블록체인 실습 – 지갑주소 생성

■ 지갑주소 생성 코드 작성

- **비트코인 개인키, 공개키 규격은 secp256k1을 채택**
  : $y^2 = x^3 + 7 \, mod \, p$ 사용
- **btc.random_key()**
  : 비트코인에서 사용할 개인키 생성
  : 256 bit의 랜덤한 수를 생성
- **btc.decode_privkey(privKey, 'hex')**
- 16진수 문자열을 10진수 숫자로 변환
- **dPrivKey < btc.N**
  : secp256k1에서 사용하는 N보다 작으면 통과
- **btc,pubkey_to_address(pubKey, version prefix)**
  : version prefix 값이 0일 경우 mainnet용,
  0x6f일 경우 testnet용의 지갑 주소를 생성

```
import bitcoin.main as btc

# 개인키를 생성한다
while (1):
    privKey = btc.random_key()
    dPrivKey = btc.decode_privkey(privKey, 'hex')
    if dPrivKey < btc.N:
        break

# 개인키로 공개키를 생성
pubKey = btc.privkey_to_pubkey(privKey)

# 공개키로 지갑 주소를 생성(mainnet 용)
addressmain = btc.pubkey_to_address(pubKey, 0)

# 공개키로 지갑 주소를 생성(testnet 용)
addresstest = btc.pubkey_to_address(pubKey, 0x6f)

# 결과 확인
print("₩n개인키 : ", privKey)
print("₩n공개키 : ", pubKey)
print("₩n지갑주소 (mainnet 용) : ", addressmain)
print("₩n지갑주소 (Testnet 용) : ", addresstest)
```

비트코인 실습 – 지갑주소 생성

■ 출력 화면

```
개인키 :  6bf6e773bc897a8fe52b14867e589d7070aa23c68021aa642e2ae76473f91cca
공개키 :  04d910272aa82d6ff0e073648798180a901f55f3d6b20fd2c4d133ff7030bcf73eb28e58fdf221c7340bab5e30e195b4a820e3d8531d7424f423a078375f62af5b
지갑주소 (mainnet 용) :  1BLMjVrC2bPmysUKNBt4YzsCxHLXy7ERWA
지갑주소 (Testnet 용) :  mqrK2YwAqcq2kyww5krSNv5XpGwEvhTCak
```

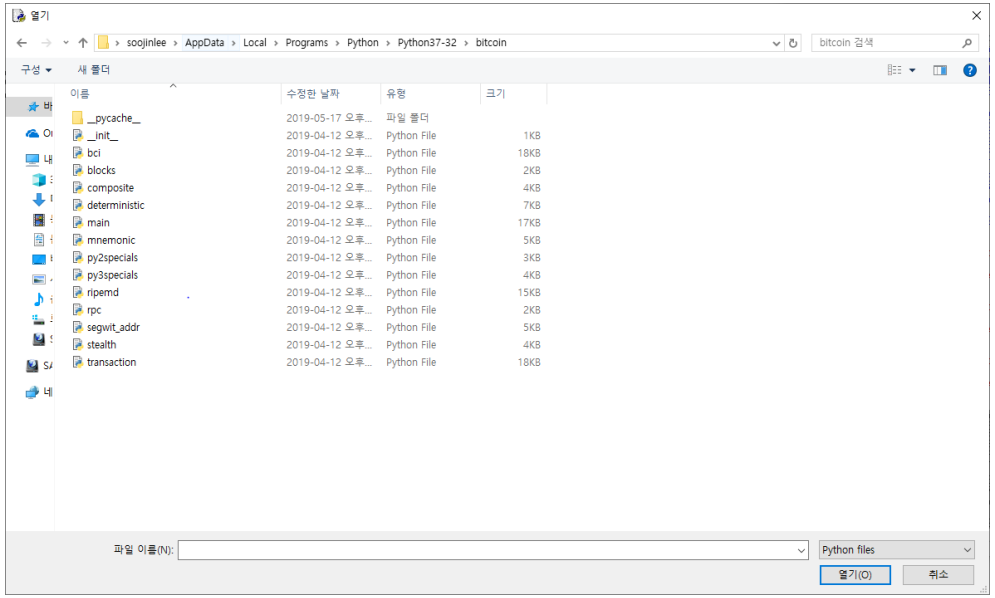
- Mainnet용 지갑주소의 경우 1부터 시작
- Testnet용 지갑주소는 m또는 n부터 시작

