National Sun Yat-sen University
Introduction To Blockchain Technology
Homework 4
Course Number: CSE222, Chapter: 7 & 8

# Notice :
1. No late homework.
2. Please submit your homework to **Cyber University of National Sun Yat-sen University** (https://cu.nsysu.edu.tw/mooc/index.php). It is not allowed to submit assignments to any other location.
3. You will need to submit **Homework 4.docx**, after you paste all screenshot of your code and code execution results of solving the following problems.
4. We only accept using **python** to write program files.
5. **Please answer each question according to the requirements below, otherwise no points will be awarded.**

1. Create your own testnet transaction, get some coins for yourself and send them back

- Hint:
    - ○ Create your own testnet key-pair (private key + address).
        - ■ Please use a phrase other than 'Jimmy Song secret' and 'nsysu bitcoin secret'
    - ○ Obtain testnet coins from a faucet. (Search “bitcoin testnet faucet” or use a faucet listed on the Bitcoin wiki.)
    - ○ Build the transaction that sends these coins back:
        - ■ This should be a one-input, two-output (e.g., payment + change).
        - ■ Sign the input with the correct private key.

    - Broadcast the transaction at https://blockstream.info/testnet/tx/push

# Grading for Problem 1:

- Total: 40%
    - The TXID: 10%
    - A direct Blockstream Explorer URL or screenshot showing the transaction: 10%
    - Correct code and code execution result: 10%
    - A short description of your steps you used to construct and sign it: 10%
      (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 1 below. ---

註 **1:** 由於 **Blockstream Explorer** 在進行作業時，進行搜尋都會出現錯誤，因此採用 **mempool.space** 所提供的 **testnet bitcoin explorer(https://mempool.space/testnet4)** 以及 **trezor bitcoin testnet4 explorer (https://blockbook.tbtc-1.zelcore.io/)**來進行
註 **2:** 本次操作使用的都是 **testnet4**
註 **3:** 由於進行 **fetch** 時，也會出現錯誤，因此 **TxFetcher class** 並沒有做到從 **explorer** 中獲取 **raw transaction** 的效果，獲取 **raw transaction** 的方式是改為到 **trezor bitcoin testnet4 explorer** 搜尋該筆交易，直接使用網站所提供的該交易的 **raw transaction**。因此 **TxFetcher class** 也進行了部分修改**(**如下方附上的程式碼所示**)**

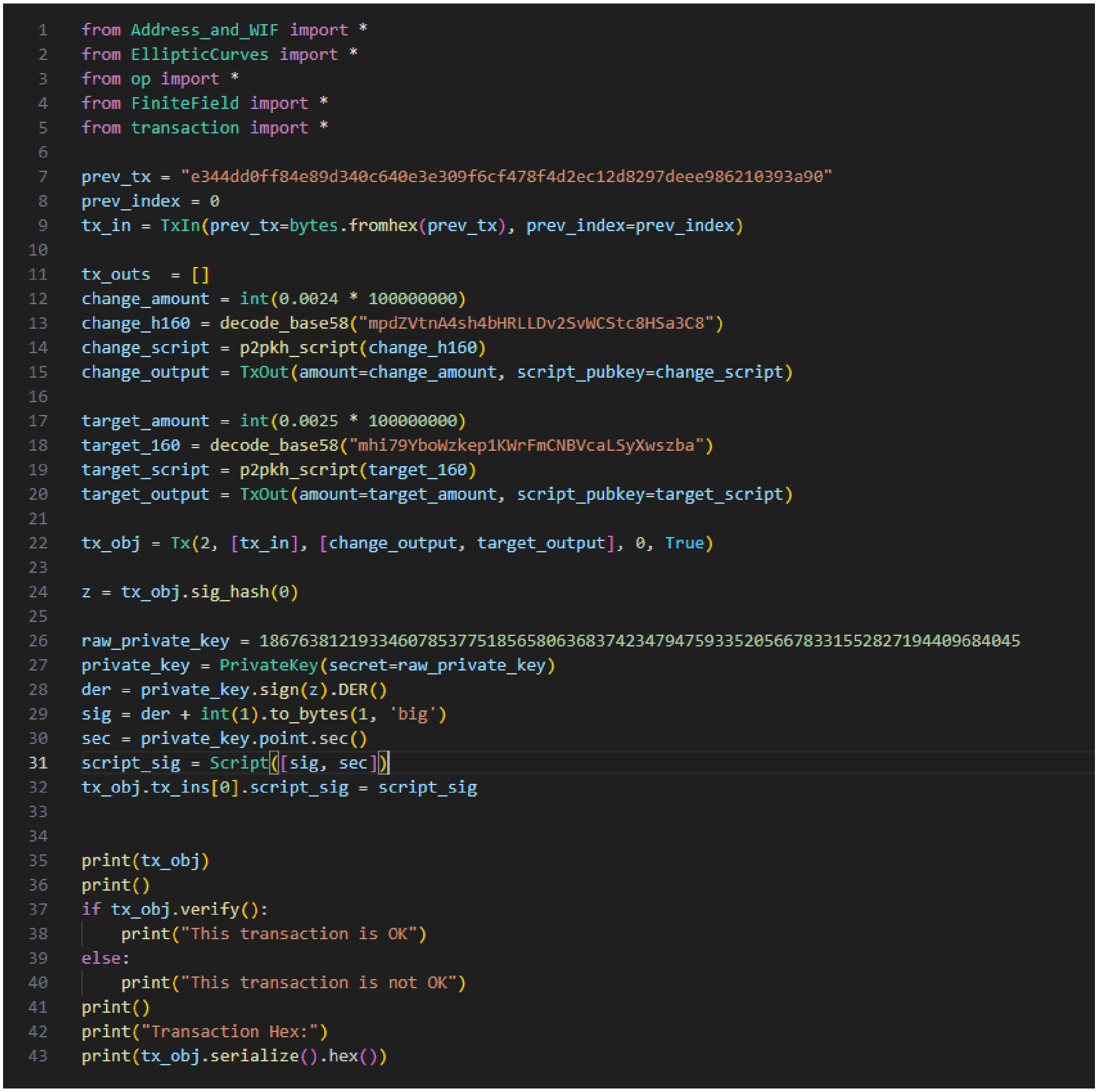
TXID:

- 從 **faucet** 獲取 **testnet bitcoin** 的 **TXID:**
  e344dd0ff84e89d340c640e3e309f6cf478f4d2ec12d8297deee986210393a90
  (https://blockbook.tbtc-1.zelcore.io/tx/e344dd0ff84e89d340c640e3e309f6cf478f4d2ec12d8297deee986210393a90)
- 將其由 **python code** 產生，將獲取的 **testnet bitcoin** 送回去的 **1-input, 2-output transaction** 並將其廣播至 **testnet** 的 **TXID:**
  4237c484e7cd362c4a9a2fe24758cc02be9ed8698b4948896a079e556e6a7e62
  (https://mempool.space/testnet4/tx/4237c484e7cd362c4a9a2fe24758cc02be9ed8698b4948896a079e556e6a7e62)
- 程式及執行結果**:(**沒有放上的程式就是與作業 **3** 的相同**)**

