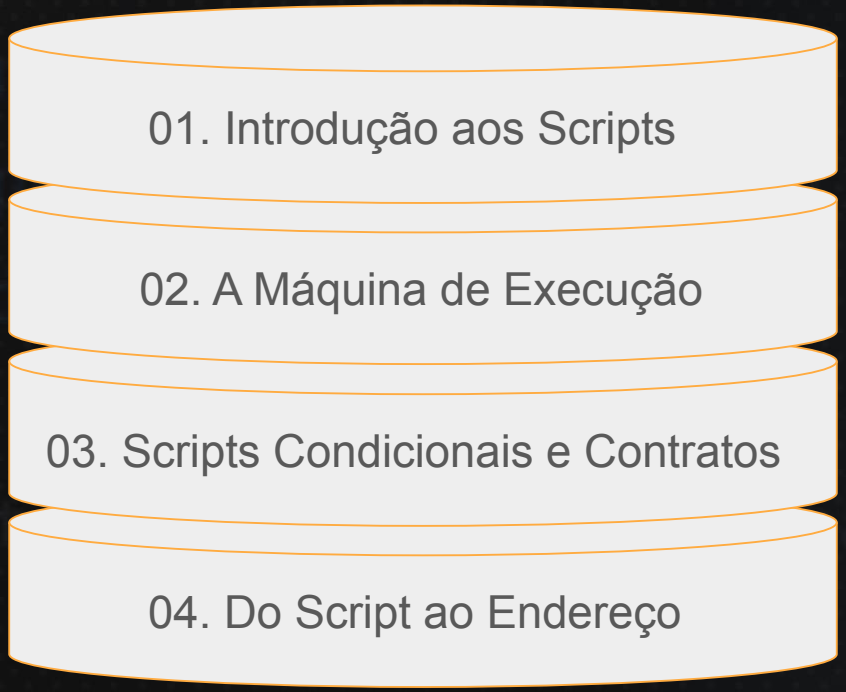
Scripts:

como o Bitcoin executa suas regras

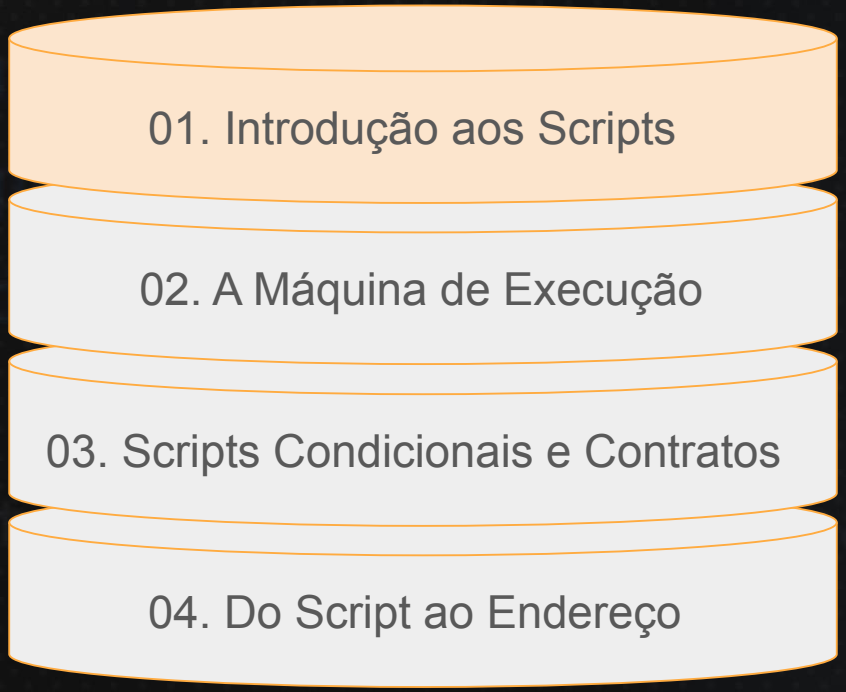
Anfitrião:
Rafael Penna

bitcoinCoders bitups

O Que Veremos!



O Que Veremos!



