Homework 1

In this HW, you will create bitcoin transactions.

Lets first install **python-bitcoinlib** which we will use to connect to the Bitcoin network and to generate a public-private key pair.

```
!pip install python-bitcoinlib
!pip install requests
```

```
Collecting python-bitcoinlib
  Downloading https://files.pythonhosted.org/packages/08/c8/07e5b77ca1904fb0127334d9f154b1cb363755c9b27a9c5e519514f5f9f3/python_bitcoinlib-0.11.0-py3-none-any.whl (103kB)
     |████████████████████████████████| 112kB 11.7MB/s
Installing collected packages: python-bitcoinlib
Successfully installed python-bitcoinlib-0.11.0
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (2.23.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests) (2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests) (2.10)
```

Lets create a Bitcoin address and accompanying private key for ourselves with the code below.

```
from os import urandom
from bitcoin import SelectParams
from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress

SelectParams('testnet')

seckey = CBitcoinSecret.from_secret_bytes(urandom(32))

print("Private key: %s" % seckey)
print("Address: %s" %P2PKHBitcoinAddress.from_pubkey(seckey.pub))
```

```
Private key: cPbAtRGnscXjJRxsSYGWhrcXByjUtmYaa4hUZqn1uwYumQJXnsW4
Address: mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85
```

Save this information to somewhere. Otherwise, it may dissappear if you accidentally erase it and get disconnected. You will not be able to access it later.

Now with this address, we can get free testnet coins from the coin faucet. Click here to get some free coins on the *Bitcoin test network*. Note that it is a nice thing to send back the bitcoin's you got to the faucet.

Once you click the button, a transaction, which is a P2PKH transaction, will be generated and sent to the Bitcoin Test network (by the faucet). One of the outputs of this transaction will be for your Public Key Hash. That output will be our main UTXO to spend.

Before continuing for the exercises, lets configure our Python environment with our public and private key pair. This is nothing fancy; we just set some variables' values to be used later in the codes and the exercises in the rest of this notebook.

Note that we will use the same bitcoin address, the one we created above, to correctly configure our environment.

```
from bitcoin import SelectParams
from bitcoin.base58 import decode
from bitcoin.wallet import CBitcoinAddress, CBitcoinSecret, P2PKHBitcoinAddress

#This says that we will connect to testnet, not the actual Bitcoin network
SelectParams('testnet')

#This is my private key. You need to add yours here
my_private_key = CBitcoinSecret('cPbAtRGnscXjJRxsSYGWhrcXByjUtmYaa4hUZqn1uwYumQJXnsW4')

#As you see, we do not explicitly see the public key  anywhere. We do not need to see it.
#The elliptic curve operations are handled in the background. Your private key
#is in fact a scalar which will be multiplied with a point on the curve to obtain
#the public key (note: elliptic curve discrete logarithm problem).
my_public_key = my_private_key.pub

#With P2PKHBitcoinAddress's from_pubkey function, we can obtain our address.
my_address = P2PKHBitcoinAddress.from_pubkey(my_public_key)

#This is the address of the faucet (a Bitcoin Test account) which we will
#send our money back during/after the exercises.
faucet_address = CBitcoinAddress('2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx')

print("Private key: %s" % my_private_key)
print("Address: %s" %P2PKHBitcoinAddress.from_pubkey(my_private_key.pub))
```

```
Private key: cPbAtRGnscXjJRxsSYGWhrcXByjUtmYaa4hUZqn1uwYumQJXnsW4
Address: mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85
```

The first thing we will do will be splitting the UTXO obtained from the faucet into multiple UTXOs. There are multiple exercises in this notebook, hence we need more than one UTXO to spend. Besides, we may do mistakes, spend an UTXO in an unlockable manner and lose some of our test coins. So it is a good exercise to split the only UTXO we have.