Main:

前面先建立一個 **1-input, 2-output** 的 **transaction**，在該 **input** 以 **private key** 進行 **signing**，最後以 **Tx.verify()**驗證是否創造新的比特畢，以及 **signature** 是否能 **unlock** 指定的 **transaction** 的 **output**。

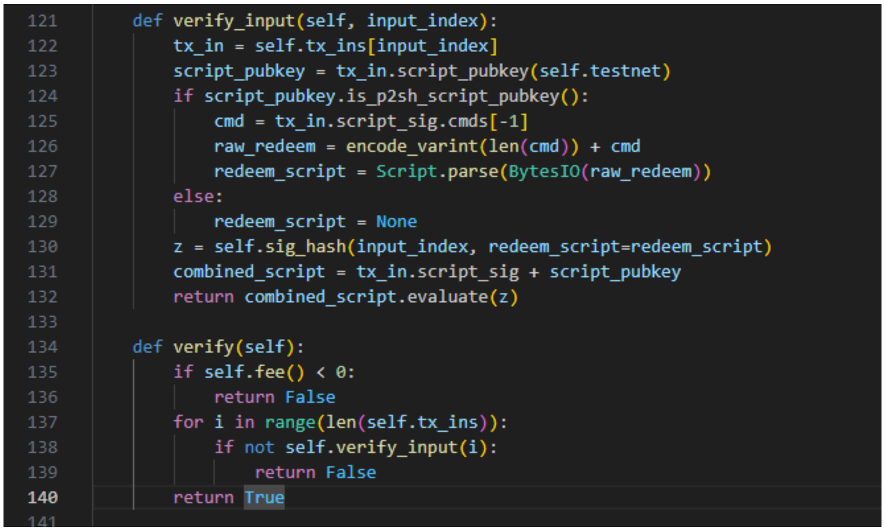
最後輸出該交易的資訊以及 **raw transaction hex**



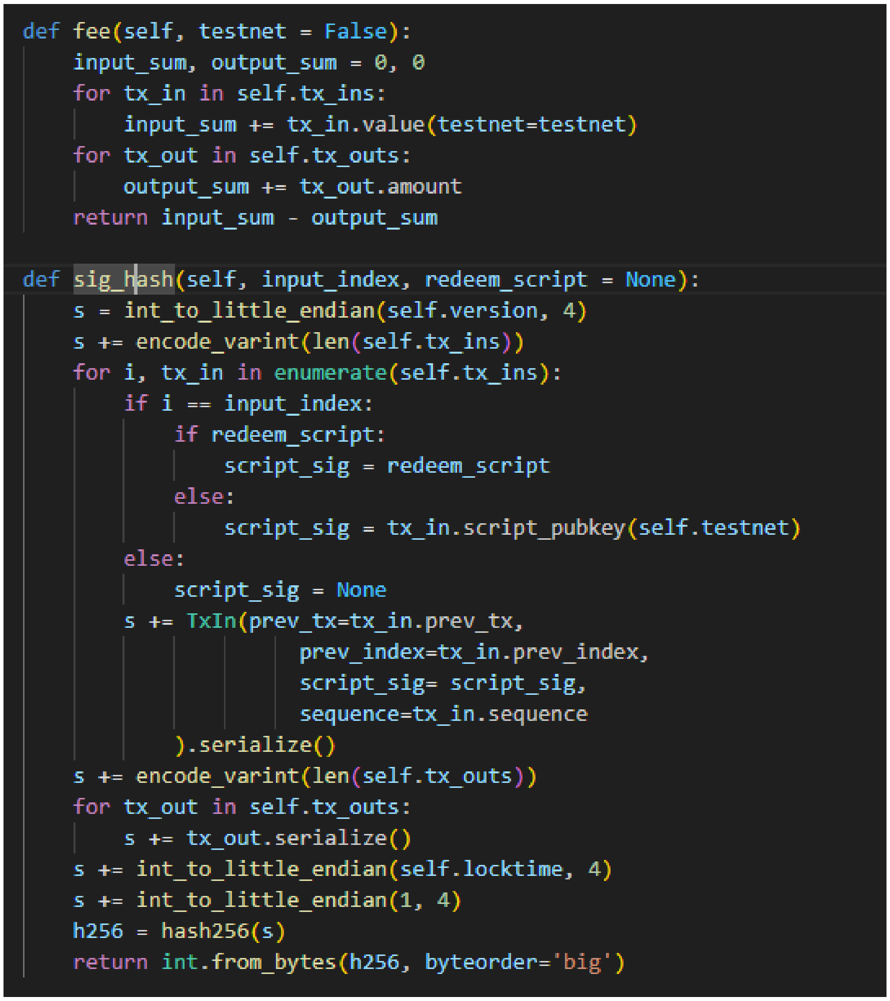
```python
from Address_and_WIF import *
from EllipticCurves import *
from op import *
from FiniteField import *
from transaction import *

prev_tx = "e344dd0ff84e89d340c640e3e309f6cf478f4d2ec12d8297deee986210393a90"
prev_index = 0
tx_in = TxIn(prev_tx=bytes.fromhex(prev_tx), prev_index=prev_index)

tx_outs  = []
change_amount = int(0.0024 * 100000000)
change_h160 = decode_base58("mpdZVtnA4sh4bHRLLDv2SvWCStc8HSa3C8")
change_script = p2pkh_script(change_h160)
change_output = TxOut(amount=change_amount, script_pubkey=change_script)

target_amount = int(0.0025 * 100000000)
target_160 = decode_base58("mhi79YboWzkep1KWrFmCNBVcaLSyXwszba")
target_script = p2pkh_script(target_160)
target_output = TxOut(amount=target_amount, script_pubkey=target_script)

tx_obj = Tx(2, [tx_in], [change_output, target_output], 0, True)

z = tx_obj.sig_hash(0)

raw_private_key = 18676381219334607853775185658063683742347947593352056678331552827194409684045
private_key = PrivateKey(secret=raw_private_key)
der = private_key.sign(z).DER()
sig = der + int(1).to_bytes(1, 'big')
sec = private_key.point.sec()
script_sig = Script([sig, sec])
tx_obj.tx_ins[0].script_sig = script_sig


print(tx_obj)
print()
if tx_obj.verify():
    print("This transaction is OK")
else:
    print("This transaction is not OK")
print()
print("Transaction Hex:")
print(tx_obj.serialize().hex())
```

Transaction.py:

