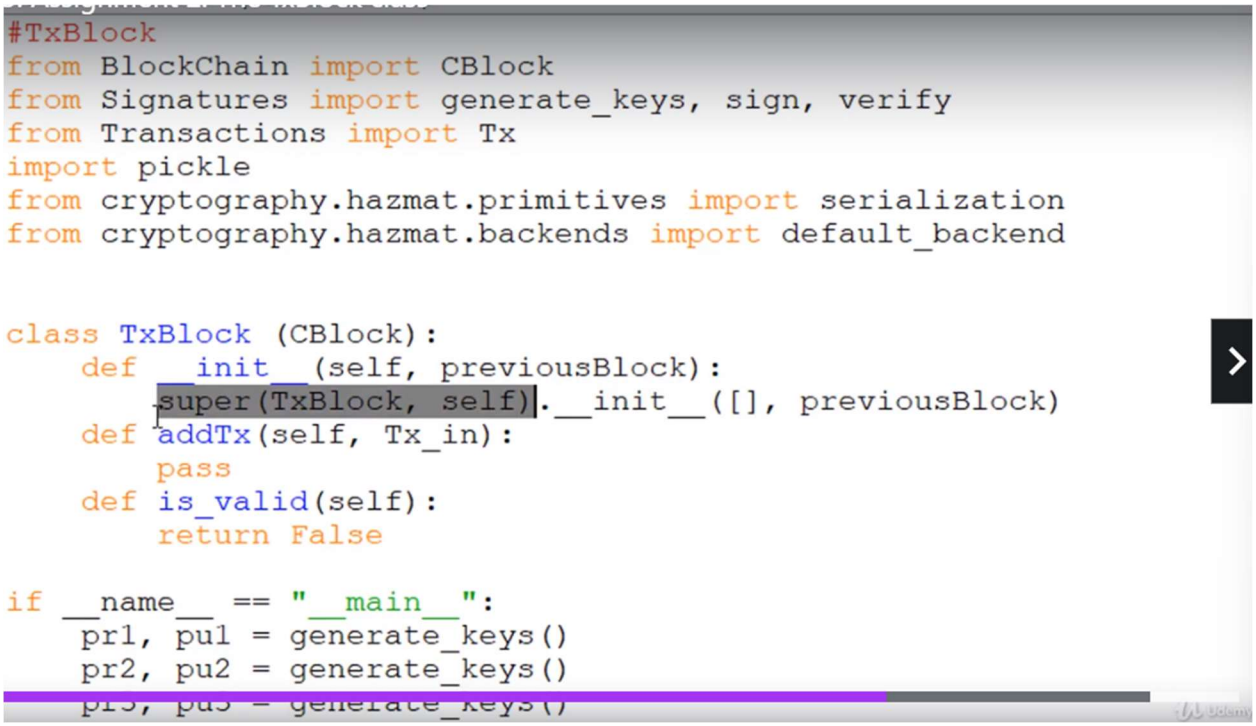
These steps can allow us to DUMP our private key, only problem is the Tx right now is storing the public key as a class instance not a serialized object that can be pickled

Going back to the signature class → instead of returning a public object which is the public key class. Serialize it as we pass it out of generate_keys and then load it as it is passed into verify

```
def is_valid(self):
    total_in = 0
    total_out = 0
    message = self.__gather()
    for addr,amount in self.inputs:                    # check that every input is correct (that every input is signed)
        found = False                                  # look thru list of sigs and see if there is a corresponding sig
        for s in self.sigs:                            # if look @ all sigs and dont find match - we dont ever set found to true
            if Signatures.verify(message, s, addr):    # if something then we'll set found to true
                found = True                           # if never reach this found = true statement then we havent found a match and will return false
        if not found:
            return False
        if amount < 0:
            return False
        total_in = total_in + amount
    for addr in self.reqd:
        found = False
        for s in self.sigs:
            if Signatures.verify(message, s, addr):
                found = True
        if not found:
            return False
    for addr,amount in self.outputs:
        if amount < 0:
            return False
        total_out = total_out + amount

    if total_out > total_in:
        return False

    if total_out > total_in:
        print("Outputs exceed inputs")
        return False

    return True

def __gather(self):
    data = []
    data.append(self.inputs)
    data.append(self.outputs)
    data.append(self.reqd)
```

Vid 33 – the TxBlock Class

10:20



```
#TxBlock
from BlockChain import CBlock
from Signatures import generate_keys, sign, verify
from Transactions import Tx
import pickle
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend


class TxBlock (CBlock):
    def __init__(self, previousBlock):
        super(TxBlock, self).__init__([], previousBlock)
    def addTx(self, Tx_in):
        pass
    def is_valid(self):
        return False

if __name__ == "__main__":
    pr1, pu1 = generate_keys()
    pr2, pu2 = generate_keys()
    pr3, pu3 = generate_keys()
```

Vid 33 – Assignment 2 Solution

used the super functionality to call the parent constructor for our particular instance of TXblock