비트코인 실습 – transaction (비트코인 testnet에서)

■ 실습 전, Blackboard에 올라온 비트코인 파일을 다운받기

■ 앞으로 작성할 파이썬 코드 파일 위치와 위치가 같아야 함

- Bitcoin 파일 내부



비트코인 실습 – transaction (비트코인 testnet에서)

■ 1. 앞의 지갑주소 생성하는 코드를 통해 개인키, 지갑주소 2쌍을 생성하여 코드에 입력

```
from bitcoin.bci import history
from bitcoin.transaction import mktx, sign, deserialize
from urllib.request import urlopen
from urllib.parse import urlencode

url = "https://testnet.blockchain.info/"  ->비트코인 testnet url
A = 0   # 첫번째 Address
B = 1   # 두번째 Address              2개의 지갑주소 입력
address = ['mhJH61ScRnWJrhJm6283BbmACr27FjzT4Y',
'mg5urjMQZpALgga9GmwZaiiKKyBftxe7Mt']
privKey =
['7c06fcd8b6d7ef34182dd86882f5f1f1834381c132653b4f6937065167062b10',
'8126b6e1b33432198cbf9139174b470285b085fd4a0ccb6cc439f31cdbf3d298']

                2개의 개인키 입력
```

- **from bitcoin.bci import history**
- **from bitcoin.transaction**
  import mktx, sign, deserialize
  :비트코인 라이브러리에서
  비트코인 송금 시 필요한 모듈을 가져옴

- **from urllib.request import urlopen**
- **from urllib,parse import urlencode**
  : urllib는 url 처리 모듈
  : urlib.request는 url을 열고 읽기 제공
  : urllib.parse는 url 구문 분석 제공

비트코인 실습 – transaction (비트코인 testnet에서)

■ 2. 각 계좌의 UTXO 조회하기

```
def getUtxo(n=A):
    if n == A or n == B:
        h = history(address[n])
        return list(filter(lambda txo: 'spend' not in txo, h))
    else:
        print("address error.")
```

- **코드 실행 후, getUtxo(A)입력**

```
>>> getUtxo(A)
[]
>>> getUtxo(B)
[]
```

->아직 계좌 잔액이 없음

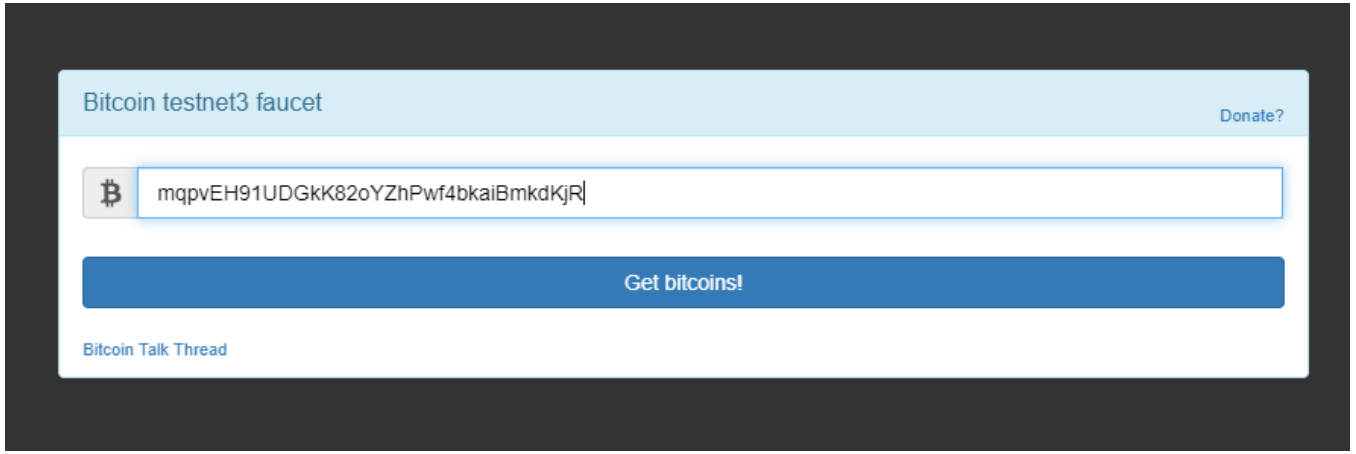
- **history(address[n])**
  :n의 주소의 모든 transaction output(txo)을 불러옴
- **lambda txo: 'spend' not in txo**
  : lambda variables: expression
  : 일시적으로 간단한 기능을 정의해서 사용하는 함수
  : txo 중에서 'spend'가 쓰여지 않은 txo,
  즉 아직 사용하지 않은 비트코인에 대해서 true를 반환
- **list(filter(lambda txo: 'spend' not in txo, h))**
  : 해당 계좌의 txo중에서 utxo만 필터링해서
  list형으로 표현