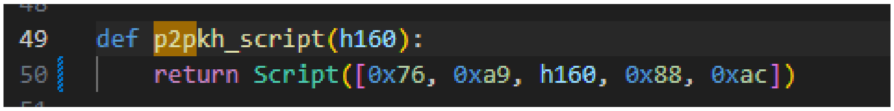
- **Class Tx 中的 verify 及 verify_input、fee 及 sig_hash、p2pkh_script**



```
    def verify_input(self, input_index):
        tx_in = self.tx_ins[input_index]
        script_pubkey = tx_in.script_pubkey(self.testnet)
        if script_pubkey.is_p2sh_script_pubkey():
            cmd = tx_in.script_sig.cmds[-1]
            raw_redeem = encode_varint(len(cmd)) + cmd
            redeem_script = Script.parse(BytesIO(raw_redeem))
        else:
            redeem_script = None
        z = self.sig_hash(input_index, redeem_script=redeem_script)
        combined_script = tx_in.script_sig + script_pubkey
        return combined_script.evaluate(z)

    def verify(self):
        if self.fee() < 0:
            return False
        for i in range(len(self.tx_ins)):
            if not self.verify_input(i):
                return False
        return True
```

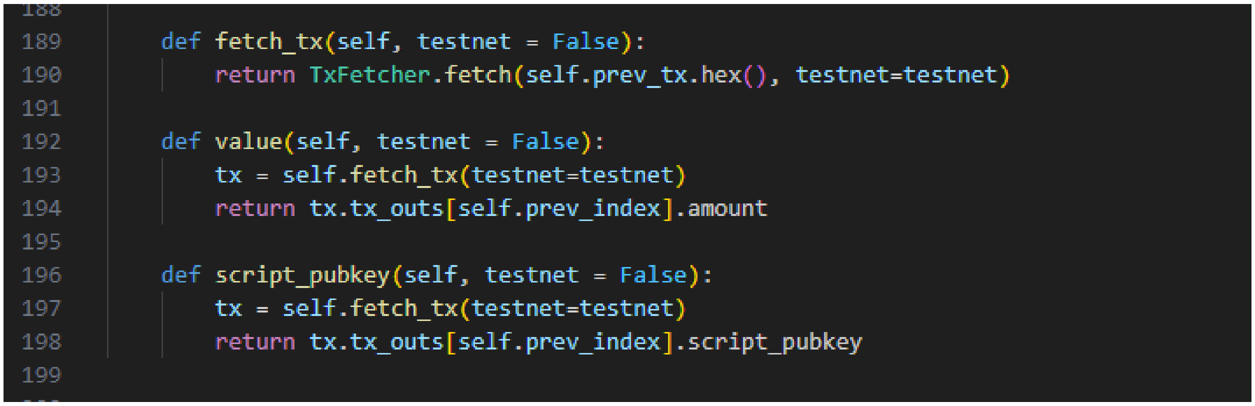


```
def fee(self, testnet = False):
    input_sum, output_sum = 0, 0
    for tx_in in self.tx_ins:
        input_sum += tx_in.value(testnet=testnet)
    for tx_out in self.tx_outs:
        output_sum += tx_out.amount
    return input_sum - output_sum

def sig_hash(self, input_index, redeem_script = None):
    s = int_to_little_endian(self.version, 4)
    s += encode_varint(len(self.tx_ins))
    for i, tx_in in enumerate(self.tx_ins):
        if i == input_index:
            if redeem_script:
                script_sig = redeem_script
            else:
                script_sig = tx_in.script_pubkey(self.testnet)
        else:
            script_sig = None
        s += TxIn(prev_tx=tx_in.prev_tx,
                      prev_index=tx_in.prev_index,
                      script_sig= script_sig,
                      sequence=tx_in.sequence
            ).serialize()
    s += encode_varint(len(self.tx_outs))
    for tx_out in self.tx_outs:
        s += tx_out.serialize()
    s += int_to_little_endian(self.locktime, 4)
    s += int_to_little_endian(1, 4)
    h256 = hash256(s)
    return int.from_bytes(h256, byteorder='big')
```



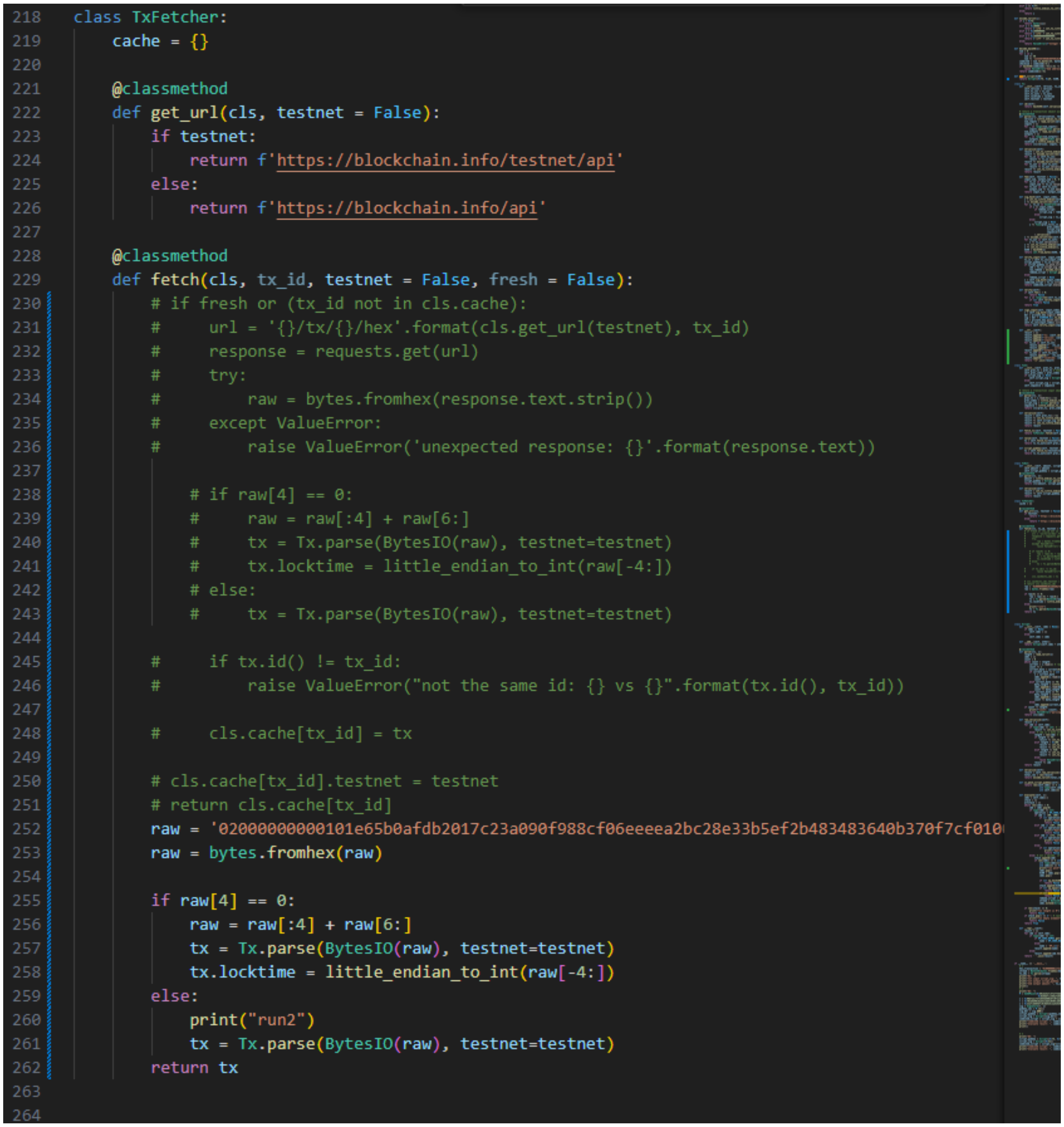
```
def p2pkh_script(h160):
    return Script([0x76, 0xa9, h160, 0x88, 0xac])
```

