PRACTICAL 4

[CS601] – Cryptography and Blockchain

Date – 20/03/2023 | *By* Aishwarya Suryakant Waghmare, PRN – 2001106059

Title/Aim of the practical :

To create a python code for the purpose of bitcoin scripting and thereby executing the same as well.

Apparatus/Tools/ Resources used :

- Lecture Notes
- E-Resources
- E-Book
- Laptop
- Google Colab
- PyCharm Community
- Jupyter Notebook

Theory of the practical :

- ✓ Bitcoin scripting refers to the scripting language used to create transactions on the Bitcoin network.
- ✓ It is a simple, stack-based language that allows users to create complex conditions for spending their Bitcoin.
- ✓ In cryptography, Bitcoin scripting uses public-key cryptography, where each user has a public key and a private key.
- ✓ A Bitcoin transaction includes a set of inputs that reference previous transactions, and a set of outputs that specify where the bitcoins are going.
- ✓ Each output is locked with a script that specifies the conditions under which it can be spent.
- ✓ The Bitcoin scripting language allows users to create a wide variety of conditions for spending their bitcoins, such as requiring multiple signatures, time-locks, and multi-party transactions.
- ✓ These conditions are enforced by the nodes on the Bitcoin network, which verify that the transaction meets the specified conditions before including it in the blockchain.
- ✓ Overall, Bitcoin scripting provides a flexible and secure way to transfer and manage bitcoins on the blockchain, and is a key component of the Bitcoin protocol's success.

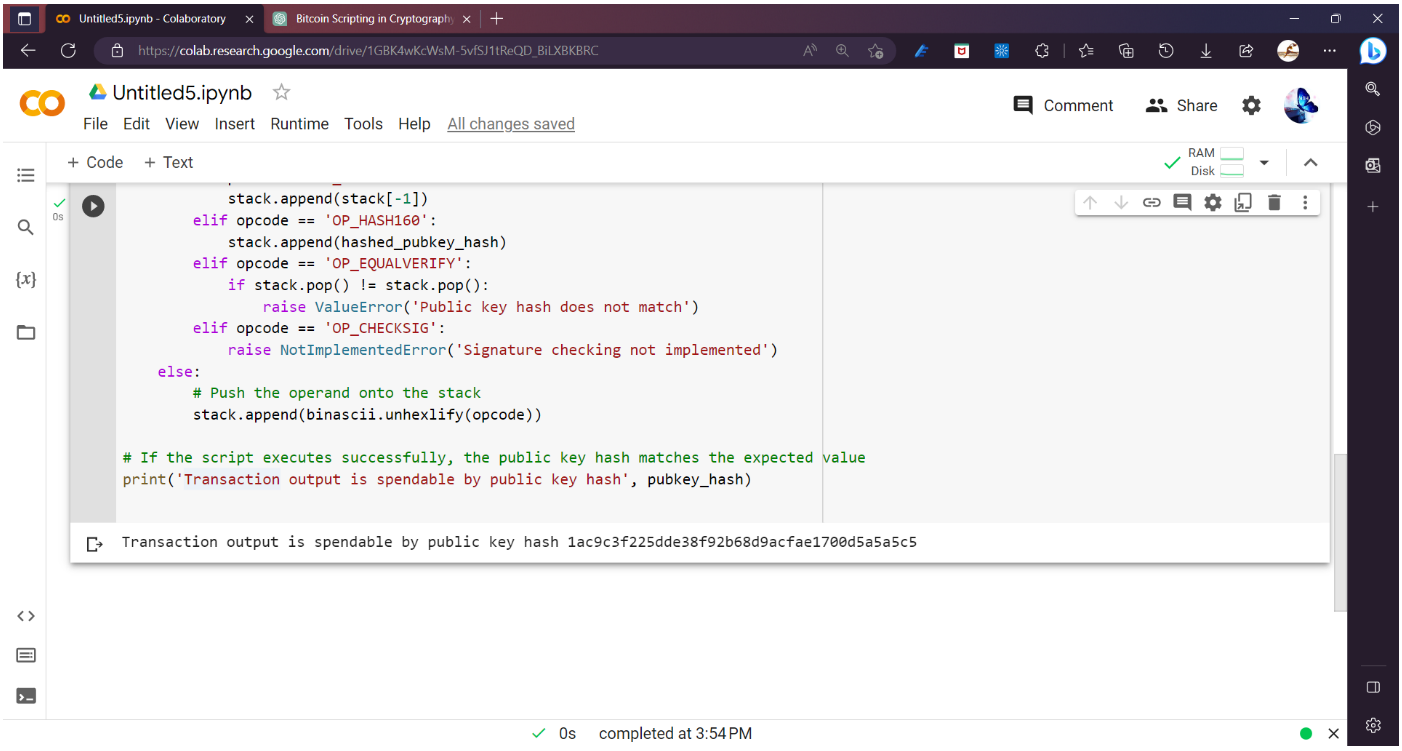
Procedure of the Practical/ Codes:

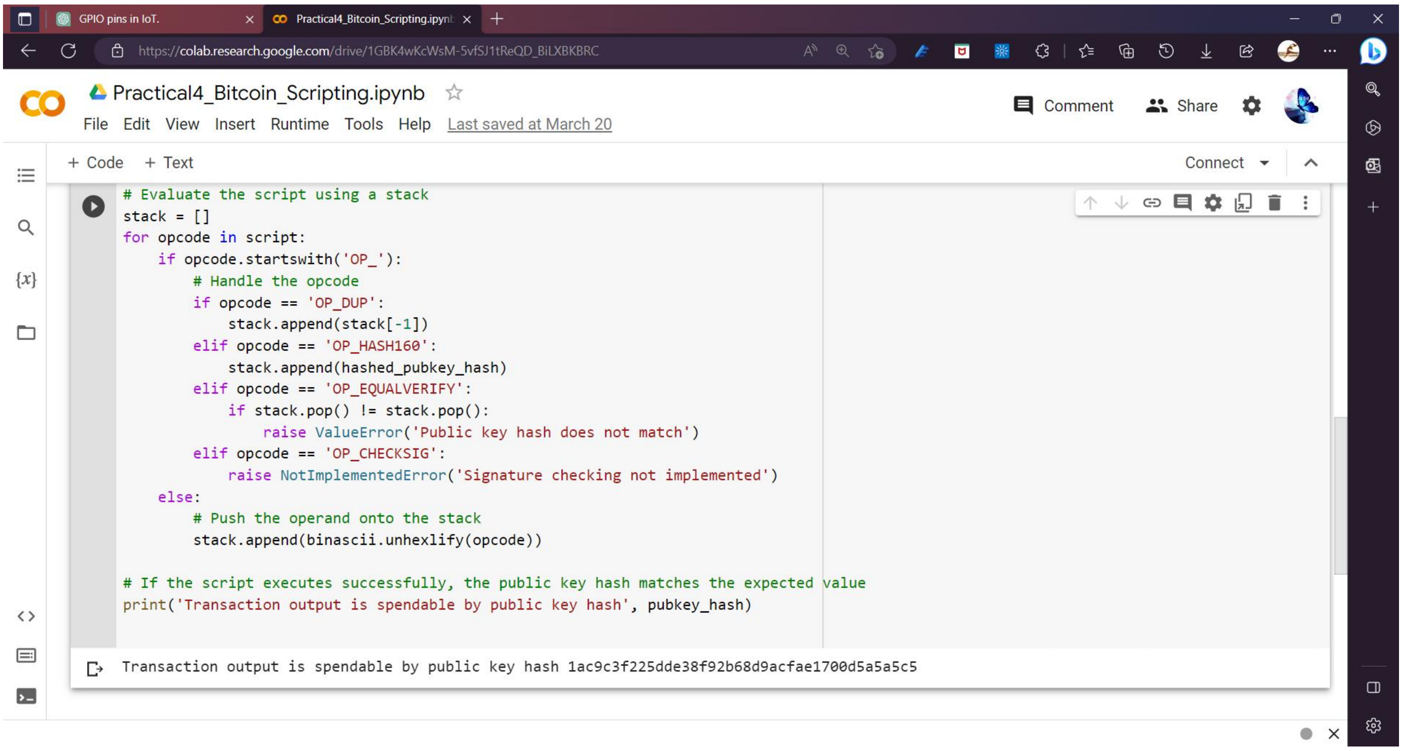
```
import hashlib
import binascii


# Define the script as a list of opcodes and operands
script = [
    '76',       # OP_DUP
    'a9',       # OP_HASH160
    '14',       # Push 20 bytes
```

```
    'd9', '2b', '20', '01', 'fc', '02', '91', '8e', '33', '6a', '3e', '1f', '25', 'b3', '01', 'dc', 'ca', '1a',
               # Public Key Hash in little-endian
    '88',        # OP_EQUALVERIFY
    'ac'         # OP_CHECKSIG
]

# Define the transaction output to be verified
tx_output = {
    'value': 100000000,              # Value in satoshis (1 BTC = 100 million satoshis)
    'scriptPubKey': ''.join(script)  # Script as a string of hexadecimal opcodes and operands
}

# Define the public key hash that should be able to spend the transaction output
pubkey_hash = '1ac9c3f225dde38f92b68d9acfae1700d5a5a5c5'

# Convert the public key hash from hexadecimal to bytes
pubkey_hash_bytes = binascii.unhexlify(pubkey_hash)

# Hash the public key hash with SHA-256 followed by RIPEMD-160
hashed_pubkey_hash = hashlib.new('ripemd160', hashlib.sha256(pubkey_hash_bytes).digest()).digest()

# Convert the hashed public key hash to a string of hexadecimal characters in little-endian order
hashed_pubkey_hash_hex = binascii.hexlify(hashed_pubkey_hash[::-1]).decode('utf-8')

# Replace the <Public Key Hash> operand in the script with the hashed public key hash
script[3] = hashed_pubkey_hash_hex

# Evaluate the script using a stack
stack = []
for opcode in script:
    if opcode.startswith('OP_'):
        # Handle the opcode
        if opcode == 'OP_DUP':
            stack.append(stack[-1])
        elif opcode == 'OP_HASH160':