01. Introdução aos Scripts

- O Bitcoin Script é uma linguagem mínima, baseada em pilha, onde cada instrução (opcode) pode:
  - Empilhar dados
  - Desempilhar dados
  - Comparar dados
  - Verificar condições

- Não há:
  - Variáveis
  - Funções
  - Loops

01. Introdução aos Scripts

- Os scripts aparecem em dois papéis complementares:
  - output da transação
    - scriptPubKey (locking script)
  - input da transação
    - scriptSig (unlocking script)
- Na validação, o nó executa “scriptSig seguido de scriptPubKey”
  - Dados do scriptSig entram na pilha
  - scriptPubKey aplica operações com OPCodes
  - Consumindo e comparando elementos do topo da pilha
  - Se, ao final, [True] → gasto válido
  - Caso contrário → transação é rejeitada

01. Introdução aos Scripts

- Chaves Públicas e Privadas e Assinaturas
  - Chave privada → segredo absoluto do usuário
  - Assinatura digital exclusiva
  - Chave pública → derivada da privada

- Quando uma transação é criada:
  - scriptSig → assinatura + chave pública
  - scriptPubKey → instruções que usam a assinatura e a chave pública para confirmar que a mensagem (a transação) foi realmente assinada pelo dono da chave privada.

01. Introdução aos Scripts

- Validação da Transação
  - P2PKH
    - scriptPubKey
      - OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
      - “Para gastar este output, apresente uma chave pública que produza este pubKeyHash, e uma assinatura válida para essa chave.”
    - scriptSig
      - <assinatura> <chave_publica>

01. Introdução aos Scripts

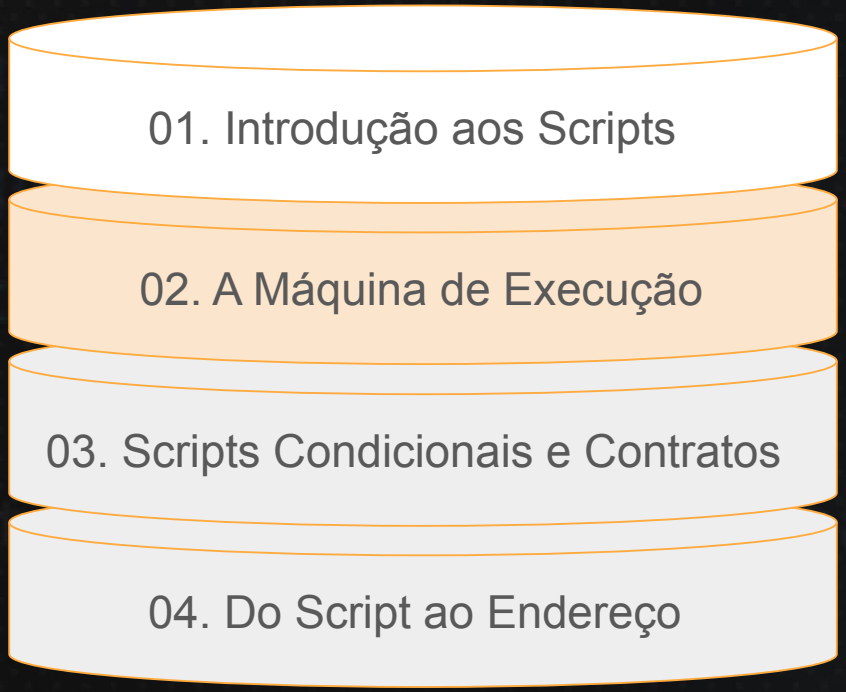
- Validação da Transação
  - <assinatura> <chave_publica> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

Etapa	Operação	Pilha após execução
1	Empilha <assinatura>	[assinatura]
2	Empilha <chave_publica>	[assinatura, chave_publica]
3	OP_DUP duplica o topo	[assinatura, chave_publica, chave_publica]
4	OP_HASH160 aplica hash no topo	[assinatura, chave_publica, hash(chave_publica)]
5	Empilha <pubKeyHash> (o destino esperado)	[assinatura, chave_publica, hash(chave_publica), pubKeyHash]
6	OP_EQUALVERIFY compara os dois últimos elementos → se iguais, remove ambos e continua	[assinatura, chave_publica]
7	OP_CHECKSIG verifica os dois últimos elementos da pilha, confirmando se a assinatura é válida para aquela chave pública e para a transação.	[TRUE]