Lets run the **split_coins** function below with the transaction we obtained from the faucet. The code is given below: You do not need to change the **split_coins** function. However, you need to add the UTXO information we obtained from the faucet at the end.

You can check the status of the outputs in the original faucet transaction from BlockCypher (as I do it here). Before running the code below it must be *unspent*. Read the code and try to understand what it does by matching what we have learned in the class with the statements below.

If you do everything correctly, at the end, you need to get a response with code 201. That is, you will see something like

```
201 Created
{
  "tx": {
        ....
  }
}
```

at the beginning of the output. And after some time, depending on the fee you give to the miners, the UTXO you used will appear as spent. Fortunately, you have many more now to be used in the exercises.

```
import requests

from bitcoin.core import b2x, lx, COIN, COutPoint, CMutableTxOut, CMutableTxIn, CMutableTransaction, Hash160
from bitcoin.core.script import *
from bitcoin.core.scripteval import VerifyScript, SCRIPT_VERIFY_P2SH

def create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, seckey):
    #We want no parts of the transaction to be changed.
    #Hence, we hash all the transaction before signing it (i.e.: SIGHASH_ALL).
    sighash = SignatureHash(txin_scriptPubKey, tx, 0, SIGHASH_ALL)
    signature = seckey.sign(sighash) + bytes([SIGHASH_ALL])
    return signature

def broadcast_transaction(tx):
    raw_transaction = b2x(tx.serialize())
    headers = {'content-type': 'application/x-www-form-urlencoded'}
    return requests.post(
        'https://api.blockcypher.com/v1/btc/test3/txs/push',
        headers=headers,
        data='{"tx": "%s"}' % raw_transaction)

#What we simply do in this transaction is splitting. We spent the faucet output and
#split the amount into n pieces each is again to our addresses in a P2PKH structure.
def split_coins(amount_to_send, txid_to_spend, utxo_index, n):
    #We are creating a transaction with a single input and n outputs.

    #txin is the input part of the transaction
    txin_scriptPubKey = my_address.to_scriptPubKey() #initial UTXO was for my address so we need to use it as scriptPubKey
    txin = CMutableTxIn(COutPoint(lx(txid_to_spend), utxo_index)) #now we have txin
    #The function lx() onverts a little-endian hex string to bytes
    #The COutPoint is the combination of a transaction hash and an index n into its vout as we have discussed in the lecture

    #This transaction's outputs are all for myself - so the default P2PKH script
    #[OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG] must be the scriptPubKey
    #The function to_scriptPubKey automatically generates this script from a P2PKHBitcoinAddress
    #Note that the script contains a base16 encoded version of checksummed/modified byte58 encoded address (This operation is reversible!).
    txout_scriptPubKey = my_address.to_scriptPubKey()

    #txout is a single output. (IMP: the function CScript which you use for the first time serializes the script created above)
    txout = CMutableTxOut((amount_to_send / n) * COIN, CScript(txout_scriptPubKey)) #amount and address are given here

    #This is the whole transaction we will broadcast
    #Note the [txout]*n in the parameters of CMutableTransaction function
    tx = CMutableTransaction([txin], [txout] * n)

    #Now lets sign the transaction to keep its integrity
    sig = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, my_private_key)
    #We need to create the unlocking script which is [signature, encoded SIGHASH_ALL] + PublicKey
    txin.scriptSig = CScript([sig, my_public_key])

    #Lets validate if the script is correctly generated...
    #This function raises a ValidationError subclass if the validation fails
    VerifyScript(txin.scriptSig, txin_scriptPubKey, tx, 0, (SCRIPT_VERIFY_P2SH,))

    #Broadcast the transaction
    response = broadcast_transaction(tx)
    print(response.status_code, response.reason)
    print(response.text)

#The mount of BTC in the output you're splitting minus fee.
#Warning: In the test network, fees are required. They can be small but since this
#is an exercise, you can be generous
amount_to_send = 0.017

#This must be the transaction id you have from the faucet's transaction
txid_to_spend = ('513c549e7beef1281dd2f324bb1b47a4e9c7baa9800cb9fa50e88202a0b235e6')

#the output UTXO location in the transaction from faucet (mine is 1, but maybe yours is 0)
utxo_index = 0

#number of transactions to be splitted
n = 17
######################################################################

split_coins(amount_to_send, txid_to_spend, utxo_index, n)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
    "addresses": [
      "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
    ],
    "total": 1700000,
    "fees": 24486,
    "size": 736,
    "vsize": 736,
    "preference": "low",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:28:15.410259564Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 17,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "513c549e7beef1281dd2f324bb1b47a4e9c7baa9800cb9fa50e88202a0b235e6",
        "output_index": 0,
        "script": "483045022100c22ebb9db43480e00d9b3f179217e77256ed37a6254ccb9bf46992b4a3fc6a150220796b1e5e62b1f4ae74062a48210d42b959de209b840b4536b7b4996f91cc0c0e0121037d6
        "output_value": 1724486,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 100000,
        "script": "76a91470780e7e8234e1af3ba9004a5d7881d36855822888ac",
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash"
      },
      {
        "value": 100000,
        "script": "76a91470780e7e8234e1af3ba9004a5d7881d36855822888ac",
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash"
      },
      {
        "value": 100000,
        "script": "76a91470780e7e8234e1af3ba9004a5d7881d36855822888ac",
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash"
```

