Vid 22 – Transactions

Chapter 3 Objective → create a transaction class that represents the transfer of coins from one public address to another. Well sign the transaction and verify the signatures on those with the tools that we created In the signature module in chapter 1.

- Already created a **Signature module** that:
  - generate_keys
  - signs statements (creates a digital signature for statements)
  - verifies digital signatures

Magic of asymmetric encryption → only a person w/ the **private key** can digitally sign a statement and if someone tries to sign a statement without the correct **private key** its obvious to everyone that the signature is fake

signed by you
My public address

Public key → address, how others identify you in the blockchain world.

When coins have been sent to your public address the only way to get them out is to use your private key to sign the transaction sending them somewhere else.

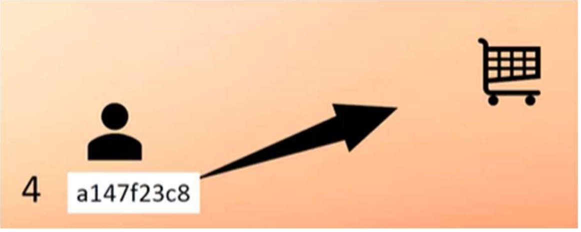
signed by me
My public address

When we say coins are sent to you → we mean that they're reserved in the public ledger in such a way that the only way to transfer them is to sign a transaction using your **private key.** As long as you keep track of the corresponding private key, you can have as many public addresses as desired

Vid 23 – Multiple output addresses

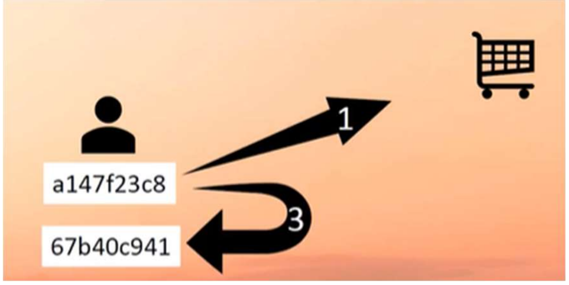
Our transaction class won't restrict us to transactions between just 2 addresses, it will store 2 lists → one for inputs and one for outputs.

For these transactions to be valid → they'll require a signature corresponding to each of the input addresses.

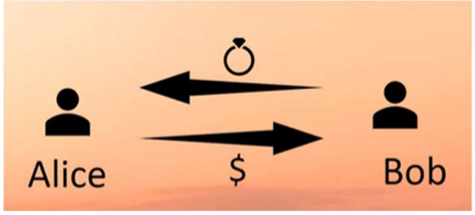


Reasons to allow several input and output addresses: Can provide better anonymity to use different addresses for each transaction.

If your address contains 4 coins and you want to buy something that costs only one coin, it might be smart to empty that address of all 4 coins by sending one coin to the seller and three to a brand-new address that you generate.

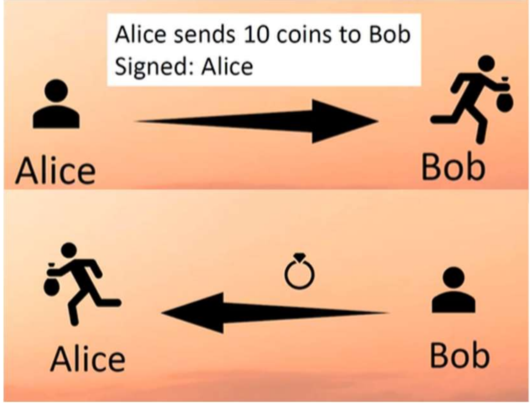


Someone who wants to track my spending has to ask themselves whether 1 coin was sent to the vendor and 3 back to me or 3 to the vendor and 1 back to me.