01. Introdução aos Scripts

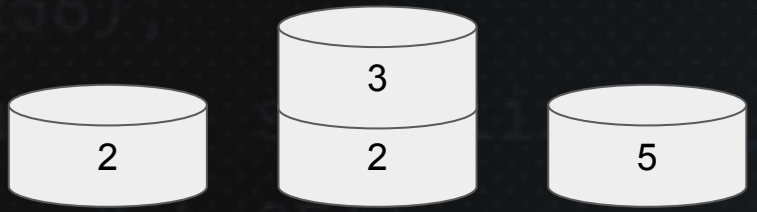
- Exemplo Prático com bitcoin-cli
  - bitcoin-cli -datadir="." decoderawtransaction 0200000001d49df01aae855b7da49b33ca2e5b9ac821ab948f129b57fe4653a548dcd19c16000000006a473044022051b11fb2c1e2b81099b09cb4410e1379612abfff5ee419467850dd8269c687dc02204bebd83e2d32ad268a9f4976e2b43364e62f0805acecc57b0f6cade86665650c0121022cb7fcfb1377e85e3c5ac300028890059ff84e9d8e1b6f35a808ebf3e477e7f1fdffffff0194110000000000001976a914aa0420f7f934bce88e817ccaa33f06208f10681588ac00000000

O Que Veremos!

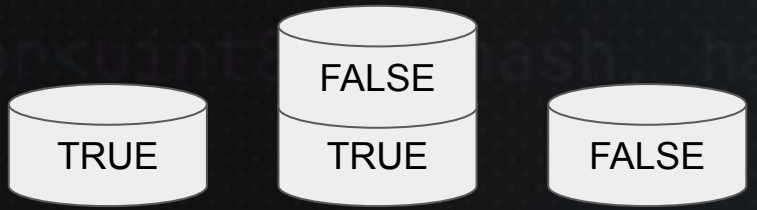


02. A Máquina de Execução

- Exemplo Aritmético
  - OP_2 OP_3 OP_ADD

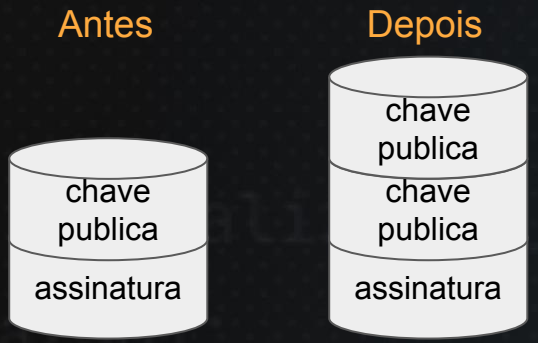


- Exemplo Lógico
  - OP_TRUE OP_FALSE OP_BOOLAND



02. A Máquina de Execução

- OPCodes - Manipulação de Pilha
  - OP_DUP — duplica o topo



  - OP_DROP — remove o topo

02. A Máquina de Execução

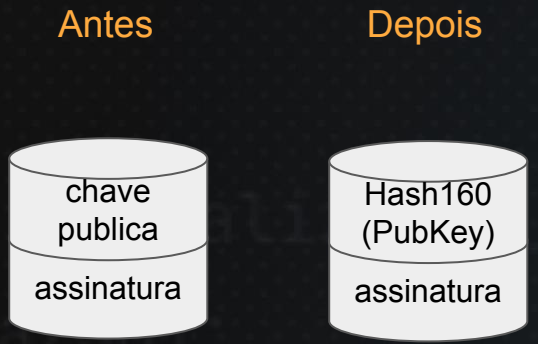
- OPCodes - Manipulação de Pilha
  - OP_SWAP — troca os dois do topo