- **TxIn class 中的 fetch_tx, value, script_pubkey**



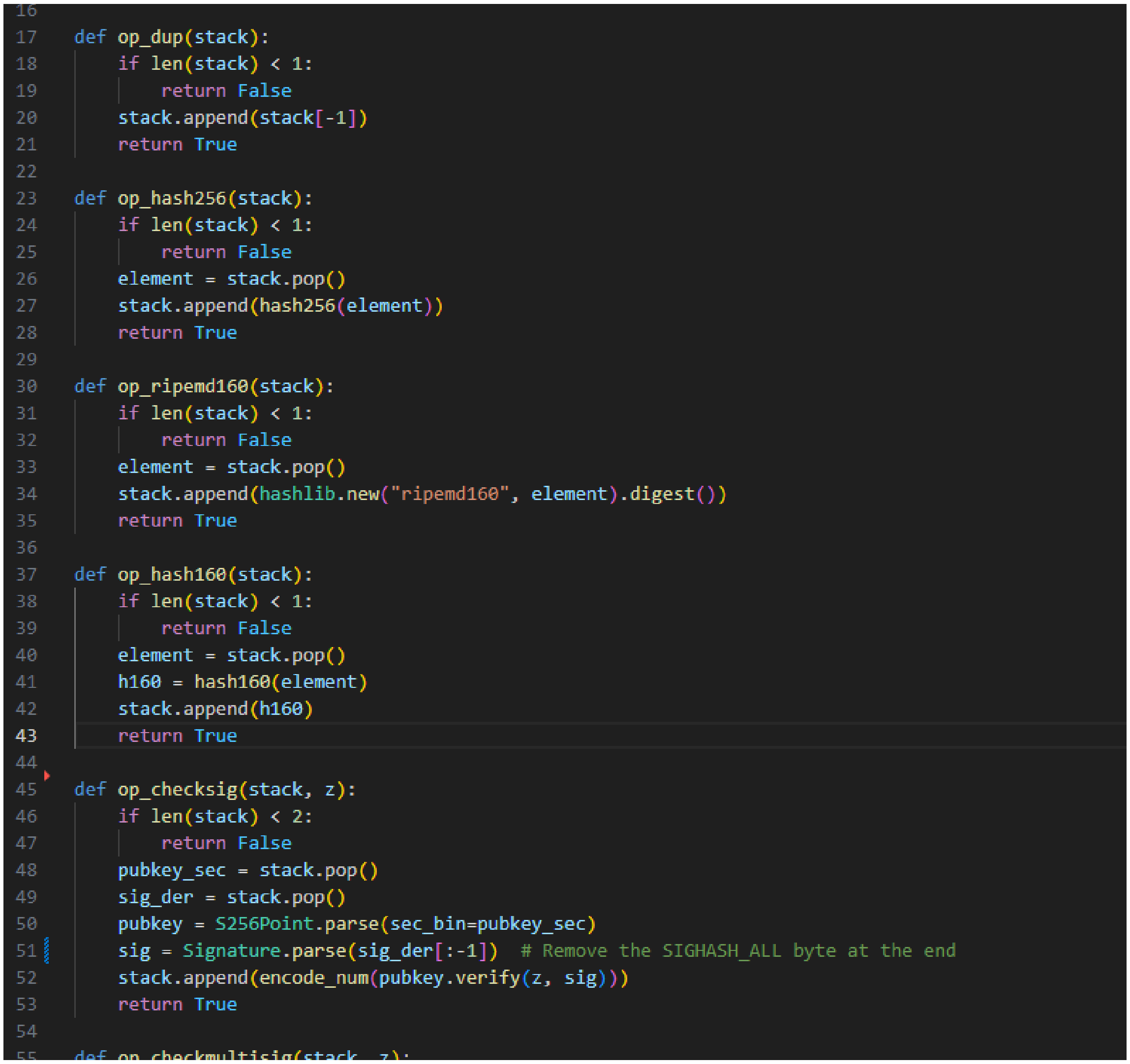
```python
    def fetch_tx(self, testnet = False):
        return TxFetcher.fetch(self.prev_tx.hex(), testnet=testnet)

    def value(self, testnet = False):
        tx = self.fetch_tx(testnet=testnet)
        return tx.tx_outs[self.prev_index].amount

    def script_pubkey(self, testnet = False):
        tx = self.fetch_tx(testnet=testnet)
        return tx.tx_outs[self.prev_index].script_pubkey

```

- **TxFetcher(裡面的 raw 直接使用從 explorer 中獲取的 raw transaction hex)**

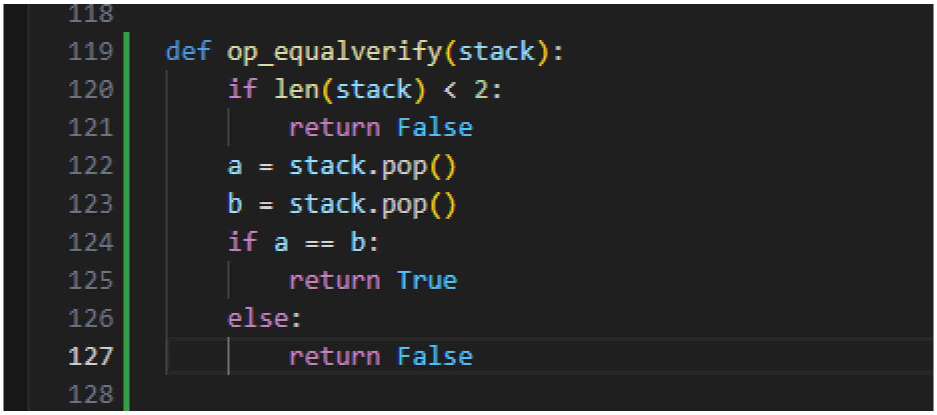


```python
class TxFetcher:
    cache = {}

    @classmethod
    def get_url(cls, testnet = False):
        if testnet:
            return f'https://blockchain.info/testnet/api'
        else:
            return f'https://blockchain.info/api'

    @classmethod
    def fetch(cls, tx_id, testnet = False, fresh = False):
        # if fresh or (tx_id not in cls.cache):
        #     url = '{}/tx/{}/hex'.format(cls.get_url(testnet), tx_id)
        #     response = requests.get(url)
        #     try:
        #         raw = bytes.fromhex(response.text.strip())
        #     except ValueError:
        #         raise ValueError('unexpected response: {}'.format(response.text))

            # if raw[4] == 0:
            #     raw = raw[:4] + raw[6:]
            #     tx = Tx.parse(BytesIO(raw), testnet=testnet)
            #     tx.locktime = little_endian_to_int(raw[-4:])
            # else:
            #     tx = Tx.parse(BytesIO(raw), testnet=testnet)

        #     if tx.id() != tx_id:
        #         raise ValueError("not the same id: {} vs {}".format(tx.id(), tx_id))

        #     cls.cache[tx_id] = tx

        # cls.cache[tx_id].testnet = testnet
        # return cls.cache[tx_id]
        raw = '02000000000101e65b0afdb2017c23a090f988cf06eeeea2bc28e33b5ef2b483483640b370f7cf010
        raw = bytes.fromhex(raw)

        if raw[4] == 0:
            raw = raw[:4] + raw[6:]
            tx = Tx.parse(BytesIO(raw), testnet=testnet)
            tx.locktime = little_endian_to_int(raw[-4:])
        else:
            print("run2")
            tx = Tx.parse(BytesIO(raw), testnet=testnet)
        return tx

