encode
CLUB

Algorand Bootcamp

Very important

- Close Out and Clear State
- **Review Next Slide**

Smart contracts are implemented using two programs:

- The `ApprovalProgram` is responsible for processing all application calls to the contract, with the exception of the clear call (described in the next bullet). This program is responsible for implementing most of the logic of an application. Like smart signatures, this program will succeed only if one nonzero value is left on the stack upon program completion or the `return` opcode is called with a positive value on the top of the stack.
- The `ClearStateProgram` is used to handle accounts using the clear call to remove the smart contract from their balance record. This program will pass or fail the same way the `ApprovalProgram` does.

Source: Link

Very important

- Close Out and Clear State

- CloseOut - Accounts use this transaction to close out their participation in the contract. This call can fail based on the TEAL logic, preventing the account from removing the contract from its balance record.
- ClearState - Similar to CloseOut, but the transaction will always clear a contract from the account's balance record whether the program succeeds or fails.

Source: Link

Assignment

- Coding assignment due Next Monday
- Expecting a little self-study
- **Hints provided today**

Any questions from previous session?

Recap

- Global Parameters
- Atomic Transactions
- Control Flow
- Seq, Cond Expressions
- If, If-Else, If-ElseIf-Else, For, While
- Subroutines

Today

- Group Transactions

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

Atomic Transactions

- Group of transactions that either ensure all go through or none go through
- Useful when having relatively complicated transaction logic when order of transactions is important
- Transactions accessed by Gtxn object

Atomic Transactions

- If transactions are grouped, they will have a group ID
- Even if one of the transactions fail, all the other transactions in the group fail
- Use case: Grouping a payment call with an application transaction call

Accessing IDs

- One of the group transaction can create an asset or an application
- Other transactions within the group can access the ID of newly created asset or an application
- GeneratedID(txnIndex) must be used

Accessing IDs

```
GeneratedID(0) # retrieves the ID from the 0th transaction in current group
GeneratedID(Int(10)) # retrieves the ID from the 10th transaction in group
```

Source: https://pyteal.readthedocs.io/en/stable/loading_group_transaction.html

Scratch Slots

- We can access scratch variables of transactions within the same group

- ImportScratchValue(txnIndex, scratchSlotIndex) must be used

Scratch Slots

App A:

```
# App is called at transaction index 0
greeting = ScratchVar(TealType.bytes, 20) # this variable will live in scratch slot 20
program = Seq([
    If(Txn.sender() == App.globalGet(Bytes("creator")))
    .Then(greeting.store(Bytes("hi creator!")))
    .Else(greeting.store(Bytes("hi user!"))),
    Return(Int(1))
])
```

App B:

```
greetingFromPreviousApp = ImportScratchValue(0, 20) # loading scratch slot 20 from the transactio
program = Seq([
    # not shown: make sure that the transaction at index 0 is an app call to App A
    App.globalPut(Bytes("greeting from prev app"), greetingFromPreviousApp),
    Return(Int(1))
])
```

Source: https://pyteal.readthedocs.io/en/stable/loading_group_transaction.html

Storage

Storage

- **Global State**
  - Stored at application level
- **Local State**
  - Stored at account level
- **Box Storage**
  - Box storage is also global and allows contracts to use larger segments of storage.

Global Storage

Global Storage

Allocation

- Can include between 0 and 64 key/value pairs for a total of 8K of memory to share among them.

- The amount of global storage is allocated in k/v units, and determined at contract creation. This schema is immutable after creation.

- The contract creator address is responsible for funding the global storage

Global Storage

Reading

- Can be read by any app call that has specified app a's ID in its foreign apps array.

- Can be read on-chain using the k/v pairs defined (from off-chain, can be read using goal or APIs + SDKs).

