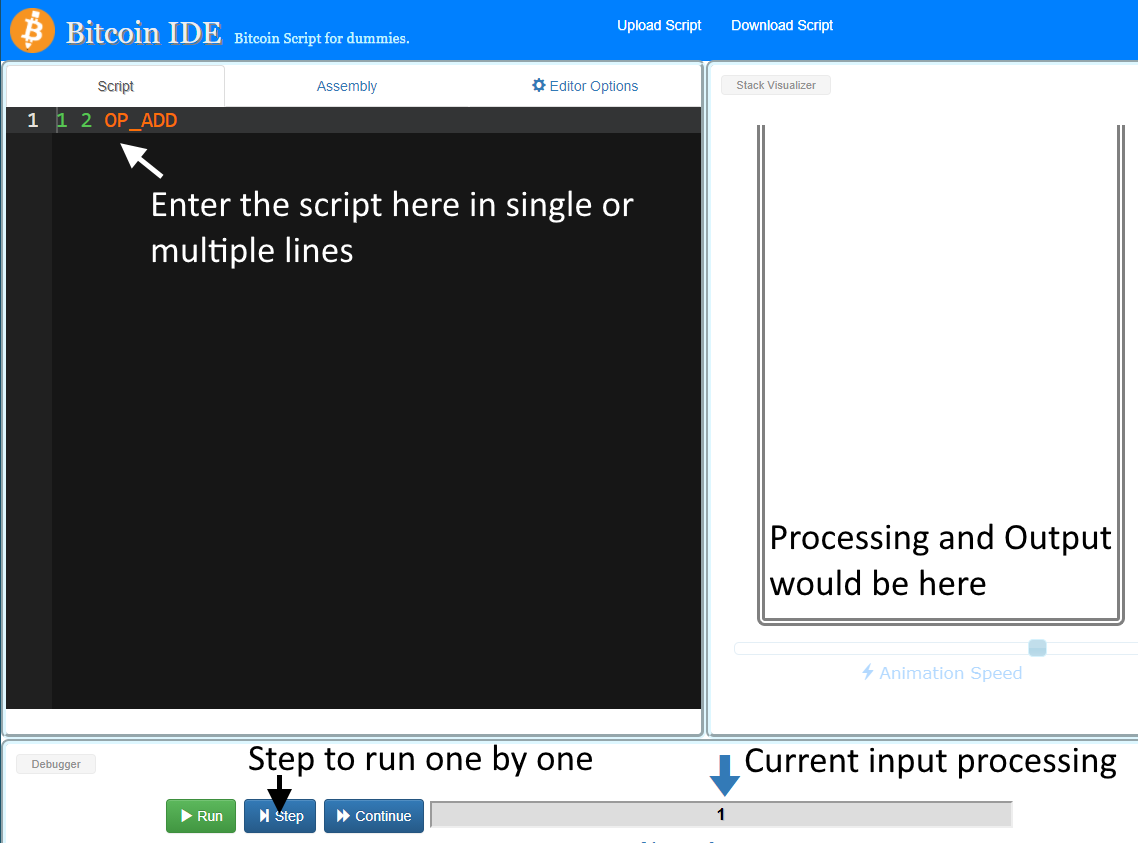
**Activity: Bitcoin Script Simulator**

We are going to use a simulator to test out some scripting in Bitcoin.

Open the browser at:

<https://siminchen.github.io/bitcoinIDE/build/editor.html>

The simulator can perform simple operations only. It can’t actually verify signatures or hash since we didn’t provide the original transactions, etc.



1. **Simple Script**

Try these script

1. 2 5 OP\_ADD 3 OP\_ADD 3 OP\_ADD 7 OP\_EQUAL
2. 2 5 OP\_ADD 3 OP\_ADD 3 OP\_SUB 7 OP\_EQUAL
3. 2 3 4 1 OP\_MUL OP\_ADD OP\_SUB

Explain the results.

1. **P2PK**

A P2PK lock just contains a public key and a CHECKSIG opcode:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/lock.png scriptPubKey** | 04ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84c OP\_CHECKSIG |  |

To unlock it you just need to provide a valid signature:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/key.png scriptSig** | 30440220576497b7e6f9b553c0aba0d8929432550e092db9c130aae37b84b545e7f4a36c022066cb982ed80608372c139d7bb9af335423d5280350fe3e06bd510e695480914f01 |  |

When the script runs, the CHECKSIG opcode compares the signature against the public key, and pushes a 1 on to the stack if it is valid.