```

op.py: p2pkh 所使用的 op_dup, op_hash160, op_equalverify, op_checksig

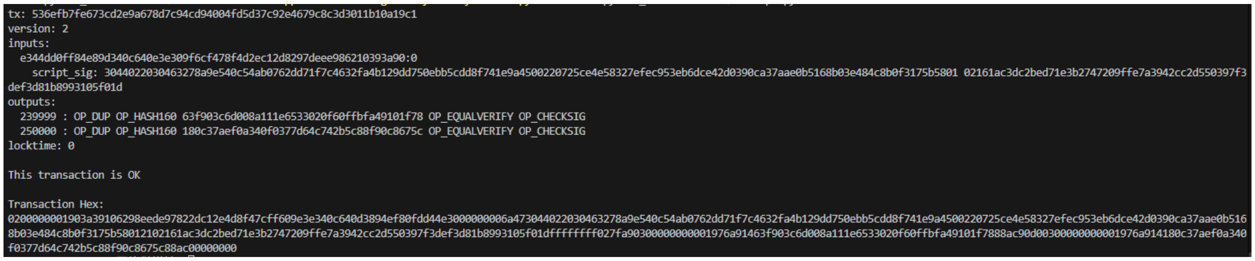


```python
def op_dup(stack):
    if len(stack) < 1:
        return False
    stack.append(stack[-1])
    return True

def op_hash256(stack):
    if len(stack) < 1:
        return False
    element = stack.pop()
    stack.append(hash256(element))
    return True

def op_ripemd160(stack):
    if len(stack) < 1:
        return False
    element = stack.pop()
    stack.append(hashlib.new("ripemd160", element).digest())
    return True

def op_hash160(stack):
    if len(stack) < 1:
        return False
    element = stack.pop()
    h160 = hash160(element)
    stack.append(h160)
    return True

def op_checksig(stack, z):
    if len(stack) < 2:
        return False
    pubkey_sec = stack.pop()
    sig_der = stack.pop()
    pubkey = S256Point.parse(sec_bin=pubkey_sec)
    sig = Signature.parse(sig_der[:-1])  # Remove the SIGHASH_ALL byte at the end
    stack.append(encode_num(pubkey.verify(z, sig)))
    return True

def op_checkmultisig(stack, z):
```



```python
def op_equalverify(stack):
    if len(stack) < 2:
        return False
    a = stack.pop()
    b = stack.pop()
    if a == b:
        return True
    else:
        return False
```

執行結果:

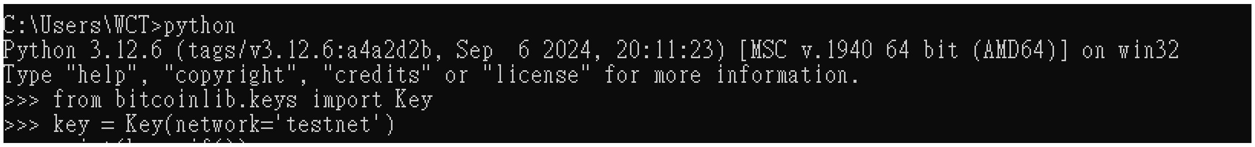


```
tx: 536efb7fe673cd2e9a678d7c94cd94004fd5d37c92e4679c8c3d3011b10a19c1
version: 2
inputs:
  e344dd0ff84e89d340c640e3e309f6cf478f4d2ec12d8297deee986210393a90:0
    script_sig: 3044022030463278a9e540c54ab0762dd71f7c4632fa4b129dd750ebb5cdd8f741e9a4500220725ce4e58327efec953eb6dce42d0390ca37aae0b5168b03e484c8b0f3175b5801 02161ac3dc2bed71e3b2747209ffe7a3942cc2d550397f3
def3d81b8993105f01d
outputs:
  239999 : OP_DUP OP_HASH160 63f903c6d008a111e6533020f60ffbfa49101f78 OP_EQUALVERIFY OP_CHECKSIG
  250000 : OP_DUP OP_HASH160 180c37aef0a340f0377d64c742b5c88f90c8675c OP_EQUALVERIFY OP_CHECKSIG
locktime: 0

This transaction is OK

Transaction Hex:
0200000001903a39106298eede97822dc12e4d8f47cff609e3e340c640d3894ef80fdd44e3000000006a473044022030463278a9e540c54ab0762dd71f7c4632fa4b129dd750ebb5cdd8f741e9a4500220725ce4e58327efec953eb6dce42d0390ca37aae0b516
8b03e484c8b0f3175b58012102161ac3dc2bed71e3b2747209ffe7a3942cc2d550397f3def3d81b8993105f01dffffffff027fa90300000000001976a91463f903c6d008a111e6533020f60ffbfa49101f7888ac90d00300000000001976a914180c37aef0a340
f0377d64c742b5c88f90c8675c88ac00000000
```

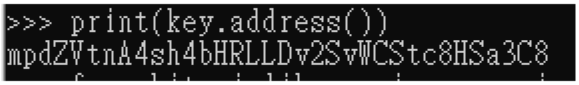
(將完成的 Tx 內容，以及 raw Tx serialization 印出)

- 步驟說明

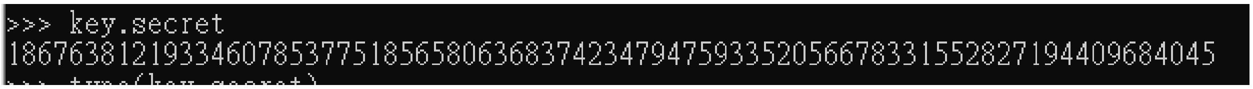
Step1. 以 **bitcoinlib** 建立 **key pair**



```
C:\Users\WCT>python
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from bitcoinlib.keys import Key
>>> key = Key(network='testnet')
```



```
>>> print(key.address())
mpdZVtnA4sh4bHRLLDv2SvWCStc8HSa3C8
```



```
>>> key.secret
18676381219334607853775185658063683742347947593352056678331552827194409684045
```

