National Sun Yat-sen University
Introduction To Blockchain Technology
Homework 3
Course Number: CSE222, Chapter: 5 & 6

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 3.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. What is the ScriptSig from the second input, ScriptPubKey from the first output and the amount of the second output for this transaction?

- hex_transaction = 010000000117e18a4a4a0af876b1b0a4764ee77c74106e07667dd94c4d61271f3d356cbf62000000006b4830450221009e661e94622a66f6c65f270d859828360c825ee755d675c9cbb2214685ba08fc022005aa4abaf21a84519f0c8ff40c633a0e4a624c639d25c0ea908d0d5e463749a80121036ddc934a5fbd5222ead406a4334462aaa62f83d0b02255c0a582f9038a17bbfdfffffff02cc162c000000000001976a914051b0771687183369 4a762ad15565b86da46622488ac16ae0e000000000001976a914c03ee4258550c77bcf61829c7cb636cd521ebfc588ac00000000

- Hint: Convert the hex_transaction to binary, create a stream using BytesIO, and use Tx.parse to get the transaction object
- You should print the answers

Script.__rept__ in script.py:

```
class Script:
…
    def __repr__(self):
        result = []
        for cmd in self.cmds:
            if type(cmd) == int:
                if OP_CODE_NAMES.get(cmd):
                    name = OP_CODE_NAMES.get(cmd)
                else:
                    name = 'OP_[{}]'.format(cmd)
                result.append(name)
            else:
                result.append(cmd.hex())
        return ' '.join(result)
…
```

Grading for Problem 1:
- Total: 40%
  - Correct code and code execution result: 40%
    (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. ---

Code:

```python
from Address_and_WIF import *
from io import BytesIO
from op import OP_CODE_FUNCTIONS, OP_CODE_NAMES

def little_endian_to_int(b):
    return int.from_bytes(b, byteorder = 'little')

def int_to_little_endian(i, length):
    return i.to_bytes(length, byteorder = 'little')

def read_varint(s):
    i = s.read(1)[0]
    if i == 0xfd:
        return little_endian_to_int(s.read(2))
    elif i == 0xfe:
        return little_endian_to_int(s.read(4))
    elif i == 0xff:
        return little_endian_to_int(s.read(8))
    else:
        return i

def encode_varint(i):
    if i < 0xfd:
        return bytes([i])
    elif i < 0x10000:
        return b'\xfd' + int_to_little_endian(i, 2)
    elif i < 0x100000000:
        return b'\xfe' + int_to_little_endian(i, 4)
    elif i < 0x10000000000000000:
        return b'\xff' + int_to_little_endian(i, 8)
    else:
        return ValueError(f"Integer too large: {i}")

```

```python
class Tx:
    def __init__(self, version, tx_ins, tx_outs, locktime, testnet = False):
        self.version = version
        self.tx_ins = tx_ins
        self.tx_outs = tx_outs
        self.locktime = locktime
        self.testnet = testnet

    def id(self):
        return hash256(self.serialize())[::-1].hex()

    # return a transaction object according to the serialization
    @classmethod
    def parse(cls, serialization, testnet = False):
        version = little_endian_to_int(serialization.read(4))
        num_inputs = read_varint(serialization)
        inputs = []
        for _ in range(num_inputs):
            inputs.append(TxIn.parse(serialization))
        num_outputs = read_varint(serialization)
        outputs = []
        for _ in range(num_outputs):
            outputs.append(TxOut.parse(serialization))
        locktime = little_endian_to_int(serialization.read(4))
        return cls(version, inputs, outputs, locktime, testnet=testnet)

    def serialize(self):
        result = int_to_little_endian(self.version, 4)
        result += encode_varint(len(self.tx_ins))
        for tx_in in self.tx_ins:
            result += tx_in.serialize()
        result += encode_varint(len(self.tx_outs))
        for tx_out in self.tx_outs:
            result += tx_out.serialize()
        result += int_to_little_endian(self.locktime, 4)
        return result
```

```python
class TxIn:
    def __init__(self, prev_tx, prev_index, script_sig = None, sequence = 0xffffffff):
        self.prev_tx = prev_tx # 32bytes byte string. last UTXO ID(result of hash256 of the previous transaction's serialization)
        self.prev_index = prev_index
        if script_sig is None:
            self.script_sig = Script()
        else:
            self.script_sig = script_sig
        self.sequence = sequence

    # return a transaction input object according to the serialization
    @classmethod
    def parse(cls, s):
        prev_tx = s.read(32)[::-1]
        prev_index = little_endian_to_int(s.read(4))
        script_sig = Script.parse(s)
        sequence = little_endian_to_int(s.read(4))
        return cls(prev_tx, prev_index, script_sig, sequence)

    def serialize(self):
        result = self.prev_tx[::-1]
        result += int_to_little_endian(self.prev_index, 4)
        result += self.script_sig.serialize()
        result += int_to_little_endian(self.sequence, 4)
```

```python

class TxOut:
    def __init__(self, amount, script_pubkey):
        self.amount = amount
        self.script_pubkey = script_pubkey
    @classmethod
    def parse(cls, s):
        amount = little_endian_to_int(s.read(8))
        script_pubkey = Script.parse(s)
        return cls(amount, script_pubkey)

    def serialize(self):
        result = int_to_little_endian(self.amount, 8)
        result += self.script_pubkey.serialize()
        return result