Vid 24 – Multi-signature transactions

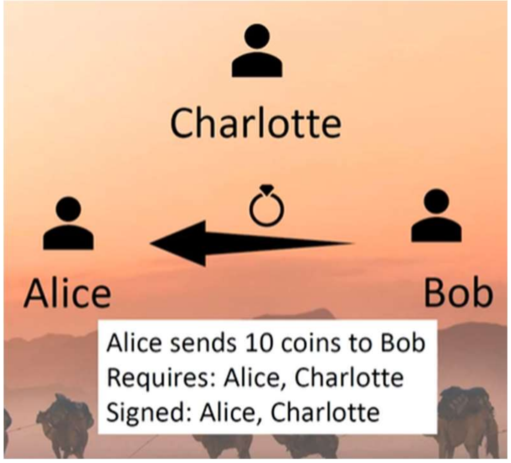
Every time coins come out of the address; we need a signature using the private key that corresponds to that address. But sometimes it can be useful to create a transaction that requires a signature from someone who doesn't have any coins.



Example → Alice is buying something expensive from a vendor, Bob, she doesn't know. They don't trust each other.

If Alice sends the crypto first Bob might disappear into the anonymity of the block chain and Alice will never get her goods.

On the other hand, if Bob sends goods without receiving payment, then Alice might never pay.



Charlotte doesn't have to be a close friend of either Bob or Alice. She's like a mutual arbiter. Alice will sign a transaction sending funds to Bob, but this will declare that its only valid with signatures from both Alice and Charlotte.

Bob can deliver the goods and then Charlotte signs the transaction releasing the funds to Bob. But if Bob fails to deliver → Charlotte wont sign the transaction and Bob wont get his coins

On the other hand, before Bob sends the goods its already too late for Alice to change her mind. She's already signed the transaction and Bob can see and verify that. Now, Alice's funds can be released by Charlotte's signature alone. So Bob is reassured enough to send the goods.

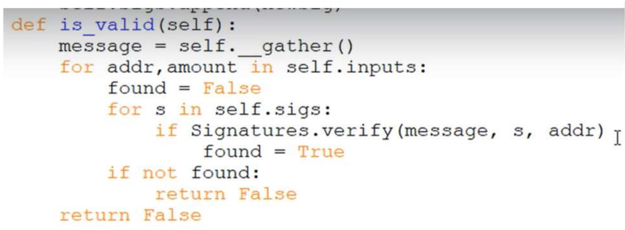
There's also no way for Charlotte to abscond with the coins. She can only sign the transactions sent by Alice and that transaction sends coins just to Bob not her. → **Escrow transaction**

Vid 27 – Solution Part A

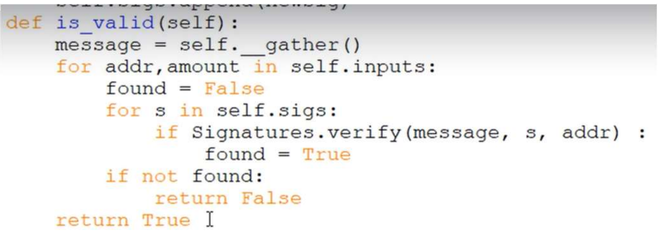
If we verify that sig for that massage with that address, if that's all true we found the sig

If not, if we go thru all the sigs for any particular address and we don't find it, that's false

But if we go thru all the input address and we found each on then we can return true at the end