Step2. 到 **https://faucet.testnet4.dev/**獲取 **testnet4 bitcoin**

Step3. 建立 **transaction**

1. 建立 **input**，選擇 **TxID** 為，從 **faucet** 獲取 **testnet4 bitcoin** 的 **TxID**，並將 **index(Vout)**設為該交易中的指定 **output(0)**
2. 建立雙方的 **output**，第一個 **output** 是給自己的找零，設為 **0.0024*100000000 sat**，第二個是給對方的 **bitcoin**，設為 **0.0025*100000000 sat (input** 有 **0.005BTC**，因此提供的 **fee** 為 **0.0001BTC)**，並且皆以 **p2pkh** 進行 **locking**
3. 將 **input** 與 **output** 建立為一個 **transaction**
4. 為該 **transaction** 的 **input** 進行 **signing**
   (1) 找到該 **input** 的 **sig_hash z**
   (2) 以 **private key** 及 **z** 建立 **sig**
   (3) 以 **private key** 建立 **sec**
   (4) 建成一個 **Script object**，提供給 **transaction** 的 **input**，完成 **signing**
5. 對該交易進行驗證，檢驗是否生成新的 **bitcoin**，以及 **input** 的 **ScriptSig** 是否能 **unlocking** 他所指定的 **output** 的 **ScriptPubkey**
6. 生成該交易的 **raw transaction hex**

Step4. 到 **https://mempool.space/testnet4/tx/push** 廣播該筆交易

2. Complete the try except statement in the op_checkmultisig(stack, z) function (you may place it in op.py, any .py file, or a Notebook cell)

```
def op_checkmultisig(stack, z):
    if len(stack) < 1:
        return False
    n = decode_num(stack.pop())
    if len(stack) < n + 1:
        return False
    sec_pubkeys = []
    for _ in range(n):
        sec_pubkeys.append(stack.pop())
    m = decode_num(stack.pop())
    if len(stack) < m + 1:
        return False
    der_signatures = []
    for _ in range(m):
        der_signatures.append(stack.pop()[:-1])  # Each DER signature is assumed to be signed with SIGHASH_ALL
    stack.pop()  # Take care of the off-by-one error by consuming the only remaining element of the stack and not doing anything with the element
    try:
        raise NotImplementedError  # The part that you need to code for this problem
    except (ValueError, SyntaxError):
        return False
    return True
```

You need to:

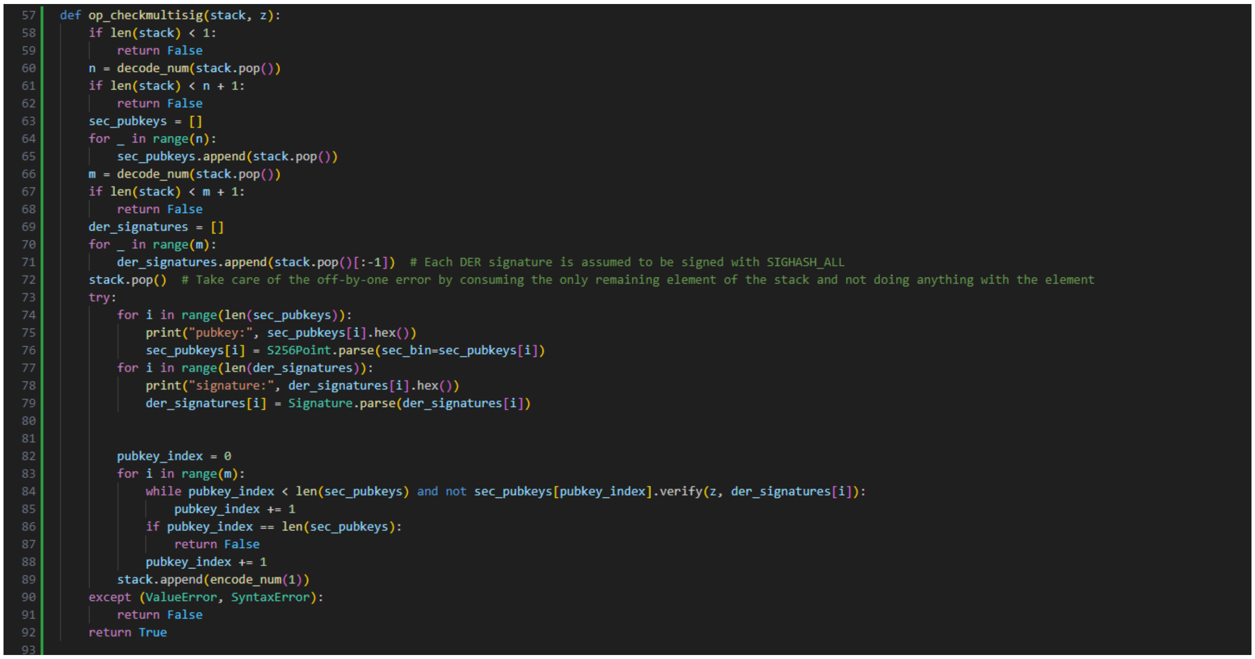
- Parse all the points.
- Parse all the signatures.
- Loop through the signatures.
    - If we have no more points, signatures are no good.
    - Loop until we find the point which works with this signature.
        - Get the current point from the list of points.
        - Check if this signature goes with the current point.
- If the signatures are valid, push a 1 to the stack

# Grading for Problem 2:
 - Total: 30%
  - Correct code: 30%
   (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code below and attach the .py file. TAs will run the test with this file. ---