```

```python
class Script:
    def __init__(self, cmds = None):
        if cmds is None:
            self.cmds = []
        else:
            self.cmds = cmds

    def __add__(self, other):
        return Script(self.cmds + other.cmds)

    @classmethod
    def parse(cls, s):
        length = read_varint(s)
        cmds = []
        count = 0
        while count < length:
            current = s.read(1) # read a byte means next command
            count += 1
            current_byte = current[0] # from bytes to int
            if current_byte >= 1 and current_byte <= 75: # means the length of the next command
                n = current_byte
                cmds.append(s.read(n))
                count += n
            elif current_byte == 76: # OP_PUSHDATA1, next byte is the length of the next command
                data_length = little_endian_to_int(s.read(1))
                cmds.append(s.read(data_length))
                count += data_length + 1
            elif current_byte == 77: # OP_PUSHDATA2, next two bytes are the length of the next command
                data_length = little_endian_to_int(s.read(2))
                cmds.append(s.read(data_length))
                count += data_length + 2
            else:
                cmds.append(current_byte)
        if count != length:
            raise ValueError("parsing script failed")
        return cls(cmds)

```

```python
    def raw_serialize(self):
        result = b''
        for cmd in self.cmds:
            if type(cmd) == int:
                result += int_to_little_endian(cmd, 1)
            else: # cmd is a byte string
                length = len(cmd) # 下面這些if在把command的長度放入
                if length <= 75:
                    result += int_to_little_endian(length, 1)
                elif length < 0x100:
                    result += int_to_little_endian(76, 1)
                    result += int_to_little_endian(length, 1)
                elif length <= 520:
                    result += int_to_little_endian(77, 1)
                    result += int_to_little_endian(length, 2)
                else:
                    raise ValueError('too long a cmd')
                result += cmd
        return result

    def serialize(self):
        result = self.raw_serialize()
        total_len = len(result)
        return encode_varint(total_len) + result

```

```python
    def evaluate(self, z):
        cmds = self.cmds[:]
        stack = []
        altstack = []
        while len(cmds) > 0:
            cmd = cmds.pop(0)
            if type(cmd) == int:
                operation = OP_CODE_FUNCTIONS[cmd]
                if cmd in (99, 100): # OP_IF, OP_NOTIF
                    if not operation(stack, cmds):
                        print(f"Operation {OP_CODE_NAMES[cmd]} failed")
                        return False
                elif cmd in (107, 108): # OP_TOALTSTACK, OP_FROMALTSTACK
                    if not operation(stack, altstack):
                        print(f"Operation {OP_CODE_NAMES[cmd]} failed")
                        return False
                elif cmd in (172, 173, 174, 175): # OP_CHECKSIG, OP_CHECKSIGVERIFY, OP_CHECKMULTISIG, OP_CHECKMULTISIGVERIFY
                    if not operation(stack, z):
                        print(f"Operation {OP_CODE_NAMES[cmd]} failed")
                        return False
                else:
                    if not operation(stack):
                        print(f"Operation {OP_CODE_NAMES[cmd]} failed")
                        return False
            else: # not a opcode, it's an element(e.g. signature, pubkey)
                stack.append(cmd)
        if len(stack) == 0:
            print("run length is 0")
            return False
        if stack.pop() == b'': # last element is empty, means the script is not valid
            print("run empty element")
            return False
        return True

    def __repr__(self):
        result = []
        for cmd in self.cmds:
            if type(cmd) == int:
                if OP_CODE_NAMES.get(cmd):
                    name = OP_CODE_NAMES.get(cmd)
                else:
                    name = 'OP_[{}]'.format(cmd)
                result.append(name)
            else:
                result.append(cmd.hex())
        return ' '.join(result)
```

```python
if __name__ == '__main__':
    # 1
    hex_transaction = "010000000117e18a4a4a0af876b1b0a4764ee77c74106e07667dd94c4d61271f3d356cbf6200000000b483045022100 9e661e94622a66f6c65f270d859828360c825ee755d675c9cbb2214685ba08fc022005aa4abaf21a84519f0c8ff40c633a0e4a624c639d25c0ea908d0d5e463749a801 036ddc934a5fbd5222ead406a4334462aaa62f83d0b02255c0a582f9038a17bbfd"
    stream = BytesIO(bytes.fromhex(hex_transaction))
    tx_obj = Tx.parse(stream)
    print("Q1: ")
    print("1st input script_sig: ", tx_obj.tx_ins[0].script_sig)
    print("1st output script_pubkey: ", tx_obj.tx_outs[0].script_pubkey)
    print("2nd output amount: ", tx_obj.tx_outs[1].amount)
    print()
```