  - OP_OVER - copia o segundo elemento para o topo

02. A Máquina de Execução

- OPCodes - Validação
  - OP_HASH160

aplica SHA256 depois RIPEMD160 no topo



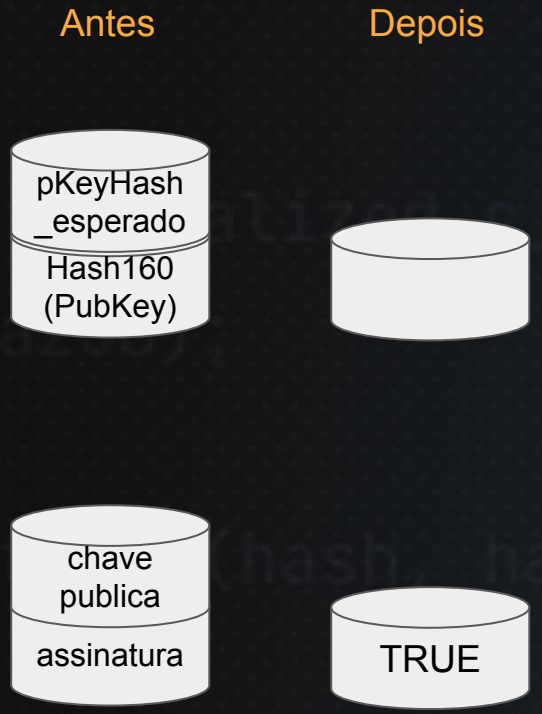
  - OP_EQUAL

compara os dois do topo, retira os 2 elementos e empilha TRUE/FALSE

02. A Máquina de Execução

- OPCodes - Validação
  - OP_EQUALVERIFY

igual ao OP_EQUAL,
mas falha se for FALSE



  - OP_CHECKSIG

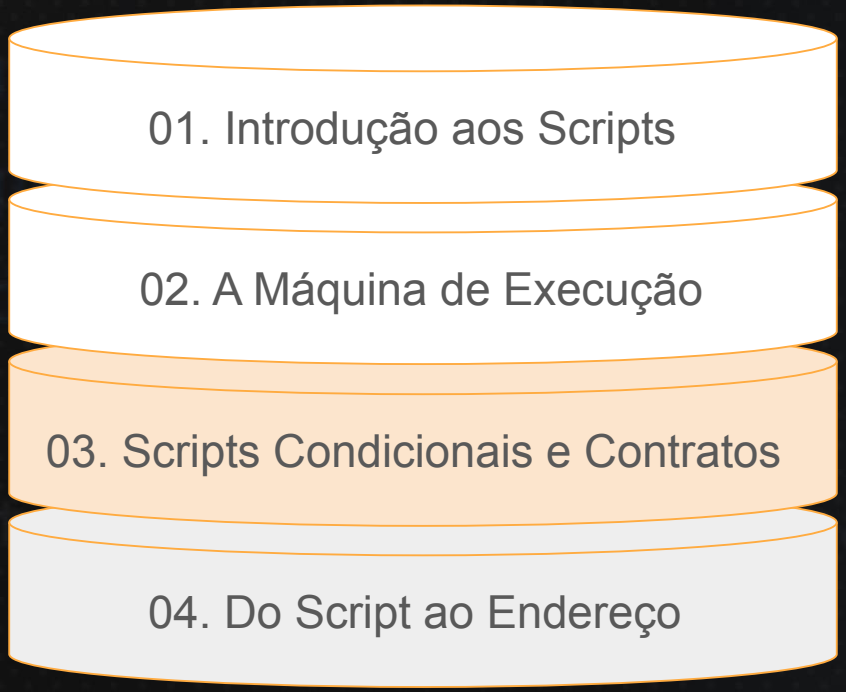
consome assinatura e chave pública;
verifica assinatura da tx

02. A Máquina de Execução

- Simuladores
  - https://siminchen.github.io/bitcoinIDE

- Exemplos:
  - OP_2 OP_3 OP_ADD 5 OP_EQUAL
  - 17 17 OP_OVER OP_EQUAL
  - 022cb7fcfb1377e85e3c5ac300028890059ff84e9d8e1b6f35a808ebf3e477e7f1 OP_DUP OP_HASH160 AD28C4C54C4084A74E944B8E60916910FB9089B3 OP_EQUALVERIFY

O Que Veremos!



03. Scripts Condicionais e Contratos

- Bitcoin Script é expressivo o suficiente para descrever quem pode gastar, quando e sob quais condições.