비트코인 실습 – transaction (비트코인 testnet에서)

■ 3. 비트코인 faucet 사이트를 통해 첫번째 계좌에 소량의 비트코인을 받기

<비트코인 testnet faucet 사이트>

- https://coinfaucet.eu/en/btc-testnet/
- https://testnet-faucet.mempool.co/
- https://bitcoinfaucet.uo1.net/
- https://testnet.help/en/btcfaucet/testnet
- https://kuttler.eu/en/bitcoin/btc/faucet/
- http://tpfaucet.appspot.com/
- https://tbtc.bitaps.com/



비트코인 실습 – transaction (비트코인 testnet에서)

■ 5. 일정 시간이 흐른 뒤 다시 UTXO 조회 – getUtxo(A)

```
>>> getUtxo(A)
[{'address': 'mqpvEH91UDGkK82oYZhPwf4bkaiBmkdKjR', 'value': 1000000, 'output':
'f445be424b3e8799a84c76d307ad66431cc8c88c532172b117336774cd9d4636:0',
'block_height': 1453413}]
```

- **address**

  :비트코인을 송금한 계좌 주소

- **Value**

  :비트코인 값, satoshi 단위 (1 BTC = 10,000,000 satoshi)

- **Output**

  :Transaction의 해시값과 output 번호(여기서 0은 해당 transaction output들 중 첫번째에 해당)

- **Block_height**

  : 해당 transaction이 포함된 블록의 높이(번호)

비트코인 실습 – transaction (비트코인 testnet에서)

■ 6. 첫번째 계좌에서 두번째 계좌로 송금하기 위한 transaction 생성하기(1)

```
def createTx(utxo, n1, n2, value, fee):
    # Input을 생성
    totalValue = 0
    inputs = []
    for i in range(len(utxo)):
        totalValue += utxo[i]['value'] * 1e-8
        inputs.append(utxo[i])
        if totalValue > (value + fee):
            break

    # 수수료를 차감한 거스름돈을 계산
    outChange = totalValue - value - fee
    if outChange < 0:
        raise ValueError("Value is larger than input amount")

    if outChange > 0:
        chgSatoshi = int(outChange * 1e8)
        change = [{'value': chgSatoshi, 'address': address[n1]}]

    else:
        change = []
```

- **def createTx(utxo, n1, n2, value, fee)**

  : 여기서 utxo는 송금 시 사용하는 utxo 목록,

  n1는 송금하는 사용자, n2는 송금 받는 사용자,

  value는 송금금액, fee는 수수료

- **for i in range(len(utxo))**

  : 1부터 utxo의 개수 만큼 해당 내용을 반복하여 실행

- **totalValue += utxo[i]['value']* 1e-8**

- **inputs.append(utxo[i])**

  : i번째 utxo의 'value' 값에 1e-8를 곱함(BTC로 환산)

  : utxo의 값을 totalValue 값에 더함

  : append()를 사용하여 utxo를 input에 list형으로 붙임

  : 한 utxo로 송금할 수 없는 큰 금액이면

  다른 utxo를 여러 개 붙여서 송금

비트코인 실습 – transaction (비트코인 testnet에서)

■ 6. 첫번째 계좌에서 두번째 계좌로 송금하기 위한 transaction 생성하기(1)

```
def createTx(utxo, n1, n2, value, fee):
    # Input을 생성
    totalValue = 0
    inputs = []
    for i in range(len(utxo)):
        totalValue += utxo[i]['value'] * 1e-8
        inputs.append(utxo[i])
        if totalValue > (value + fee):
            break

    # 수수료를 차감한 거스름돈을 계산
    outChange = totalValue - value - fee
    if outChange < 0:
        raise ValueError("Value is larger than input amount")

    if outChange > 0:
        chgSatoshi = int(outChange * 1e8)
        change = [{'value': chgSatoshi, 'address': address[n1]}]

    else:
        change = []
```