Part 1 - Exactly same with the given practice exercise

Here is the first part: it is about spending one of the smaller transactions and send it back to the **faucet**. Make it complete, run, and be sure that you have the correct output with code *201 Created*.

Hint: Almost all the information you need, the codes, functions to be used etc. can be read and copied from the **split_coins** example above. Before, we created a transaction to split the original UTXO obtained from the faucet. You will do the same here but this time you will not direct the output to yourself. The money needs to go back to the faucet.

```
from bitcoin.core.script import *

# TODO: Complete this script to unlock the BTC that was sent to you
# in the PayToPublicKeyHash transaction. You may need to use variables
# that are globally defined.
def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index, txout_scriptPubKey):
    txin = CMutableTxIn(COutPoint(lx(txid_to_spend), utxo_index))
    txout = CMutableTxOut(amount_to_send * COIN, CScript(txout_scriptPubKey))
    tx = CMutableTransaction([txin], [txout])

    txin_scriptPubKey = my_address.to_scriptPubKey()
    signature = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, my_private_key);
    txin.scriptSig = CScript([signature, my_public_key])

    VerifyScript(txin.scriptSig, txin_scriptPubKey, tx, 0, (SCRIPT_VERIFY_P2SH,))

    return broadcast_transaction(tx)

######################################################################
```

```
######################################################################
# TODO: set all these parameters correctly
amount_to_send = 0.0008 #Do not forget the fee
txid_to_spend = ('d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970')
utxo_index = 0
txout_scriptPubKey = faucet_address.to_scriptPubKey() #[OP_DUP ....]
######################################################################

response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index, txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "c8a29dac1247de0d9722f6e0d9cd18e7fde4c1d3101038f12545132e4fa24e1f",
    "addresses": [
      "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85",
      "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
    ],
    "total": 80000,
    "fees": 20000,
    "size": 189,
    "vsize": 189,
    "preference": "high",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:29:17.042396081Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 0,
        "script": "47304402200d4c7102a2bfb321f516383aab5e1929785374f20080cd52f70bd82d3d7ca223022045e22576c313b2a47ac01a9f99adab876fc223be83027eacc3c4cb102df8513d0121037d65c
        "output_value": 100000,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 80000,
        "script": "a91487d3e275113bbff7046cff3688ad40a0ac70fcc587",
        "addresses": [
          "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
        ],
        "script_type": "pay-to-script-hash"
      }
    ]
  }
}
```

Part 2 - The puzzle is different than the one in the exercise

In this part, you will create a UTXO and later redeem it by using P2SH approach. The locking script will contain the hash of a puzzle whose solution will be given in the redeem script.