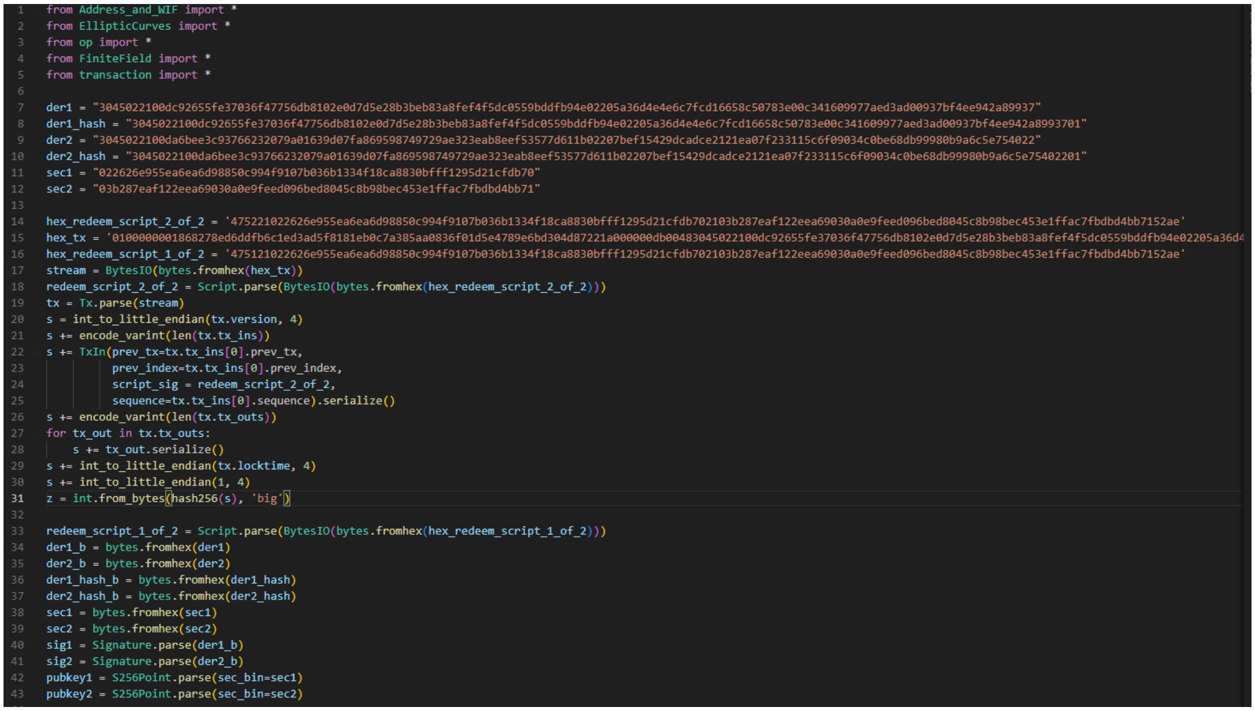
檔案繳交備註: 我在網大放上了 op.py, Address_and_WIF.py, ElipticCurve.py, FiniteField.py 合計四個 python 檔案，其中 op_checkmultisig(stack, z)在 op.py 中，其他的檔案是執行這個 method 需要呼叫到的副程式。

Code: op.py 中的 op_checkmultisig()

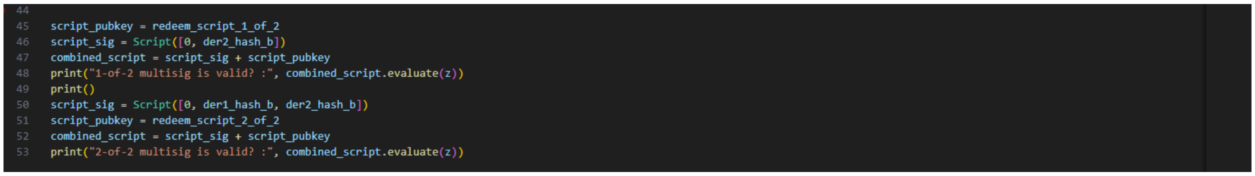


```python
def op_checkmultisig(stack, z):
    if len(stack) < 1:
        return False
    n = decode_num(stack.pop())
    if len(stack) < n + 1:
        return False
    sec_pubkeys = []
    for _ in range(n):
        sec_pubkeys.append(stack.pop())
    m = decode_num(stack.pop())
    if len(stack) < m + 1:
        return False
    der_signatures = []
    for _ in range(m):
        der_signatures.append(stack.pop()[:-1])  # Each DER signature is assumed to be signed with SIGHASH_ALL
    stack.pop()  # Take care of the off-by-one error by consuming the only remaining element of the stack and not doing anything with the element
    try:
        for i in range(len(sec_pubkeys)):
            print("pubkey:", sec_pubkeys[i].hex())
            sec_pubkeys[i] = S256Point.parse(sec_bin=sec_pubkeys[i])
        for i in range(len(der_signatures)):
            print("signature:", der_signatures[i].hex())
            der_signatures[i] = Signature.parse(der_signatures[i])


        pubkey_index = 0
        for i in range(m):
            while pubkey_index < len(sec_pubkeys) and not sec_pubkeys[pubkey_index].verify(z, der_signatures[i]):
                pubkey_index += 1
            if pubkey_index == len(sec_pubkeys):
                return False
            pubkey_index += 1
        stack.append(encode_num(1))
    except (ValueError, SyntaxError):
        return False
    return True
```

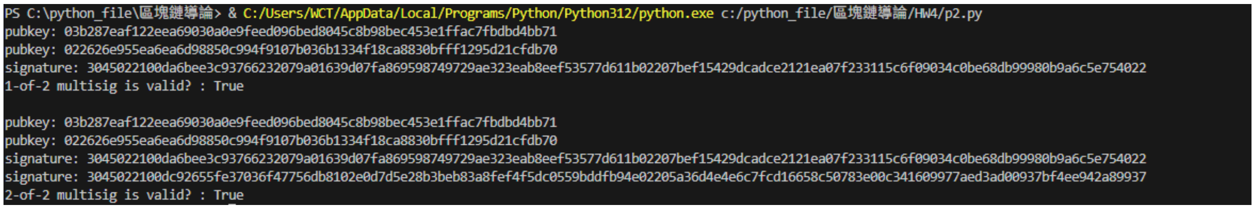
Result: (有自己以第三題提供的 redeem_script 與 signature 來測試)



```python
from Address_and_WIF import *
from EllipticCurves import *
from op import *
from FiniteField import *
from transaction import *

der1 = "3045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a89937"
der1_hash = "3045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a8993701"
der2 = "3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022"
der2_hash = "3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e75402201"
sec1 = "022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb70"
sec2 = "03b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb71"

hex_redeem_script_2_of_2 = '475221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb7152ae'
hex_tx = '0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a385aa0836f01d5e4789e6bd304d87221a000000db00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4
hex_redeem_script_1_of_2 = '475121022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb7152ae'
stream = BytesIO(bytes.fromhex(hex_tx))
redeem_script_2_of_2 = Script.parse(BytesIO(bytes.fromhex(hex_redeem_script_2_of_2)))
tx = Tx.parse(stream)
s = int_to_little_endian(tx.version, 4)
s += encode_varint(len(tx.tx_ins))
s += TxIn(prev_tx=tx.tx_ins[0].prev_tx,
          prev_index=tx.tx_ins[0].prev_index,
          script_sig = redeem_script_2_of_2,
          sequence=tx.tx_ins[0].sequence).serialize()
s += encode_varint(len(tx.tx_outs))
for tx_out in tx.tx_outs:
    s += tx_out.serialize()
s += int_to_little_endian(tx.locktime, 4)
s += int_to_little_endian(1, 4)
z = int.from_bytes(hash256(s), 'big')

redeem_script_1_of_2 = Script.parse(BytesIO(bytes.fromhex(hex_redeem_script_1_of_2)))
der1_b = bytes.fromhex(der1)
der2_b = bytes.fromhex(der2)
der1_hash_b = bytes.fromhex(der1_hash)
der2_hash_b = bytes.fromhex(der2_hash)
sec1 = bytes.fromhex(sec1)
sec2 = bytes.fromhex(sec2)
sig1 = Signature.parse(der1_b)
sig2 = Signature.parse(der2_b)
pubkey1 = S256Point.parse(sec_bin=sec1)
pubkey2 = S256Point.parse(sec_bin=sec2)
```



```python

script_pubkey = redeem_script_1_of_2
script_sig = Script([0, der2_hash_b])
combined_script = script_sig + script_pubkey
print("1-of-2 multisig is valid? :", combined_script.evaluate(z))
print()
script_sig = Script([0, der1_hash_b, der2_hash_b])
script_pubkey = redeem_script_2_of_2
combined_script = script_sig + script_pubkey
print("2-of-2 multisig is valid? :", combined_script.evaluate(z))
```



```
PS C:\python_file\區塊鏈導論> & C:/Users/WCT/AppData/Local/Programs/Python/Python312/python.exe c:/python_file/區塊鏈導論/HW4/p2.py
pubkey: 03b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb71
pubkey: 022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb70
signature: 3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022
1-of-2 multisig is valid? : True

pubkey: 03b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb71
pubkey: 022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb70
signature: 3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022
signature: 3045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a89937
2-of-2 multisig is valid? : True
```