- **If totalValue > (value + fee) break**

  : 만약 totalValue 값이 사용해야 하는 (value + fee) 값보다 크다면 멈춤

- **outChange = totalValue – value – fee**

  : 다시 돌려받을 거스름돈을 계산

- **If outChange < 0: raise ValueError(“msg”)**

  : 계산한 거스름돈이 0보다 작을 경우 에러 출력

- **If outChange > 0:**

  : 계산한 거스름돈이 0보다 클 경우 통과

비트코인 실습 – transaction (비트코인 testnet에서)

■ 6. 첫번째 계좌에서 두번째 계좌로 송금하기 위한 transaction 생성하기(1)

```
def createTx(utxo, n1, n2, value, fee):
    # Input을 생성
    totalValue = 0
    inputs = []
    for i in range(len(utxo)):
        totalValue += utxo[i]['value'] * 1e-8
        inputs.append(utxo[i])
        if totalValue > (value + fee):
            break

    # 수수료를 차감한 거스름돈을 계산
    outChange = totalValue - value - fee
    if outChange < 0:
        raise ValueError("Value is larger than input amount")

    if outChange > 0:
        chgSatoshi = int(outChange * 1e8)
        change = [{'value': chgSatoshi, 'address': address[n1]}]

    else:
        change = []
```

- **chgSatoshi = int(outChange*1e8)**
  : 거스름돈을 BTC로 환산 후 int형으로 변환
- **change = [{‘value’:chgSatoshi, ‘address’: address[n1]}]**
  : 거스름돈을 받기 위한 output 생성
  : 거스름돈 output을 따로 생성하지 않을 경우,
  송금 후 남은 돈은 모두 채굴자가 수수료로 가져 감
- **else: change = []**
  : 거스름돈이 없는 경우 change는 비워 둠

비트코인 실습 – transaction (비트코인 testnet에서)

■ 6. 첫번째 계좌에서 두번째 계좌로 송금하기 위한 transaction 생성하기(2)

```
# Output을 생성
   outputs = [{'value': int(value * 1e8), 'address': address[n2]}]

# transaction을 만듦
   tx = mktx(inputs,outputs+change)

   return tx, len(inputs)
```

- **outputs = [{'value': int(value*1e8)}, 'address': address[n2]]**
  : 앞에서 계산 value값, 받는 사람의 계좌 주소를 이용하여 output을 생성함
- **Tx = mktx(inputs,outputs+change)**
  : mktx(input,output)를 이용하여 transaction을 생성
  : input에는 앞에서 구한 사용할 utxo 목록인 inputs 입력
  : output에는 outputs와 change를 더해서 입력
- **return tx, len(inputs)**
  : 생성한 transaction과 input의 개수를 반환
  : 이때 input 길이는 서명할 때 사용

비트코인 실습 – transaction (비트코인 testnet에서)

■ 7. transaction에 첫번째 계좌의 개인키로 서명

```
def signTx(tx, nInput, n1):
    for i in range(nInput):
        tx = sign(tx, i, privKey[n1])
    return tx
```

- **def signTx(tx, nInput, n1)**
  : tx와 nInput에는 이전에 반환한 transaction과 input의 수, n1은 서명하는 사용자(송금하는 사람)을 입력
- **for I in range(nInput)**
  : 1부터 input의 수만큼 아래 내용을 반복
- **sign(tx, I, privKey[n1])**
  : tx는 서명하고자 하는 transaction, i는 input index, privKey[n1]에는 서명에 사용하는 개인키 입력
- **return tx**
  : 서명된 transaction을 반환

비트코인 실습 – transaction (비트코인 testnet에서)