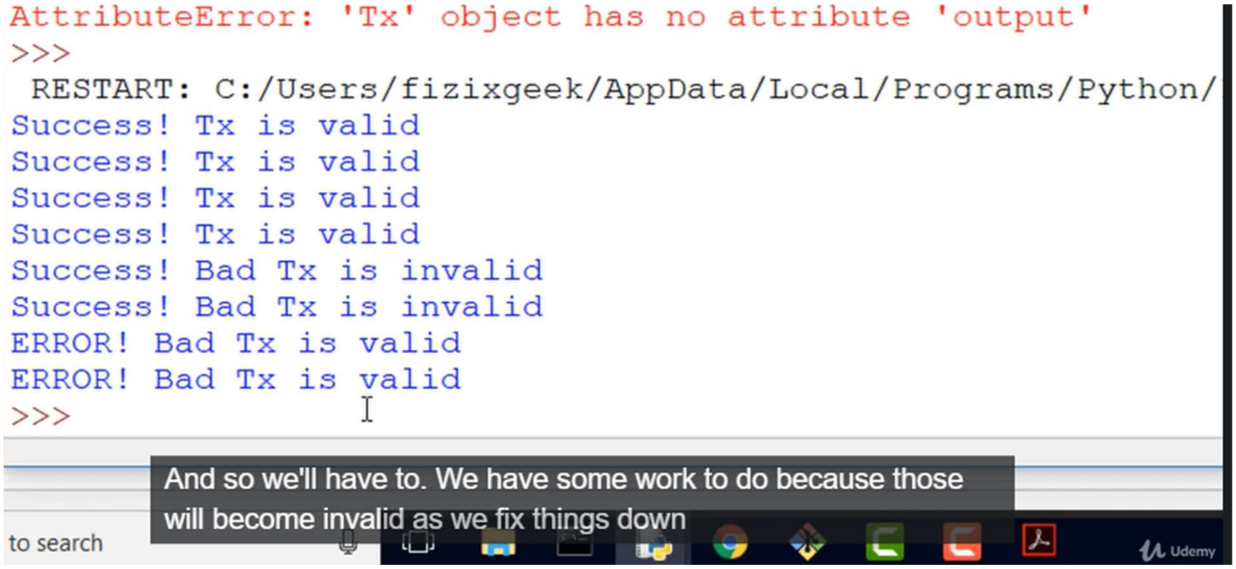
```
def is_valid(self):
    message = self.__gather()
    for addr,amount in self.inputs:
        found = False
        for s in self.sigs:
            if Signatures.verify(message, s, addr)
                found = True
        if not found:
            return False
    return False
```



```
def is_valid(self):
    message = self.__gather()
    for addr,amount in self.inputs:
        found = False
        for s in self.sigs:
            if Signatures.verify(message, s, addr) :
                found = True
        if not found:
            return False
    return True
```

For each address I'm going to go thru each sig, look for it, if its found then set found to true

Then when you get a new address, set found to false again

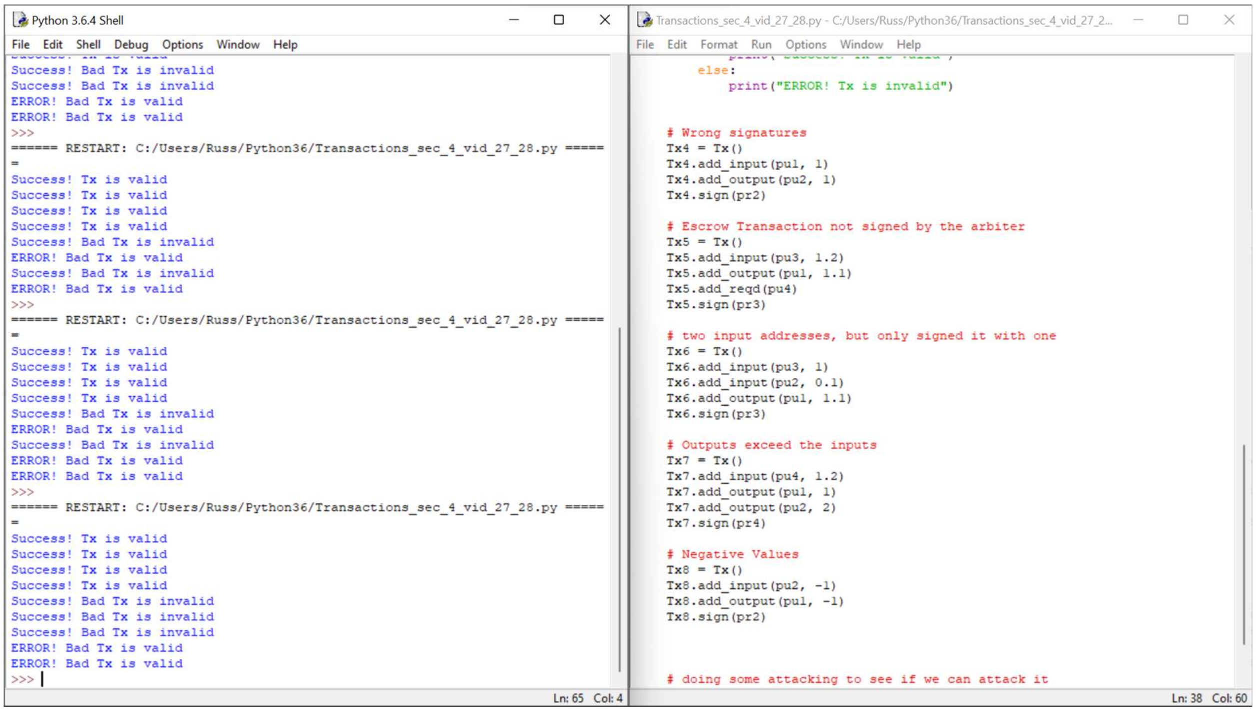


```
AttributeError: 'Tx' object has no attribute 'output'
>>>
 RESTART: C:/Users/fizixgeek/AppData/Local/Programs/Python/
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Bad Tx is invalid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
ERROR! Bad Tx is valid
>>>
```