To unlock, the locking script will require three numbers whose sum is 15 and maximum is smaller than 8. Your script must check these equalities programatically.

For the opcodes you may need to use please check this webpage. Again if you are successful, you will get the return code **201** after the transaction broadcast.

You will complete this exercise in two parts. First a transaction will be created with a single output. Then it will be redeemed (by youself, to be sent to the faucet). The starting code for the first part is given below:

```
from sys import exit
from bitcoin.core.script import *

######################################################################
# TODO: Complete the scriptPubKey implementation with operations
# Be careful while using OP_EQUAL and OP_VERIFY - read them from https://en.bitcoin.it/wiki/Script
ex2a_txout_scriptPubKey = [OP_3DUP, OP_8, OP_LESSTHAN, OP_VERIFY, OP_8, OP_LESSTHAN, OP_VERIFY, OP_8, OP_LESSTHAN, OP_VERIFY, OP_ADD, OP_ADD, OP_15, OP_EQUAL]
######################################################################

######################################################################
# TODO: set these parameters correctly
amount_to_send = 0.0008
txid_to_spend = ('d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970')
utxo_index = 1
######################################################################

#This function is from the first exercise
response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index, ex2a_txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "66f7527072e5832e3bdb2cdc8fcc2f93e576178bb2e5d7b514341517dc23ddaa",
    "addresses": [
      "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
    ],
    "total": 80000,
    "fees": 20000,
    "size": 180,
    "vsize": 180,
    "preference": "high",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:31:38.564725643Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 1,
        "script": "47304402200f58b15591babbafad1d0ebb42f73dbbf20ede3a69f2412692fe7c2fa0ef4f940220201bbd5939e49d207256fbf755163045157123a40953d370749a82f24e95ce270121037d65c
        "output_value": 100000,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 80000,
        "script": "6f589f69589f69589f6993935f87",
        "addresses": null,
        "script_type": "unknown"
      }
    ]
  }
}
```

In the second part, we will redeem by only providing the solution of the locking script. Note that anyone can do this. Since the lock script does not contain any signature, address etc. check, one only need to provide **a**, **b** and **c**. You can follow the results, e.g., confirmations, from BlockCypher test network explorer by using the transaction hash you obtained after succesfully completing this part.

```
def send_from_custom_transaction(amount_to_send, txid_to_spend, utxo_index, txin_scriptPubKey, txin_scriptSig, txout_scriptPubKey):
    txout = CMutableTxOut(amount_to_send * COIN, CScript(txout_scriptPubKey))
    txin = CMutableTxIn(COutPoint(lx(txid_to_spend), utxo_index))
    tx = CMutableTransaction([txin], [txout])

    txin.scriptSig = CScript(txin_scriptSig)
    VerifyScript(txin.scriptSig, CScript(txin_scriptPubKey), tx, 0, (SCRIPT_VERIFY_P2SH,))

    return broadcast_transaction(tx)

######################################################################
# TODO: set these parameters correctly
amount_to_send = 0.0005
txid_to_spend = '66f7527072e5832e3bdb2cdc8fcc2f93e576178bb2e5d7b514341517dc23ddaa'
utxo_index = 0
######################################################################

txin_scriptPubKey = ex2a_txout_scriptPubKey
######################################################################
# TODO: implement the scriptSig for redeeming the transaction created in  Exercise 2a.
# This must be a very chort scriptSig. No signature, address check etc.
# Anyone who can submit the solution of the problem (a and b in correct form) can spend the transaction
txin_scriptSig = [OP_7,OP_7,OP_1]
######################################################################
```

```
txout_scriptPubKey = faucet_address.to_scriptPubKey() #sending money to faucet after solving the puzzle

response = send_from_custom_transaction(amount_to_send, txid_to_spend, utxo_index, txin_scriptPubKey, txin_scriptSig, txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
```

```
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "373c123864f8443201e96dec139a826ef278ff73d9554ffe9c8367023b85053e",
    "addresses": [
      "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
    ],
    "total": 50000,
    "fees": 30000,
    "size": 86,
    "vsize": 86,
    "preference": "high",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:32:44.465246432Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "66f7527072e5832e3bdb2cdc8fcc2f93e576178bb2e5d7b514341517dc23ddaa",
        "output_index": 0,
        "script": "575751",
        "output_value": 80000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 50000,
        "script": "a91487d3e275113bbff7046cff3688ad40a0ac70fcc587",
        "addresses": [
          "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
        ],
        "script_type": "pay-to-script-hash"
      }
    ]
  }
}
```