■ 8. 콘솔창에서 확인

```
>>> utxo = getUtxo(A)

>>> tx,ninput = createTx(utxo, A, B, 0.001, 0.0001)

>>> tx = signTx(tx, ninput, A)

>>> tx
'0100000000136469dcd74673317b17221538cc8c81c4366ad07d3764ca899873e4b42be45f40
00000008b483045022100d9ae9147e3f91ad8d0fffaf8d0a24e2d94a8f66130239ff58ac916e0b
fe6f6ac02202736adc13909742dec1779f1d9062649dd8076947292ed0efe1e3b84b8cff5fe014
104bc71f79b09bce42a4203c304cda7748aeffec3416eb3e2e5ead46f2ca1568aa3012a0df381b
5de0a2ddfdbe60dcb609683c90a0637469aa9d72bf0edf8bc9238ffffffff02a086010000000000
1976a914d8100310a1b0b7570f48b7e4fdedd1a1de9c532188ac90940d00000000001976a91
47117057423687a091cb3e7f338318e19f268986f88ac00000000'
```

-> 전자서명까지 완료된 transaction 생성 완료

비트코인 실습 – transaction (비트코인 testnet에서)

- 9. 생성된 transaction 디코딩해보기
- https://live.blockcypher.com/btc/decodetx/

Input New Transaction Hex

Transaction Hex*

```
01000000136469dcd74673317b17221538cc8c81c4366ad07d3764ca899873e4b42be45f4000000008b483045022100d9ae9147e3f91ad8d0fffaf8d0a24e2d
94a8f66130239ff58ac916e0bfe6f6ac02202736adc13909742dec1779f1d9062649dd8076947292ed0efe1e3b84b8cff5fe014104bc71f79b09bce42a4203c304
cda7748aeffec3416eb3e2e5ead46f2ca1568aa3012a0df381b5de0a2ddfdbe60dcb609683c90a0637469aa9d72bf0edf8bc9238ffffffff02a08601000000000019
76a914d8100310a1b0b7570f48b7e4fdedd1a1de9c532188ac90940d00000000001976a9147117057423687a091cb3e7f338318e19f268986f88ac00000000
```

생성한 Tx Hex값 입력

Network*

Bitcoin Testnet

비트코인 실습 – transaction (비트코인 testnet에서)

■ 9. 생성된 transaction 디코딩해보기


```
{
    "addresses": [
        "n1DPS2A3daQrTdDCeTZC5KmymVyo5jDPGR",
        "mqpvEH91UDGkK82oYZhPwf4bkaiBmkdKjR"
    ],
    "block_height": -1,
    "block_index": -1,
    "confirmations": 0,
    "double_spend": false,
    "fees": 10000,
    "hash": "89bcece67a9ed3944bb19cc23ddb2d79a2bc2ac11cc95ed782232bda1f5b22c1",
    "inputs": [
        {
            "addresses": [
                "mqpvEH91UDGkK82oYZhPwf4bkaiBmkdKjR"
            ],
            "age": 1489020,
            "output_index": 0,
            "output_value": 1000000,
            "prev_hash": "f445be424b3e8799a84c76d307ad66431cc8c88c532172b117336774cd9d4636",
            "script": "48304502210d9ae9147e3f91ad8d0fffaf8d0a24e2d94a8f66130239ff58ac916e0bfe6f6ac02202736adc13909742dec1779f1d9062649dd8076947292ed0efe1e3b84b8cff5fe014104bc71f79b09bce
            "script_type": "pay-to-pubkey-hash",
            "sequence": 4294967295
        }
    ],
    "outputs": [
        {
            "addresses": [
                "n1DPS2A3daQrTdDCeTZC5KmymVyo5jDPGR"
            ],
            "script": "76a914d8100310a1b0b7570f48b7e4fdedd1a1de9c532188ac",
            "script_type": "pay-to-pubkey-hash",
            "value": 100000
        },
        {
            "addresses": [
                "mqpvEH91UDGkK82oYZhPwf4bkaiBmkdKjR"
            ],
            "script": "76a91471170574236 87a091cb3e7f338318e19f268986f88ac",
            "script_type": "pay-to-pubkey-hash",
            "value": 890000
        }
    ],
    "preference": "low",
    "received": "2019-04-14T11:13:28.434861018Z",
    "relayed_by": "3.92.229.249",
    "size": 258,
    "total": 990000,
    "ver": 1,
    "vin_sz": 1,
    "vout_sz": 2
}
```

비트코인 실습 – transaction (비트코인 testnet에서)

■ 10. testnet.blockchain.info API 서버에 거래 전송 요청

```
def sendTx(tx):
    params = {'tx': tx}
    payload = urlencode(params).encode('utf-8')
    response = urlopen(url + 'pushtx', payload).read()
    print(response.decode('utf-8'))
```