- Instruções simples (opcodes) podem criar combinações lógicas:
  - “se X e Y assinarem, o gasto é válido”;
  - “se o prazo ainda não passou, use esta regra; senão, use outra”
  - “só aceite a transação se a altura do bloco for maior que N”

03. Scripts Condicionais e Contratos

- bitcoin-cli -datadir="." decodescript "6351670068"

{ "asm": "OP_IF 1 OP_ELSE 0 OP_ENDIF",
….}

```
SE (condição verdadeira)
        empilha TRUE
SENÃO
        empilha FALSE
FIM
```

Hex	Opcode	Função
63	**OP_IF**	Verifica o topo da pilha: se for diferente de zero (TRUE), executa o bloco seguinte até OP_ELSE; caso contrário, pula para depois de OP_ELSE.
51	**OP_TRUE**	Coloca o valor lógico TRUE (equivalente a 1) na pilha.
67	**OP_ELSE**	Define o início do bloco alternativo (executado se a condição for FALSE).
00	**OP_FALSE**	Coloca o valor lógico FALSE (equivalente a 0) na pilha.
68	**OP_ENDIF**	Encerra a estrutura condicional iniciada por OP_IF.

03. Scripts Condicionais e Contratos

- Multisig com OP_CHECKMULTISIG
  - Formato do script multisig

    <m> <pubkey1> <pubkey2> <pubkey3> <n> OP_CHECKMULTISIG

  - Onde:
    - <m> é o número mínimo de assinaturas exigidas;
    - <n> é o número total de chaves envolvidas;
    - OP_CHECKMULTISIG verifica se as assinaturas fornecidas realmente correspondem às chaves declaradas.

03. Scripts Condicionais e Contratos

- Exemplo prático com o Bitcoin Core - Multisig
- 1. Criar 3 novos endereços:
  bitcoin-cli -datadir="." getnewaddress
  # tb1qge4dzkxfphnuke75evgavsg5hzv8yepp5zcp27
  bitcoin-cli -datadir="." getnewaddress
  # tb1qcxy5ygmsj2fnrt082y09esdn5ya4k7jdwztdjn
  bitcoin-cli -datadir="." getnewaddress
  # tb1qdnf9w7u235vvqag3cfqzjupzsg2y3646fm2mpd
- 2. Obter as pubkey dos 3 endereços:
  bitcoin-cli -datadir="." getaddressinfo
  tb1qge4dzkxfphnuke75evgavsg5hzv8yepp5zcp27 { ..., "pubkey":
  "0276d4d259f156bc953231ae2365555c9a445d50f40e0f9d49f43b062a38df6
  213", ... } //fazer o mesmo para os outros 2

03. Scripts Condicionais e Contratos

- Exemplo prático com o Bitcoin Core - Multisig
- 3. Criar o script multisig (2-de-3):
  bitcoin-cli -datadir="." createmultisig 2
'["0276d4d259f156bc953231ae2365555c9a445d50f40e0f9d49f43b062a38df6213","020498177dced0022d4538f0b39a09d315ec944c7205307dc126af8f132975cae3","0202835b62db105222c0d5208b31189660bd8a2f636d25d91329f73be56282fe52"]'

{ "address": "2N2iqBAJvyKA2zzhHGUFPwbufqzNs3vGKJo",
"redeemScript":"52210276d4d259f156bc953231ae2365555c9a445d50f40e0f9d49f43b062a38df62132102049817 7dced0022d4538f0b39a09d315ec944c7205307dc126af8f132975cae3210202835b62db105222c0d5208b31189660bd8a2f636d25d91329f73be56282fe5253ae", … }

03. Scripts Condicionais e Contratos

- Exemplo prático com o Bitcoin Core - Multisig
- 4. Decodificar o script

```
bitcoin-cli -datadir="." decodescript
"52210276d4d259f156bc953231ae2365555c9a445d50f40e0f9d49f43b062a38df6
2132102049817 7dced0022d4538f0b39a09d315ec944c7205307dc126af8f132975
cae3210202835b62db105222c0d5208b31189660bd8a2f636d25d91329f73be562
82fe5253ae"

{ "asm": "2
0276d4d259f156bc953231ae2365555c9a445d50f40e0f9d49f43b062a38df6213
020498177dced0022d4538f0b39a09d315ec944c7205307dc126af8f132975cae3
0202835b62db105222c0d5208b31189660bd8a2f636d25d91329f73be56282fe52
3 OP_CHECKMULTISIG",...}
```

