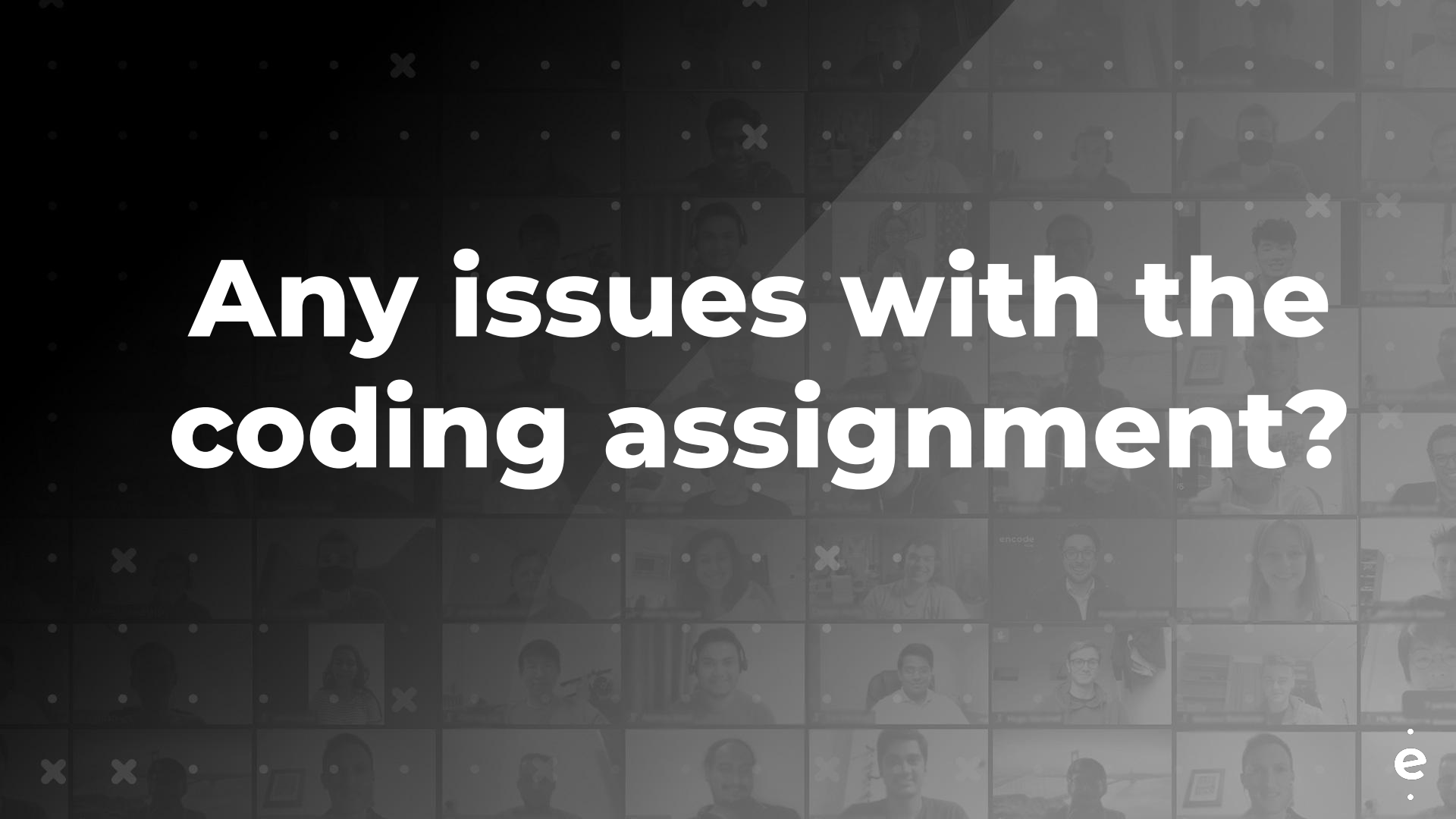
encode
CLUB

Algorand Bootcamp

Any questions from previous session?


Any issues with the coding assignment?