- **params = {'tx': tx}**
- : dictionary 생성('tx'는 key 값, tx는 value로 정의)
- **urlencode(params).encode('utf-8')**
- :dictionary를 받아 이를 URL에 들어갈 수 있는 형태로 인코딩함
- **urlopen(url + 'pushtx', payload).read()**
- : 주어진 url을 열고 payload를 입력 후 , 바이트형으로 데이터를 읽어 옴
- **print(response.decode('utf-8))**
- : 읽을 수 있게 디코딩하고 출력함

비트코인 실습 – transaction (비트코인 testnet에서)

■ 11. 계좌 송금 함수 작성 – 앞에서 생성한 함수들 이용

```
def transaction(From, To, value, fee):
    utxo = getUtxo(nFrom)
    tx, nInput = makeTx(utxo, From, To, value, fee)
    tx = signTx(tx, nInput, From)
    sendTx(tx)
    return tx
```

비트코인 실습 – transaction (비트코인 testnet에서)

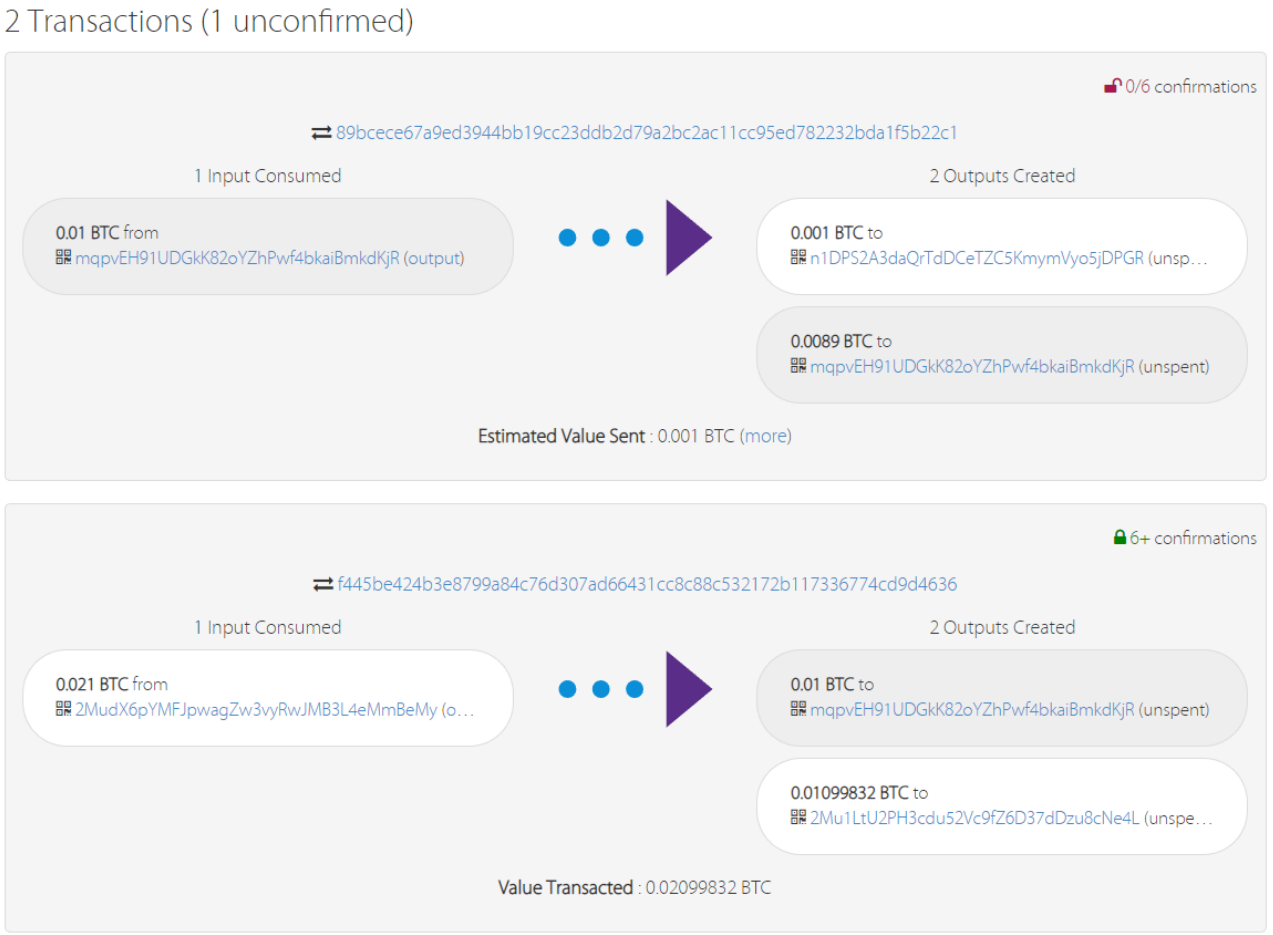
■ 12. 첫번째 계좌에서 두번째 계좌로 송금 실행

```
>>> transaction(A, B, 0.001, 0.0001)
Transaction Submitted  ->송금이 실행됨

'010000000136469dcd74673317b17221538cc8c81c4366ad07d3764ca899873e4b42
be45f4000000008b483045022100d9ae9147e3f91ad8d0fffaf8d0a24e2d94a8f661302
39ff58ac916e0bfe6f6ac02202736adc13909742dec1779f1d9062649dd8076947292e
d0efe1e3b84b8cff5fe014104bc71f79b09bce42a4203c304cda7748aeffec3416eb3e2
e5ead46f2ca1568aa3012a0df381b5de0a2ddfdbe60dcb609683c90a0637469aa9d72
bf0edf8bc9238ffffffff02a086010000000001976a914d8100310a1b0b7570f48b7e4fd
edd1a1de9c532188ac90940d000000000001976a9147117057423687a091cb3e7f3383
18e19f268986f88ac00000000'
```