OP_2 <pubkey1> <pubkey2> <pubkey3> OP_3 OP_CHECKMULTISIG

bitcoinCoders bitups

03. Scripts Condicionais e Contratos

- Vendo a execução na pilha - Multisig

ScriptSig (chave):

`OP_0 <assinatura1> <assinatura2>`

RedeemScript (cadeado):

`OP_2 <pubkey1> <pubkey2> <pubkey3> OP_3 OP_CHECKMULTISIG`

Etapa	Operação executada	Estado da pilha	Observação
1	Empilha `OP_0` (dummy) e as duas assinaturas	`[OP_0, assinatura1, assinatura2]`	O `OP_0` é necessário por um bug histórico.
2	Empilha `OP_2` (número de assinaturas requeridas)	`[OP_0, assinatura1, assinatura2, 2]`	O valor 2 indica que duas assinaturas devem ser válidas.
3	Empilha três chaves públicas e `OP_3`	`[OP_0, assinatura1, assinatura2, 2, pubkey1, pubkey2, pubkey3, 3]`	Agora há 3 chaves públicas e o total de chaves n = 3.
4	Executa `OP_CHECKMULTISIG`	`[TRUE]`	Opcode verifica se pelo menos 2 assinaturas são válidas para as 3 chaves. Retorna TRUE se sim.

03. Scripts Condicionais e Contratos

- Script dual
  - Contrato Bitcoin completo, que representa duas possibilidades de gasto, controladas por tempo e assinatura
    - Antes do bloco 273100 → B pode gastar.
    - Depois do bloco 273100 → A também pode gastar.
  - Estrutura do script

```
OP_IF
    036ae41e66b25bab55932a82b0f523d078158c98a0a0d60326bc6a83e370bbe35f
    OP_CHECKSIGVERIFY 273100 OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_ELSE
    0307b4825b0ad5ce1dd42b0cf2d9add5305e07d9eb76105f7bc130c0cac5bce70f
    OP_CHECKSIG
OP_ENDIF
```

03. Scripts Condicionais e Contratos

- Script dual
  - Em hexadecimal
    6321036ae41e66b25bab55932a82b0f523d078158c98a0a0d60326bc6a83e370b
    be35fac03cc2a04b17567210307b4825b0ad5ce1dd42b0cf2d9add5305e07d9eb7
    6105f7bc130c0cac5bce70fac68
  - Decodificando:
    bitcoin-cli -datadir="." decodescript
    "6321036ae41e66b25bab55932a82b0f523d078158c98a0a0d60326bc6a83e370b
    be35fac03cc2a04b17567210307b4825b0ad5ce1dd42b0cf2d9add5305e07d9eb7
    6105f7bc130c0cac5bce70fac68"
    { "asm": "OP_IF
    036ae41e66b25bab55932a82b0f523d078158c98a0a0d60326bc6a83e370bbe35f
    OP_CHECKSIG 273100 OP_CHECKLOCKTIMEVERIFY OP_DROP OP_ELSE
    0307b4825b0ad5ce1dd42b0cf2d9add5305e07d9eb76105f7bc130c0cac5bce70f
    OP_CHECKSIG OP_ENDIF", ... }