Recap

- Group Transactions

- State access and manipulation
  - Local State
  - Global State
  - External State
  - Box Storage

Today

- Asset Information
- App Information
- Indexer
- Smart Contract Testing (one of the few methods to do it)

Asset Information (On-chain)

Asset Information

- In addition to manipulating state on the blockchain, stateful smart contracts can also look up information about assets and account balances.

- On-chain, you can also look up the following details on PyTeal:
  - Algo Balances
    - Including Minimum Balance
  - Asset Holdings
  - If asset is frozen for an account
  - Asset Parameters

Algo Balance

- Balance() expression can be used to look up an account's balance

- Result is returned in microAlgos (1 ALGO = 1,000,000 microAlgos)

```
senderBalance = Balance(Txn.sender()) # get the balance of the sender
account1Balance = Balance(Txn.accounts[1]) # get the balance of Txn.accounts[1]
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

Minimum Balance

- MinBalance() expression can be used to look up an account's minimum balance

- Result is returned in microAlgos (1 ALGO = 1,000,000 microAlgos)

```
senderMinBalance = MinBalance(Txn.sender()) # get the minimum balance of the sender by passing the
account1MinBalance = MinBalance(Txn.accounts[1]) # get the minimum balance of Txn.accounts[1] by p
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

Minimum Balance

- Additionally, Balance and MinBalance can be used together to calculate how many Algos an account can spend without closing.

```
senderSpendableBalance = Balance(Txn.sender()) - MinBalance(Txn.sender()) # calculat
account1SpendableBalance = Balance(Txn.accounts[1]) - MinBalance(Txn.accounts[1]) #
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

Asset Holdings

- In addition to Algos, the Algorand blockchain also supports additional on-chain assets called Algorand Standard Assets (ASAs).

- The AssetHolding group of expressions can be used to look up information about the ASAs that an account holds.

- Similar to external state expressions, these expression return a MaybeValue.

Asset Holdings

- This value cannot be used directly, but has methods MaybeValue.hasValue() and MaybeValue.value().

- If the account has opted into the asset being looked up, hasValue() will return 1 and value() will return the value being looked up (either the asset's balance or frozen status).

- Otherwise, hasValue() and value() will return 0

Asset Balance

- The AssetHolding.balance expression can be used to look up how many units of an asset an account holds.

```
# get the balance of the sender for asset `Txn.assets[0]`
# if the account is not opted into that asset, returns 0
senderAssetBalance = AssetHolding.balance(Txn.sender(), Txn.assets[0])
program = Seq([
    senderAssetBalance,
    senderAssetBalance.value()
])

# get the balance of Txn.accounts[1] for asset `Txn.assets[1]`
# if the account is not opted into that asset, exit with an error
account1AssetBalance = AssetHolding.balance(Txn.accounts[1], Txn.assets[1])
program = Seq([
    account1AssetBalance,
    Assert(account1AssetBalance.hasValue()),
    account1AssetBalance.value()
])
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

Is Frozen?

- The AssetHolding.frozen expression can be used to check if an asset is frozen for an account. A value of 1 indicates frozen and 0 indicates not frozen

```
# get the frozen status of the sender for asset `Txn.assets[0]`
# if the account is not opted into that asset, returns 0
senderAssetFrozen = AssetHolding.frozen(Txn.sender(), Txn.assets[0])
program = Seq([
    senderAssetFrozen,
    senderAssetFrozen.value()
])

# get the frozen status of Txn.accounts[1] for asset `Txn.assets[1]`
# if the account is not opted into that asset, exit with an error
account1AssetFrozen = AssetHolding.frozen(Txn.accounts[1], Txn.assets[1])
program = Seq([
    account1AssetFrozen,
    Assert(account1AssetFrozen.hasValue()),
    account1AssetFrozen.value()
])
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

Asset Parameters

- The AssetParam group of expressions are used to access asset parameters
- Like AssetHolding, these expressions return a MaybeValue.
- The hasValue() method will return 0 only if the asset being looked up does not exist (i.e. the ID in Txn.assets does not represent an asset).

Asset Parameters

- For optional parameters that are not set, hasValue() will still return 1 and value() will return a zero-length byte string (all optional parameters are TealType.bytes).

Asset Parameters

- AssetParam.total()
- AssetParam.decimals()
- AssetParam.defaultFrozen()
- AssetParam.unitName()
- AssetParam.name()
- AssetParam.url()
- AssetParam.metadataHash()
- AssetParam.creator()
- AssetParam.manager()
- AssetParam.reserve()
- AssetParam.freeze()
- AssetParam.clawback()

AssetParam

- Example of checking the total supply of an asset:

```
# get the total number of units for asset `Txn.assets[0]`
# if the asset is invalid, exit with an error
assetTotal = AssetParam.total(Txn.assets[0])

program = Seq([
    assetTotal,
    Assert(assetTotal.hasValue()),
    assetTotal.value()
])
```

Source: https://pyteal.readthedocs.io/en/stable/assets.html#asset-holdings

App Information (On-chain)

App Parameters

- Similar to assets, you can also access an application's information on-chain using AppParam

- AppParam allows you to retrieve the application specific information such as escrow address, approvalProgram, clearStateProgram etc.

- AppParam methods also return a MaybeValue

App Parameters

- An application ID must be passed into the applications array when preparing a transaction

- address()
  - AppParam.address(app): Get the escrow address of an application

- approvalProgram()
  - AppParam.approvalProgram(app): Get the bytecode of Approval Program for the application

App Parameters

- clearStateProgram()
  - AppParam.clearStateProgram(app): Get the bytecode of Clear State Program for the application

- creator()
  - AppParam.creator(app): Get the creator address for the application

- And more.....

Indexer

Indexer

- The primary purpose of this Indexer is to provide a REST API interface of API calls to support searching the Algorand Blockchain.

- The Indexer provides a set of REST API calls for searching blockchain Transactions, Accounts, Assets and Blocks.

- Each of these calls also provides several filter parameters to support refining searches.

Instantiation

```
# requires Python SDK version 1.3 or higher
from algosdk.v2client import indexer

# instantiate indexer client
myindexer = indexer.IndexerClient(indexer_token="", indexer_address="http://localhost:8980")
```

Source:
https://developer.algorand.org/docs/get-details/indexer/?from_query=Indexer#sdk-client-instantiations

Indexer

- Indexer calls can also be customized to return:
  - Paginated Results
  - Historical Data Searches
  - Note Field Searching

Paginated Results

- When searching large amounts of blockchain data often the results may be too large to process in one given operation

- The indexer imposes hard limits on the number of results returned for specific searches.

- The default limits for these searches are summarized in the table next slide.

Paginated Results

Search Type	Maximum number of results per search
API Resources Per Account	1,000
Transactions Search	1,000
Accounts Search	100
Assets Search	100
Balances Search	1,000
Applications Search	100

Source: https://developer.algorand.org/docs/get-details/indexer/?from_query=Indexer#paginated-results

Paginated Results

- When trying to find specific transactions, the Indexer supplies a pagination method that allows separating the results into several REST calls to return larger result sets.

- When used with the limit parameter the results for large data sets can be returned in expected result counts.

Paginated Results

```
response = myindexer.search_transactions(
min_amount=10, limit=5)

# Pretty Printing JSON string
print(json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Historical Data

- Many of the REST calls support getting values at specific rounds.

- This means that the Indexer will do calculations that determine what specific values were at a specific round.

Historical Data

```
response = myindexer.account_info(
    address="7WENHRCKEAZHD37QMB5T7I2KWU7IZGMCC3EVAO7TQADV7V5APXOKUBILCI", block=50)
print("Account Info: " + json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Note Field

- Every transaction has the ability to add up to a 1kb note in the note field

- Several of the REST APIs provide the ability to search for a prefix that is present in the note field, meaning that the note starts with a specific string.

Note Field

```
import base64
note_prefix = 'showing prefix'.encode()

response = myindexer.search_transactions(
    note_prefix=note_prefix)

print("note_prefix = " +
    json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Accounts

- The ‘/accounts’ call can be used to search for accounts on the Algorand blockchain.
- This call also supports Historical Data Searches if the Indexer is configured for the /accounts call.

```
response = myindexer.accounts(
    asset_id=312769)
print("Account Info: " + json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Account

- You can also query a specific account to look up its balances