And so we'll have to. We have some work to do because those will become invalid as we fix things down

Even the first 2 bad ones were detected, and then we failed on the others

Vid 28 – Solution 2


Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Ln: 65 Col: 4
Transactions_sec_4_vid_27_28.py - C:/Users/Russ/Python36/Transactions_sec_4_vid_27_2...
File Edit Format Run Options Window Help
Ln: 38 Col: 60

```
Success! Tx is valid
Success! Bad Tx is invalid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
ERROR! Bad Tx is valid
>>> 
====== RESTART: C:/Users/Russ/Python36/Transactions_sec_4_vid_27_28.py =====
=
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
>>> 
====== RESTART: C:/Users/Russ/Python36/Transactions_sec_4_vid_27_28.py =====
=
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
ERROR! Bad Tx is valid
>>> 
====== RESTART: C:/Users/Russ/Python36/Transactions_sec_4_vid_27_28.py =====
=
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Tx is valid
Success! Bad Tx is invalid
Success! Bad Tx is invalid
Success! Bad Tx is invalid
ERROR! Bad Tx is valid
ERROR! Bad Tx is valid
>>> |
```

```python
            print("Success! Tx is valid")
        else:
            print("ERROR! Tx is invalid")


    # Wrong signatures
    Tx4 = Tx()
    Tx4.add_input(pu1, 1)
    Tx4.add_output(pu2, 1)
    Tx4.sign(pr2)

    # Escrow Transaction not signed by the arbiter
    Tx5 = Tx()
    Tx5.add_input(pu3, 1.2)
    Tx5.add_output(pu1, 1.1)
    Tx5.add_reqd(pu4)
    Tx5.sign(pr3)

    # two input addresses, but only signed it with one
    Tx6 = Tx()
    Tx6.add_input(pu3, 1)
    Tx6.add_input(pu2, 0.1)
    Tx6.add_output(pu1, 1.1)
    Tx6.sign(pr3)

    # Outputs exceed the inputs
    Tx7 = Tx()
    Tx7.add_input(pu4, 1.2)
    Tx7.add_output(pu1, 1)
    Tx7.add_output(pu2, 2)
    Tx7.sign(pr4)

    # Negative Values
    Tx8 = Tx()
    Tx8.add_input(pu2, -1)
    Tx8.add_output(pu1, -1)
    Tx8.sign(pr2)



    # doing some attacking to see if we can attack it
```

Didn't detect outputs exceed inputs and negative values