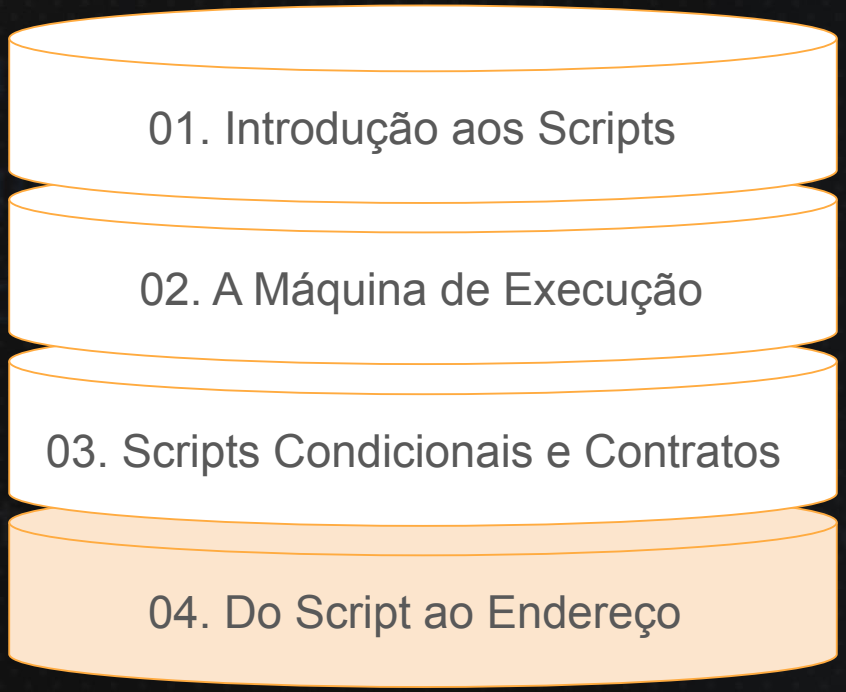
03. Scripts Condicionais e Contratos

- Script dual

Caminho 1 — [TRUE] → depois do bloco 273100 (assinatura de A)
Com ScriptSig junto (<assinatura_A> TRUE <redeemScript>)

Etapa	Operação	Pilha (topo à direita)	Descrição
1	PUSH <assinatura_A>	[assinatura_A]	A assinatura é empilhada
2	PUSH TRUE	[assinatura_A, TRUE]	Valor condicional inicial
3	OP_IF	[assinatura_A]	Entra no ramo IF
4	PUSH <pubkey_A>	[assinatura_A, pubkey_A]	Empilha a chave pública de A
5	OP_CHECKSIGVERIFY	[]	Verifica assinatura de A — se inválida → falha
6	PUSH 273100	[273100]	Empurra o limite de bloco
7	OP_CHECKLOCKTIMEVERIFY	[273100]	Compara o nLockTime da transação: deve ser ≥ 273100
8	OP_DROP	[]	Remove o número do topo (não precisa mais)
Final		[TRUE]	Script retorna verdadeiro — gasto autorizado por A

O Que Veremos!



04. Do Script ao Endereço

- Toda transação no Bitcoin contém
  - Script de bloqueio (scriptPubKey)
    - um pequeno programa que define quem pode gastar e em quais condições.

- Endereço → representação compacta desse script

04. Do Script ao Endereço

- 1. Escolha o “cadeado” — a condição de gasto
  Tipo de endereço → tipo diferente de cadeado → define o que será exigido quando os fundos forem gastos

Tipo	Regra de gasto	Hash usado
P2PKH (legacy)	Apresente `pubkey` + `assinatura` que valide para `HASH160(pubkey)`	`HASH160(pubkey)`
P2SH (script-hash)	Apresente um `redeemScript` cujo `HASH160(script)` corresponda + os dados que o satisfazem	`HASH160(script)`
P2WPKH (SegWit v0)	Witness program `v=0`, com 20 bytes de `HASH160(pubkey)`	`HASH160(pubkey)`
P2WSH (SegWit v0)	Witness program `v=0`, com 32 bytes de `SHA256(script)`	`SHA256(script)`

04. Do Script ao Endereço

- 2. O scriptPubKey — o “cadeado” em código
  scriptPubKey de acordo com o tipo

Tipo	Estrutura do scriptPubKey	Observação
P2PKH	`OP_DUP OP_HASH160 <20B> OP_EQUALVERIFY OP_CHECKSIG`	o mais comum (endereços “1” ou “m/n”)
P2SH	`OP_HASH160 <20B> OP_EQUAL`	usado em multisig, scripts compostos
P2WPKH	`0 <20B>`	formato SegWit simplificado
P2WSH	`0 <32B>`	usado em scripts complexos SegWit

04. Do Script ao Endereço

- 3. “Envelopando” o script em um endereço
  - script (ou seu hash) é convertido em um endereço legível
  - Base58Check
  - Bech32 / Bech32m