Copy the content of scriptSig and scriptPubKey into the Simulator. Press Step to observe the program execution. The locking script scriptPubKey should precede the unlocking scriptSig.

Note: The simulator won’t actually be able to verify the signature.

1. **P2PKH**

The P2PKH script pattern contains a hashed public key surrounded by these opcodes:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/lock.png scriptPubKey** | OP\_DUP OP\_HASH160 12ab8dc588ca9d5787dde7eb29569da63c3a238c OP\_EQUALVERIFY OP\_CHECKSIG | P2PKH |

To solve this script, the owner of the hashed public key above needs to provide the original public key, along with a valid signature for it:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/key.png scriptSig** | 304502203f004eeed0cef2715643e2f25a27a28f3c578e94c7f0f6a4df104e7d163f7f8f022100b8b248c1cfd8f77a0365107a9511d759b7544d979dd152a955c867afac0ef78601 044d05240cfbd8a2786eda9dadd520c1609b8593ff8641018d57703d02ba687cf2f187f0cee2221c3afb1b5ff7888caced2423916b61444666ca1216f26181398c |  |

In short, when this script runs:

* The original public key is DUPlicated and then HASH160’ed.
* This hashed value is compared with the hashed public key in the scriptPubKey to make sure it is EQUALVERIFY.
* If it matches, the script continues and the CHECKSIG checks the signature against the public key (just like a P2PK script).

1. **P2MS**

Multisig scripts are pretty straightforward to create. For the locking script:

1. Include an opcode M to indicate how many signatures are required.
2. Include the public keys.
3. Include another opcode N to indicate how many public keys there are.
4. Put the CHECKMULTISIG opcode at the end.

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/lock.png scriptPubKey** | OP\_1 04cc71eb30d653c0c3163990c47b976f3fb3f37cccdcbedb169a1dfef58bbfbfaff7d8a473e7e2e6d317b87bafe8bde97e3cf8f065dec022b51d11fcdd0d348ac4 0461cbdcc5409fb4b4d42b51d33381354d80e550078cb532a34bfa2fcfdeb7d76519aecc62770f5b0e4ef8551946d8a540911abe3e7854a26f39f58b25c15342af OP\_2 OP\_CHECKMULTISIG | P2MS |

To unlock a P2MS script, you just need to provide the required number of signatures. In this case M is **one**:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/key.png scriptSig** | OP\_0 304402203f16c6f40162ab686621ef3000b04e75418a0c0cb2d8aebeac894ae360ac1e780220ddc15ecdfc3507ac48e1681a33eb60996631bf6bf5bc0a0682c4db743ce7ca2b01 |  |

When this script executes, all of the signatures and public keys get pushed on to the stack.

Then we get to CHECKMULTISIG, which:

1. Pops off N, and then pops that number of public keys of the stack.
2. Pops off M, and then pops that number of signatures of the stack.

After popping all the public keys and signatures off the stack, it compares each signature with each public key:

* If the signature does not match the public key, we move on to the next public key and check that one. However, that public key will also be **ignored** for every subsequent signature (so make sure you put your signatures in order in the scriptSig!).
* If the signature matches the public key, we increment a tally and repeat for the next signature.

If the **tally of valid signatures is equal to**M after all of the signatures have been checked, then CHECKMULTISIG pushes a 1 on to the stack and the script is valid.

1. **P2SH**

The locking script contains the hash of your custom locking script (the “script hash”), surrounded by the HASH160 and EQUAL opcodes:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/lock.png scriptPubKey** | OP\_HASH160 748284390f9e263a4b766a75d0633c50426eb875 OP\_EQUAL | P2SH |

The unlocking script then contains your original custom locking script (the “redeem script”), preceded by the data/opcodes needed to unlock it:

|  |  |  |
| --- | --- | --- |
| **https://learnmeabitcoin.com/common/images/icons/key.png scriptSig** | OP\_0 3046022100a07b2821f96658c938fa9c68950af0e69f3b2ce5f8258b3a6ad254d4bc73e11e022100e82fab8df3f7e7a28e91b3609f91e8ebf663af3a4dc2fd2abd954301a5da67e701 5121022afc20bf379bc96a2f4e9e63ffceb8652b2b6a097f63fbee6ecec2a49a48010e2103a767c7221e9f15f870f1ad9311f5ab937d79fcaeee15bb2c722bca515581b4c052ae |  |

So in the unlocking script above, the redeem script is a P2MS locking script itself, and the preceding signature is what’s needed to unlock it