            stack.append(hashed_pubkey_hash)
        elif opcode == 'OP_EQUALVERIFY':
```

```
        if stack.pop() != stack.pop():
            raise ValueError('Public key hash does not match')
    elif opcode == 'OP_CHECKSIG':
        raise NotImplementedError('Signature checking not implemented')
  else:
    # Push the operand onto the stack
    stack.append(binascii.unhexlify(opcode))

# If the script executes successfully, the public key hash matches the expected value
print('Transaction output is spendable by public key hash', pubkey_hash)
```

Result/ Output/ Screenshots of the Practical :





```python
# Evaluate the script using a stack
stack = []
for opcode in script:
    if opcode.startswith('OP_'):
        # Handle the opcode
        if opcode == 'OP_DUP':
            stack.append(stack[-1])
        elif opcode == 'OP_HASH160':
            stack.append(hashed_pubkey_hash)
        elif opcode == 'OP_EQUALVERIFY':
            if stack.pop() != stack.pop():
                raise ValueError('Public key hash does not match')
        elif opcode == 'OP_CHECKSIG':
            raise NotImplementedError('Signature checking not implemented')
    else:
        # Push the operand onto the stack
        stack.append(binascii.unhexlify(opcode))

# If the script executes successfully, the public key hash matches the expected value
print('Transaction output is spendable by public key hash', pubkey_hash)
```

```
Transaction output is spendable by public key hash 1ac9c3f225dde38f92b68d9acfae1700d5a5a5c5
```

Parameters achieved/ Conclusion :

Therefore, understood – implemented and created a python code for the bitcoin scripting as well successful execution of the same as well.