Part 3: Slightly different than the practice

In the first part of the third exercise, you will create and broadcast a MULTISIG transaction to the test network. The second part will be about providing an unlock script for this transaction. You will definitely want to read how OP_CHECKMULTISIG works from here.

The scenario is as follows: in a company, there is a manager and three employees. The multisig signature you will create (in the first part) can be redeemed by the manager combined with one of the employees. However, only the manager, or only the employees with enough number of signatures will not be able to redeem the transaction. **Unlike the HW, the order of the signatures should not matter**.

You may assume the role of the manager for this exercise so that the manager's private key is your private key and the manager's public key is your public key.

For the employee private keys and addresses, you can use the key generation code at the beginning of this notebook. That is generate 3 random Bitcoin secret keys to be used in this exercise with the code below

```
from os import urandom
from bitcoin import SelectParams
from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress

SelectParams('testnet')

seckey = CBitcoinSecret.from_secret_bytes(urandom(32))

print("Private key: %s" % seckey)
print("Address: %s" %P2PKHBitcoinAddress.from_pubkey(seckey.pub))
```

After filling the spaces ---xxx-- in the first part below, we can create the transaction locked as described above.

```
from sys import exit
from bitcoin.core.script import *

#Run the key generation code at the beginning of this notebook multiple times
#to create private keys and addresses for the employees.
employee1_private_key = CBitcoinSecret('cQhtfzciQqEnHmASa1uzc1bf1ntydPE2Rx4zo9h3qmsbkCi3nmSR')
employee1_public_key = employee1_private_key.pub
employee2_private_key = CBitcoinSecret('cSfArKAns62aYGAWi75eT9h6ewDjvDRg4EdYpk4BaMahd5UutJFs')
employee2_public_key = employee2_private_key.pub
employee3_private_key = CBitcoinSecret('cUsLnLBcj9He2tdiknoUfWXbi7zT7W2dv5aAEFYmRFtRsSARb8je')
employee3_public_key = employee3_private_key.pub

######################################################################
# TODO: Complete the scriptPubKey implementation for Exercise 3
# You can assume the role of the manager for the purposes of this problem
# and use my_public_key and my_private_key in lieu of manager_public_key and
# manager_private_key.

ex3a_txout_scriptPubKey = [OP_2DUP, my_public_key, OP_CHECKSIG, OP_IF, OP_DROP, OP_ELSE, my_public_key, OP_CHECKSIGVERIFY, OP_SWAP, OP_ENDIF, OP_DROP,
                           OP_1, employee1_public_key, employee2_public_key, employee3_public_key, OP_3, OP_CHECKMULTISIG]
######################################################################

######################################################################
# TODO: set these parameters correctly
amount_to_send = 0.0008
txid_to_spend = ('d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970')
utxo_index = 2
######################################################################

response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index, ex3a_txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "635f486e455ec28f08d3dc82c891f62a329964ccf72b66ee25bab27ffb7bca8c",
    "addresses": [
      "zX6137ny3k8PJLGH8C37XEJNnXFieXdVV6",
      "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
    ],
    "total": 80000,
    "fees": 20000,
    "size": 348,
    "vsize": 348,
    "preference": "low",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:33:40.41676608Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 2,
        "script": "473044022025d15af0c21cf8421c68b6d04a315f38a85702caa9549227cd47ea3e025f4c21022079ab4cb45337c5650c0e2306caab6cec02a83f9704a8c619856c529b72cd36410121037d65c
        "output_value": 100000,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 80000,
        "script": "6e21037d65c6c4024a3e38d1dc0be2ff211b51d31b0c2062ad3a97fb19bdd49d2dc83eac63756721037d65c6c4024a3e38d1dc0be2ff211b51d31b0c2062ad3a97fb19bdd49d2dc83ead7c687
        "addresses": [
          "zX6137ny3k8PJLGH8C37XEJNnXFieXdVV6"
        ],
        "script_type": "pay-to-multi-pubkey-hash"
      }
    ]
  }
}
```