3. Validate the second signature from the following transaction.

- from io import BytesIO
- from ecc import S256Point, Signature
- from helper import encode_varint, hash256, int_to_little_endian
- from script import Script
- from tx import Tx, SIGHASH_ALL # SIGHASH_ALL = 1, SIGHASH_NONE = 2, SIGHASH_SINGLE = 3
- hex_tx = '0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a385aa0836f01d5e4789e6bd304d87221a000000db00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a8993701483045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e75402201475221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb7152aeffffffff04d3b11400000000001976a914904a49878c0adfc3aa05de7afad2cc15f483a56a88ac7f400900000000001976a914418327e3f3dda4cf5b9089325a4b95abdfa

0334088ac722c0c00000000001976a914ba35042cfe9fc66fd35ac2224eebdafd1028ad2788acdc4ace020000000017a91474d691da1574e6b3c192ecfb52cc8984ee7b6c568700000000'

- hex_sec = '03b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb71' # the second sec public key
- hex_der = '3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022' # the DER-encoded value that appears second in the ScriptSig of the transaction
- hex_redeem_script = '475221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb7152ae'
- sec = bytes.fromhex(hex_sec)
- der = bytes.fromhex(hex_der)
- redeem_script = Script.parse(BytesIO(bytes.fromhex(hex_redeem_script)))
- stream = BytesIO(bytes.fromhex(hex_tx))

You need to:

1. Modify the transaction
2. Start with version

3. Add number of inputs
4. Modify the single TxIn to have the ScriptSig to be the RedeemScript
5. Add the number of outputs
6. Add each output serialization
7. Add the locktime
8. Add the SIGHASH_ALL
9. Hash256 the result
10. Interpret as a Big-Endian number
11. Parse the S256Point
12. Parse the Signature
13. Verify that the point, z and signature work

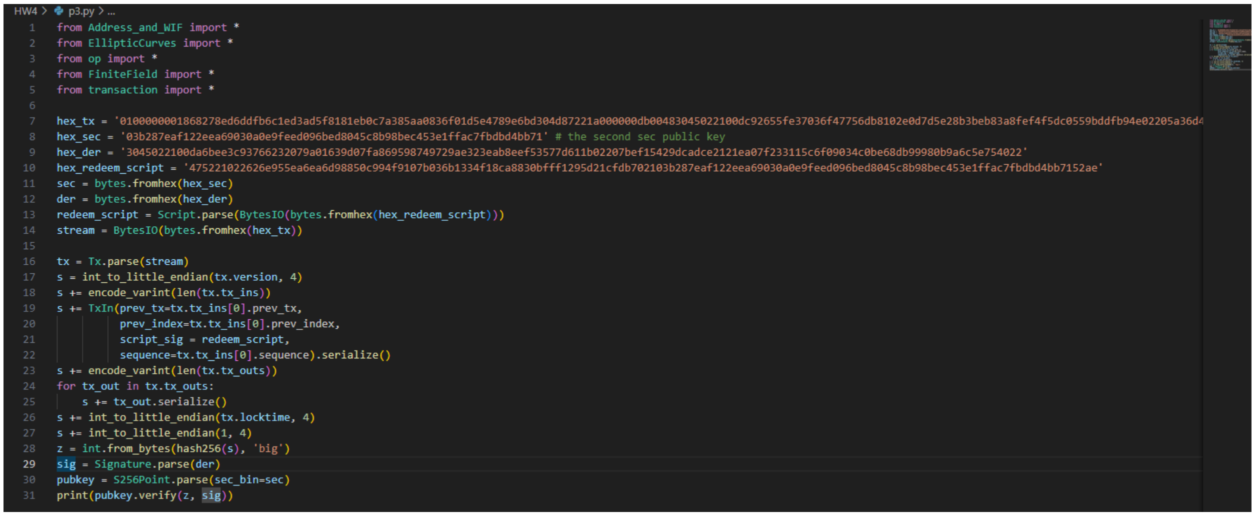
# Grading for Problem 3:

- Total: 30%
  - Correct code and code execution result: 30%

    (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of <u>your code and code execution result</u> of solving problem 3 below. ---

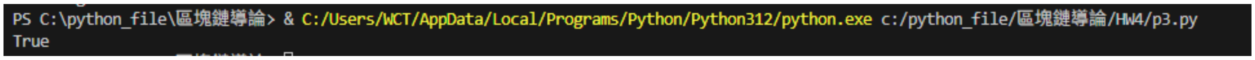
Code:



HW4 > p3.py > ...

```python
from Address_and_WIF import *
from EllipticCurves import *
from op import *
from FiniteField import *
from transaction import *

hex_tx = '0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a385aa0836f01d5e4789e6bd304d87221a000000db00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e02205a36d4
hex_sec = '03b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb71' # the second sec public key
hex_der = '3045022100da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022'
hex_redeem_script = '475221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec453e1ffac7fbdbd4bb7152ae'
sec = bytes.fromhex(hex_sec)
der = bytes.fromhex(hex_der)
redeem_script = Script.parse(BytesIO(bytes.fromhex(hex_redeem_script)))
stream = BytesIO(bytes.fromhex(hex_tx))

tx = Tx.parse(stream)
s = int_to_little_endian(tx.version, 4)
s += encode_varint(len(tx.tx_ins))
s += TxIn(prev_tx=tx.tx_ins[0].prev_tx,
          prev_index=tx.tx_ins[0].prev_index,
          script_sig = redeem_script,
          sequence=tx.tx_ins[0].sequence).serialize()
s += encode_varint(len(tx.tx_outs))
for tx_out in tx.tx_outs:
    s += tx_out.serialize()
s += int_to_little_endian(tx.locktime, 4)
s += int_to_little_endian(1, 4)
z = int.from_bytes(hash256(s), 'big')
sig = Signature.parse(der)
pubkey = S256Point.parse(sec_bin=sec)
print(pubkey.verify(z, sig))
```

Result:



```
PS C:\python_file\區塊鏈導論> & C:/Users/WCT/AppData/Local/Programs/Python/Python312/python.exe c:/python_file/區塊鏈導論/HW4/p3.py
True
```