Result:

```
C:\python_file\區塊鏈導論\HW3>python transaction.py
Q1:
1st input script_sig:  30450221009e661e94622a66f6c65f270d859828360c825ee755d675c9cbb2214685ba08fc022
005aa4abaf21a84519f0c8ff40c633a0e4a624c639d25c0ea908d0d5e463749a801 036ddc934a5fbd5222ead406a4334462
aaa62f83d0b02255c0a582f9038a17bbfd
1st output script_pubkey:  OP_DUP OP_HASH160 051b07716871833694a762ad15565b86da466224 OP_EQUALVERIFY
 OP_CHECKSIG
2nd output amount:  962070
```

2. Implement an op_checksig(stack, z) function (you may place it in op.py, any .py file, or a Notebook cell)

Grading for Problem 2:

- Total: 30%

- Correct code: 30%
  (Only have correct code and correct code execution can get score.)

--- Please paste screenshot of <u>your code and attach the .py file</u>. We 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_checksig(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。

程式說明: 考量到作業四當中，進行 p2pkh 時，會在 signature 的 DER 後方加上 1byte 的 sighash type，因此如同 checkmultisig 相同，對於從 stack 中取出的 signature 消除最後一個 byte 的 sighash type

op_checksig(stack, z)的部分:

```python
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
```

自行設計的測試 code 以及結果:
(pubkey, signature, z 的值，使用作業二的第一題的值)

```python
    # 2
    print("Q2: ")
    P = S256Point(0x801be5a7c4faf73dd1c3f28cebf78d6ba7885ead88879b76ffb815d59056af14,
                  0x826ddfcc38dafe6b8d463b609facc009083c8173e21c5fc45b3424964e85f49e)
    z = 0x90d7aecf3f2855d60026f10faab852562c76e7e043cf243474ba5018447c2c22
    r = 0xf01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f
    s = 0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8
    sig = Signature(r, s)
    pubkey_sec = P.sec()
    der = sig.DER()
    sig = der+int(1).to_bytes(1, 'big')
    script_pubkey = Script([pubkey_sec, 0xac])
    script_sig = Script([sig])
    combined_script = script_sig + script_pubkey
    print("combined script: ", combined_script)
    print("evaluate result: ", combined_script.evaluate(z))
    print()
```

```
Q2:
combined script:  3045022100f01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f022022afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d801 02801be5a7c4faf73dd1c3f28cebf78d6ba7885ead8
8879b76ffb815d59056af14 OP_CHECKSIG
evaluate result:  True
```

整個 **op.py(**包含其他的 **OPCODE):**

這邊 **import** 的 **Address_and_WIF**，裡面包含了 **S256Point** 以及其 **verify** 的 **method**

```python
import hashlib
from Address_and_WIF import *


def hash256(s):
    return hashlib.sha256(hashlib.sha256(s).digest()).digest()

def hash160(s):
    return hashlib.new("ripemd160", hashlib.sha256(s).digest()).digest()

def little_endian_to_int(b):
    return int.from_bytes(b, byteorder = 'little')

def int_to_little_endian(i, length):
    return i.to_bytes(length, byteorder = 'little')

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
```