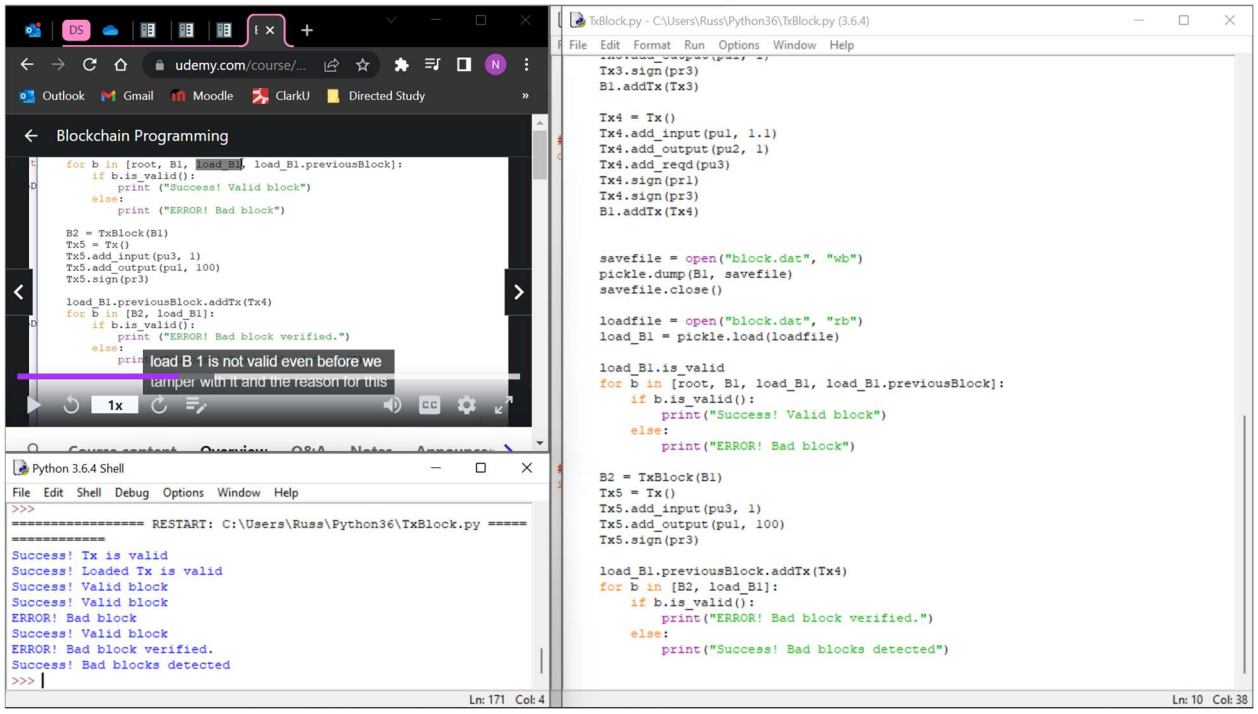
```
class TxBlock(CBlock):
    def __init__(self, previousBlock):
        super(TxBlock, self).__init__([],previousBlock) # used the super functionalit
    def addTx(self, Tx_in):
        pass
    def is_valid(self):
        return False
```

its going to run back and find this constructor in CBlock and its gonna make an empty list (data) and then look at the previousBlock, compute its hash and store that as self.previousHash – 00:41

we don't have to do it in 2 different places, our TxBlock will just do as CBlock is instructed to do with an empty list

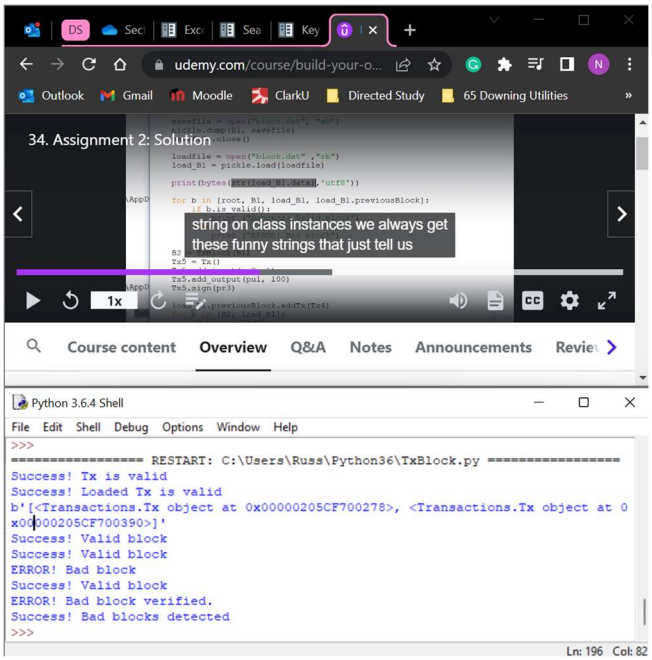
```
class CBlock:
    data = None
    previousHash = None
    previousBlock = None
    def __init__(self, data, previousBlock):
        self.data = data
        self.previousBlock = previousBlock
        if previousBlock != None:
            self.previousHash = previousBlock.computeHash()
```

Need to figure out why load_B1 is not valid even before the tampering



Vid 34 – Assignment 2 Solution

When we call this str on class instances we always get these funny strs that just tell us the name of the class and its location. Instead, wed like this str to be all the data inside the particular transaction.




```
def __gather(self):
    data = []
    data.append(self.inputs)
    data.append(self.outputs)
    data.append(self.reqd)
    return data
def __repr__(self): # gonna be called whenever we converit it to a st
    reprstr = "INPUTS:\n"
    for addr,amt in self.inputs:
        reprstr = reprstr + str(amt) + " from " + str(addr) + "\n"
    reprstr = reprstr + "OUTPUTS:\n"
    for addr,amt in self.outputs:
        reprstr = reprstr + str(amt) + " to " + str(addr) + "\n"
    reprstr = reprstr + "REQD:\n"
    for r in self.reqd:
        reprstr = reprstr + str(r) + "\n"
    reprstr = reprstr + "SIGS:\n"
    for s in self.sigs:
        reprstr = reprstr + str(s) + "\n"
    reprstr = reprstr + "END:\n"
    return reprstr
```

Right now we have a centralized tamper proof blockchain