```
# gets account
response = myindexer.account_info(
    address="TDO7JWA77FH3T2HP5ZOZWFKUQDQEAPD25HDKDVEAAWQKBWTMNMYRXOOYGA")
print("Account Info: " + json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Account

- You can look at applications created by a specific account

```
address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4"

response = myindexer.lookupAccountCreatedApplications(address)

print(json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Account

- You can look at assets created by a specific account

```
address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4"

response = myindexer.lookupAccountCreatedAssets(address)

print(json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Account

- You can look at all the assets that an account has opted into

```
address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4"

response = myindexer.lookupAccountAssets(address)

print(json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Local State

- You can look at application’s local state for a specific account

```
address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4"

response = myindexer.lookupAccountAppLocalStates(address)

print(json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Time range

- The range of transactions to be searched can be restricted based on the time by using the before-time and after-time parameters. JS example:

```
(async () => {
    let address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4";
    let start_time = "2020-06-03T10:00:00-05:00";
    let response = await indexerClient.searchForTransactions()
        .address(address)
        .afterTime(start_time).do();
    console.log("start_time: 06/03/2020 11:00:00 = " + JSON.stringify(response, undefined, 2));
    }
)().catch(e => {
    console.log(e);
    console.trace();
});
```

Source: https://developer.algorand.org/docs/get-details/indexer

Round range

- Transaction searches can also be restricted to round ranges using the min-round and max-round parameters. JS example:

```
(async () => {
    let address = "XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4";
    let min_round = 7048876;
    let max_round = 7048878;
    let response = await indexerClient.searchForTransactions()
        .address(address).maxRound(max_round)
        .minRound(min_round).do();
    console.log("Information for Transaction search: " + JSON.stringify(response, undefined, 2));
    }
)().catch(e => {
    console.log(e);
    console.trace();
});
```

Source: https://developer.algorand.org/docs/get-details/indexer

Transaction ID

- Searching for a specific transaction can be achieved by supplying the transaction id using the txid parameter.

```
response = myindexer.search_transactions_by_address(
    address="XIU7HGGAJ3QOTATPDSIIHPFVKMICXKHMOR2FJKHTVLII4FAOA3CYZQDLG4",
    txid="QZS3B2XBBS47S6X5CZGKKC2FC7HRP5VJ4UNS7LPGHP24DUECHAAA")

print("txid: QZS3B2XBBS47S6X5CZGKKC2FC7HRP5VJ4UNS7LPGHP24DUECHAAA = " +
    json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Transaction Type

- The Indexer supports looking for pay, keyreg, appl, acfg, axfer and afrz transaction types.

```
response = myindexer.search_transactions_by_address(
    address="SWOUICD7Y5PQBWWEYC4XZAQZI7FJRZLD5O3CP4GU2Y7FP3QFKA7RHN2WJU",
    txn_type="acfg")

print("txn_type: acfg = " +
    json.dumps(response, indent=2, sort_keys=True))
```

Source: https://developer.algorand.org/docs/get-details/indexer

Indexer

- ……and so many more

- With an indexer, you can potentially look up the entire Algorand network

- You can limit the results, look for specific results, accounts, assets, and applications

Smart Contract Testing

Assignment

1. Review the slides and recordings of this week
2. Google Forms Quiz (Test of Knowledge)
3. Smart Contract Testing:
    a. Python is recommended, but you can use other languages as well
    b. Build upon the example discussed today, and create additional unit test cases for each smart contract method/action (register, vote, clear)
    c. Assert the state of application (global/local) with the expected state after the operation is performed to test for functionality

Next Week

1. Frontend (ReactJS)
2. Wallets integration with frontend (MyAlgo, Algosigner, Pera)
3. Final Project Discussion:
    a. What is expected?
    b. How to approach?
    c. What to submit?

Resources

1. Testnet Funds Dispenser: Link
2. AlgoNode APIs: Link
3. PyTeal Docs: Link
4. Algorand Developer Portal: Link

Thank You

Email: hi@lalithmedury.com

Twitter: @LalithMedury

Web: https://lalithmedury.com

Email: info@encode.club

Twitter: @encodeclub

Web: https://encode.club