In the second part, you will redeem the transaction. Keep scriptPubKey as small as you can. One can use any legal combination of signatures to redeem the transaction but make sure that all combinations would have worked.

```
from sys import exit
from bitcoin.core.script import *

#employee ID is 1,2 or 3
```

```
def multisig_scriptSig(tx, txin_scriptPubKey, employee_ID):
    manager_sig = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, my_private_key)
    if employee_ID == 1:
      employee_sig = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, employee1_private_key)
    elif employee_ID == 2:
      employee_sig = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, employee2_private_key)
    else:
      employee_sig = create_OP_CHECKSIG_signature(tx, txin_scriptPubKey, employee3_private_key)
    ######################################################################
    # TODO: Complete this script to unlock the BTC that was locked in the multisig transaction created in Exercise 3a.
    return [OP_0, employee_sig, manager_sig]
    ######################################################################

def send_from_multisig_transaction(amount_to_send, txid_to_spend, utxo_index, txin_scriptPubKey, txout_scriptPubKey):
    txin = CMutableTxIn(COutPoint(lx(txid_to_spend), utxo_index))
    txout = CMutableTxOut(amount_to_send * COIN, CScript(txout_scriptPubKey))
    tx = CMutableTransaction([txin], [txout])

    txin.scriptSig = CScript(multisig_scriptSig(tx, CScript(txin_scriptPubKey), 2))
    VerifyScript(txin.scriptSig, CScript(txin_scriptPubKey), tx, 0, (SCRIPT_VERIFY_P2SH,))

    return broadcast_transaction(tx)

######################################################################
# TODO: set these parameters correctly
amount_to_send = 0.0005
txid_to_spend = ('635f486e455ec28f08d3dc82c891f62a329964ccf72b66ee25bab27ffb7bca8c')
utxo_index = 0
######################################################################

txin_scriptPubKey = ex3a_txout_scriptPubKey
txout_scriptPubKey = faucet_address.to_scriptPubKey()

response = send_from_multisig_transaction(amount_to_send, txid_to_spend, utxo_index, txin_scriptPubKey, txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "3da9e3e26e02f7d8f9f2204b97a05cce8b5cdd225ef00e3b699d02f1fa358f04",
    "addresses": [
      "zX6137ny3k8PJLGH8C37XEJNnXFieXdVV6",
      "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
    ],
    "total": 50000,
    "fees": 30000,
    "size": 230,
    "vsize": 230,
    "preference": "high",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:33:59.780525664Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "635f486e455ec28f08d3dc82c891f62a329964ccf72b66ee25bab27ffb7bca8c",
        "output_index": 0,
        "script": "0048304502210095480721330f9546f1a09cae2c64ec07e0e5cc70ed5b672abb3352fa8d6c9abc02203e90891a86e011a22b4b8698d6a49b3b4b56683a688461b87bd7795265aec55a0148304
        "output_value": 80000,
        "sequence": 4294967295,
        "addresses": [
          "zX6137ny3k8PJLGH8C37XEJNnXFieXdVV6"
        ],
        "script_type": "pay-to-multi-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 50000,
        "script": "a91487d3e275113bbff7046cff3688ad40a0ac70fcc587",
        "addresses": [
          "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
        ],
        "script_type": "pay-to-script-hash"
      }
    ]
  }
}
```