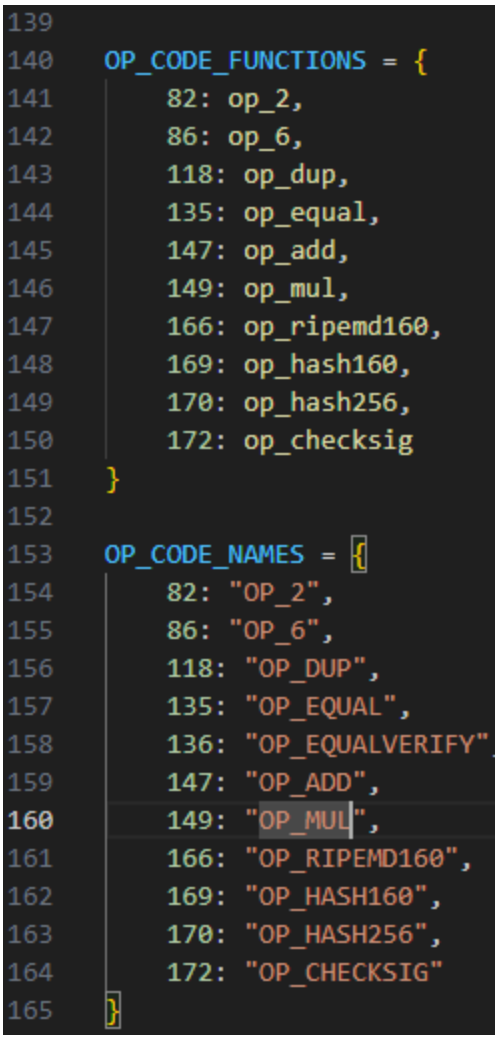
```
47  def op_checksig(stack, z):
48      if len(stack) < 2:
49          return False
50      pubkey_sec = stack.pop()
51      sig_der = stack.pop()
52      pubkey = S256Point.parse(sec_bin=pubkey_sec)
53      sig = Signature.parse(sig_der)
54      stack.append(encode_num(pubkey.verify(z, sig)))
55      return True
56
57  def OP_IF(stack, cmds):
58      pass
59
60  def OP_NOTIF(stack, cmds):
61      pass
62
63  def op_6(stack):
64      stack.append(encode_num(6))
65      return True
66
67  def op_2(stack):
68      stack.append(encode_num(2))
69      return True
70
71  def op_equal(stack):
72      if len(stack) < 2:
73          return False
74      a = stack.pop()
75      b = stack.pop()
76      stack.append(encode_num(a == b))
77      return True
78
79  def op_add(stack):
80      if len(stack) < 2:
81          return False
82      a = decode_num(stack.pop())
83      b = decode_num(stack.pop())
84      stack.append(encode_num(a + b))
85      return True
86
87  def op_mul(stack):
88      if len(stack) < 2:
89          return False
90      a = decode_num(stack.pop())
91      b = decode_num(stack.pop())
92      stack.append(encode_num(a * b))
93      return True
```

```python
def encode_num(num):
    if num == 0:
        return b''
    abs_num = abs(num)
    negative = num < 0
    result = bytearray()
    while abs_num: # convert number to little endian
        result.append(abs_num & 0xff)
        abs_num >>= 8
    if result[-1] & 0x80:
        if negative:
            result.append(0x80)
        else:
            result.append(0)
    elif negative:
        result[-1] |= 0x80
    return bytes(result)

def decode_num(element):
    if element == b'':
        return 0
    big_endian = element[::-1]
    if big_endian[0] == 0x80:
        negative = True
        result = big_endian[0] & 0x7f
    else:
        negative = False
        result = big_endian[0]
    for c in big_endian[1:]: # byte 0已經放入，現在只要在放1之後的byte
        result <<= 8
        result += c
    if negative:
        return -result
    else:
        return result

def op_0(stack):
    stack.append(encode_num(0))
    return True

```



```
139
140 OP_CODE_FUNCTIONS = {
141     82: op_2,
142     86: op_6,
143     118: op_dup,
144     135: op_equal,
145     147: op_add,
146     149: op_mul,
147     166: op_ripemd160,
148     169: op_hash160,
149     170: op_hash256,
150     172: op_checksig
151 }
152
153 OP_CODE_NAMES = {
154     82: "OP_2",
155     86: "OP_6",
156     118: "OP_DUP",
157     135: "OP_EQUAL",
158     136: "OP_EQUALVERIFY",
159     147: "OP_ADD",
160     149: "OP_MUL",
161     166: "OP_RIPEMD160",
162     169: "OP_HASH160",
163     170: "OP_HASH256",
164     172: "OP_CHECKSIG"
165 }
```

3. Create a ScriptSig that can unlock this ScriptPubKey. Note `OP_MUL` multiplies the top two elements of the stack:

- 767695935687
    - script_pubkey = Script([0x76, 0x76, 0x95, 0x93, 0x56, 0x87])
- 56 = OP_6
- 76 = OP_DUP
- 87 = OP_EQUAL
- 93 = OP_ADD
- 95 = OP_MUL
- You can look up what various opcodes do at https://en.bitcoin.it/wiki/Script
- Hint: You should combine this ScriptPubKey with your ScriptSig, evaluate the combined script, and print the result

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. ---

採用的 **ScriptSig: [0x52]**，即僅有一個 **OP_2** 的 **command**

Code:

```
251     # 3
252     print("Q3: ")
253     script_pubkey = Script([0x76, 0x76, 0x95, 0x93, 0x56, 0x87])
254     script_sig = Script([0x52])
255     combined_script = script_sig+script_pubkey
256     print("combined script: ", combined_script)
257     print("evaluate result: ", combined_script.evaluate(0))
```

Result:

```
Q3:
combined script:  OP_2 OP_DUP OP_DUP OP_MUL OP_ADD OP_6 OP_EQUAL
evaluate result:  True
PS C:\python_file\區塊鏈導論>
```