  - Legacy — Base58Check

Tipo	Prefixo	Exemplo (Signet/Testnet)
P2PKH	versão `0x6f` → começa com `m` ou `n`	`mw1v7YAUzMdCpWHjNm6He56SkFc2NabyGx`
P2SH	versão `0xc4` → começa com `2`	`2MuHsVSqmSpoddRxrDoAbpP6Jya38pDNFid`

04. Do Script ao Endereço

- 3. “Envelopando” o script em um endereço
  - codificação Base58Check
    - Letras e Números
    - Base64 - (‘0’, ‘O’, ‘I’, ‘l’, ‘+’, ‘/’)
  - prefixo + hash + checksum
  - Exemplo de P2PKH na signet:
    Hash160(pubkey) = aa0420f7f934bce88e817ccaa33f06208f106815
    Versão = 6f
    Payload = 6faa0420f7f934bce88e817ccaa33f06208f106815
    Checksum = 86aa3fbd (SHA256^2)

04. Do Script ao Endereço

- 3. “Envelopando” o script em um endereço
  - Exemplo de P2PKH na signet (continuação):
    Hash160(pubkey) = aa0420f7f934bce88e817ccaa33f06208f106815
    Versão = 6f
    Payload = 6faa0420f7f934bce88e817ccaa33f06208f106815
    Checksum = 86aa3fbd (SHA256^2)
  - Concatena o payload + checksum
    6faa0420f7f934bce88e817ccaa33f06208f10681586aa3fbd
  - hexadecimal → Base58
    mw1v7YAUzMdCpWHjNm6He56SkFc2NabyGx

04. Do Script ao Endereço

- 4. Interpretação do endereço — decodificando o envelope
  - bitcoin-cli -datadir="." sendtoaddress mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv 0.0001

Etapa	Descrição	Valor (hex)
1	**Endereço (Base58Check)**	`mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv`
2	**Decodifica Base58Check → bytes Hex (25B)**	`6f70973eeb12ae6b7aba4af95fa045251792219ac626959791`
3	**Primeiro byte (versão)**	`6f` → prefixo **testnet/signet** para P2PKH
4	**Próximos 20 bytes (hash)**	`70973eeb12ae6b7aba4af95fa045251792219ac6` → **HASH160(pubkey)**
5	**Últimos 4 bytes (checksum)**	`26959791` → usado só para detecção de erro (não entra no script)

04. Do Script ao Endereço

- 4. Interpretação do endereço — decodificando o envelope
  - bitcoin-cli -datadir="." sendtoaddress mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv 0.0001

Etapa	Descrição	Valor (hex)
1	**Endereço (Base58Check)**	`mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv`
2	**Decodifica Base58Check → bytes Hex (25B)**	`6f70973eeb12ae6b7aba4af95fa045251792219ac626959791`
3	**Primeiro byte (versão)**	`6f` → prefixo **testnet/signet** para P2PKH
4	**Próximos 20 bytes (hash)**	`70973eeb12ae6b7aba4af95fa045251792219ac6` → **HASH160(pubkey)**
5	**Últimos 4 bytes (checksum)**	`26959791` → usado só para detecção de erro (não entra no script)

04. Do Script ao Endereço

- bitcoin-cli -datadir="." validateaddress mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv

  { "isvalid": true, "address": "mqnHAHB3qrdLinFYASUyPnhFUNjEi7BJdv", "scriptPubKey": "76a91470973eeb12ae6b7aba4af95fa045251792219ac688ac", "isscript": false, "iswitness": false }

- `0x76` → `OP_DUP`
- `0xa9` → `OP_HASH160`
- `0x14` → `push de 20 bytes`
- `70973eeb12ae6b7aba4af95fa045251792219ac6` → `HASH160(pubkey) do destinatário`
- `0x88` → `OP_EQUALVERIFY`
- `0xac` → `OP_CHECKSIG`

scriptPubKey de um endereço P2PKH:
OP_DUP OP_HASH160 <20B> OP_EQUALVERIFY OP_CHECKSIG

FIM

Obrigado!

Grupo Whatsapp - Bitup Coders
https://chat.whatsapp.com/IB3rBamPe6QIvfncMBb3nF?mode=ems_copy_c

Rafael Penna
rapennas@gmail.com

bitcoinCoders bitups