비트코인 실습 – transaction (비트코인 testnet에서)

- 12. https://live.blockcypher.com/btc-testnet에서 다시 계좌 조회
- transaction 전송확인



비트코인 실습 – transaction (비트코인 testnet에서)

- 13. 일정 시간이 흐른 후, 첫번째 계좌의 UTXO를 조회하면
- transaction 전송확인

```
>>> getUtxo(A)
[{'address': 'mqpvEH91UDGkK82oYZhPwf4bkaiBmkdKjR', 'value': 890000, 'output':
'89bcece67a9ed3944bb19cc23ddb2d79a2bc2ac11cc95ed782232bda1f5b22c1:1',
'block_height': None}]
```

Transaction에서 output index

```
>>> getUtxo(B)
[{'address': 'n1DPS2A3daQrTdDCeTZC5KmymVyo5jDPGR', 'value': 100000, 'output':
'89bcece67a9ed3944bb19cc23ddb2d79a2bc2ac11cc95ed782232bda1f5b22c1:0',
'block_height': None}]
```

블록체인 실습 – 참고

- https://pypi.org/project/bitcoin/
- Pybitcointools, Python library for Bitcoin signatures and transactions

- privkey_to_pubkey : (privkey) -> pubkey
- privtopub : (privkey) -> pubkey
- pubkey_to_address : (pubkey) -> address
- pubtoaddr : (pubkey) -> address
- privkey_to_address : (privkey) -> address
- privtoaddr : (privkey) -> address
- add : (key1, key2) -> key1 + key2 (works on privkeys or pubkeys)
- multiply : (pubkey, privkey) -> returns pubkey * privkey
- ecdsa_sign : (message, privkey) -> sig
- ecdsa_verify : (message, sig, pubkey) -> True/False
- ecdsa_recover : (message, sig) -> pubkey
- random_key : () -> privkey
- random_electrum_seed : () -> electrum seed
- electrum_stretch : (seed) -> secret exponent
- electrum_privkey : (seed or secret exponent, i, type) -> privkey
- electrum_mpk : (seed or secret exponent) -> master public key
- electrum_pubkey : (seed or secexp or mpk) -> pubkey
- bip32_master_key : (seed) -> bip32 master key
- bip32_ckd : (private or public bip32 key, i) -> child key
- bip32_privtopub : (private bip32 key) -> public bip32 key
- bip32_extract_key : (private or public bip32_key) -> privkey or pubkey
- deserialize : (hex or bin transaction) -> JSON tx
- serialize : (JSON tx) -> hex or bin tx