Part 4: This is the same with the practice

Redirect all remaining UTXOs to faucet

```python
from bitcoin.core.script import *

def send_multiple_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_indexes, txout_scriptPubKey):
    txins = [];
    for utxo_index in utxo_indexes:
      txin = CMutableTxIn(COutPoint(lx(txid_to_spend), utxo_index))
      txins.append(txin)

    txout = CMutableTxOut(amount_to_send * COIN, CScript(txout_scriptPubKey))
    tx = CMutableTransaction(txins, [txout])

    txin_scriptPubKey = my_address.to_scriptPubKey()
    for input_index, utxo_index in enumerate(utxo_indexes):
      sighash = SignatureHash(txin_scriptPubKey, tx, input_index, SIGHASH_ALL)
      signature = seckey.sign(sighash) + bytes([SIGHASH_ALL])
      txins[input_index].scriptSig = CScript([signature, my_public_key])
      VerifyScript(txins[input_index].scriptSig, txin_scriptPubKey, tx, input_index, (SCRIPT_VERIFY_P2SH,))
    return broadcast_transaction(tx)

######################################################################
# TODO: set all these parameters correctly
amount_to_send = 0.01 #Do not forget the fee
txid_to_spend = ('d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970')
```

```
utxo_indexes = [3,4,5,6,7,8,9,10,11,12,13,14,15,16]
txout_scriptPubKey = faucet_address.to_scriptPubKey() #[OP_DUP ....]
######################################################################

response = send_multiple_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_indexes, txout_scriptPubKey)
print(response.status_code, response.reason)
print(response.text)
```

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "c3585999adc86d0a7fa0d1daef61b73858e7391d0dbfd0ae97adb57fc99d27d6",
    "addresses": [
      "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85",
      "2N5dR3BTQ9FNXVetgqrdAXUD576NrS6x3xx"
    ],
    "total": 1000000,
    "fees": 400000,
    "size": 2109,
    "vsize": 2109,
    "preference": "high",
    "relayed_by": "34.91.190.75",
    "received": "2021-03-28T15:52:41.812809272Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 14,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 3,
        "script": "483045022100854e28e44d2bee854c9589bcf149e6d24e9e680e8c7683e7742e88f063e46bdb022000fde066ff2e91677fa081ddc416c0b4d81739a23ecf3d8c75ac167ae1405b8f0121037d6
        "output_value": 100000,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 1968145
      },
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 4,
        "script": "47304402203769 4baefb08650f02af70c62164e11d2ea832022009466058a77143aec5c9d8022055c2c42f175b71294d70f3c47096c76f2e7d0c50dae66d8ce8e594f9d50263ff0121037d65c
        "output_value": 100000,
        "sequence": 4294967295,
        "addresses": [
          "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 1968145
      },
      {
        "prev_hash": "d52a1428d6fb0e862430cd6a13e75c878b04a9c896434df869f9d353d110a970",
        "output_index": 5,
        "script": "47304402204d211a1cb0b3f2db70f812beb87cc8c14fefcd25240c044c6de3e5e9dae9748802207974ed67dfcc5bdee0be99adbd29e61dfd219ea904327bf2beaa041fea957db00121037d65c
        "output_value": 100000,
          "sequence": 4294967295,
          "addresses": [
            "mqmdoDfEDCroYRB6hDbcwKC8qC7wt7Ns85"
          ],
          "script_type": "pay-to-pubkey-hash",
          "age": 1968145
        },
```

✓ 0s completed at 6:52 PM