Global Storage

Reading

```
App.globalGet(Bytes("status"))
App.globalGet(Bytes("total supply"))
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#global-state

Global Storage

Writing

- Can only be written by app a.

Deletion

- Is deleted when app a is deleted.
- Cannot otherwise be deallocated (though of course the contents can be cleared by app a, but this does not change the minimum balance requirement).

Global Storage

Writing

```
App.globalPut(Bytes("status"), Bytes("active")) # write a byte slice
App.globalPut(Bytes("total supply"), Int(100)) # write a uint64
```

Deleting

```
App.globalDel(Bytes("status"))
App.globalDel(Bytes("total supply"))
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#global-state

Local Storage

Local Storage

Allocation

- Is allocated when account x opts in to app a (submits a transaction to opt-in to app a).
- Can include between 0 and 16 key/value pairs for a total of 2KB of memory to share among them.
- The amount of local storage is allocated in k/v units, and determined at contract creation. This cannot be edited later.
- The opted-in user address is responsible for funding the local storage (by an increase to their minimum balance).

Local Storage

Reading

- Can be read by any app call that has app x in its foreign apps array and account x in its foreign accounts array.

- Can be read on-chain using the k/v pairs defined (from off-chain, can be read using goal and the SDKs).

Local Storage

Reading

```
App.localGet(Txn.sender(), Bytes("role")) # read from the sender's account
App.localGet(Txn.sender(), Bytes("balance")) # read from the sender's account
App.localGet(Txn.accounts[1], Bytes("balance")) # read from Txn.accounts[1]
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#local-state

Local Storage

Writing

- Is editable only by app a, but is delete-able by app a or the user x

Local Storage

Deletion

- Deleting an app does not affect its local storage. Accounts must clear out of app to recover minimum balance.

- *Clear state*. Every Smart Contract on Algorand has two programs: the *approval* and the *clear state* program. An account holder can clear their local state for an app at any time

Local Storage

Deletion

- The purpose of the clear state program is to allow the app to handle the clearing of that local state gracefully.

- Account x can request to clear its local state using a close out transaction.

- Account x can clear its local state for app a using a clear state transaction, which will always succeed, even after app a is deleted.

Local Storage

Writing

```
App.localPut(Txn.sender(), Bytes("role"), Bytes("admin")) # write a byte slice to the sender's
App.localPut(Txn.sender(), Bytes("balance"), Int(10)) # write a uint64 to the sender's account
App.localPut(Txn.accounts[1], Bytes("balance"), Int(10)) # write a uint64 to Txn.account[1]
```

Deleting

```
App.localDel(Txn.sender(), Bytes("role")) # delete "role" from the sender's account
App.localDel(Txn.sender(), Bytes("balance")) # delete "balance" from the sender's account
App.localDel(Txn.accounts[1], Bytes("balance")) # delete "balance" from Txn.accounts[1]
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#global-state

External State

External State

- It’s possible for applications to read state written by other applications.
- This is possible using the App.globalGetEx and App.localGetEx functions.
- Unlike the other state access functions, App.globalGetEx and App.localGetEx return a MaybeValue.

External State

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

- If the key being accessed exists in the context of the app being read, hasValue() will return 1 and value() will return its value

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

External Global

- To read a value from the global state of another application, use the App.globalGetEx function.

- In order to use this function you need to pass in an integer that represents an application to read from. This integer corresponds to an actual application ID that appears in the Txn.applications array.

External Global

```
# get "status" from the global context of Txn.applications[0] (the current app)
# if "status" has not been set, returns "none"
myStatus = App.globalGetEx(Txn.applications[0], Bytes("status"))

program = Seq([
    myStatus,
    If(myStatus.hasValue(), myStatus.value(), Bytes("none"))
])

# get "status" from the global context of Txn.applications[1]
# if "status" has not been set, returns "none"
otherStatus = App.globalGetEx(Txn.applications[1], Bytes("status"))
program = Seq([
    otherStatus,
    If(otherStatus.hasValue(), otherStatus.value(), Bytes("none"))
])

# get "total supply" from the global context of Txn.applications[1]
# if "total supply" has not been set, returns the default value of 0
otherSupply = App.globalGetEx(Txn.applications[1], Bytes("total supply"))
program = Seq([
    otherSupply,
    otherSupply.value()
])
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#external-global

External Local

- To read a value from an account's local state for another application, use the App.localGetEx function.

- There are three arguments:
  - The first argument is the address of the account to read from (in the same format as App.localGet)
  - The second argument is the ID of the application to read from
  - And the third argument is the key to read.

External Local

```
# get "role" from the local state of Txn.accounts[0] (the sender) for the current app
# if "role" has not been set, returns "none"
myAppSenderRole = App.localGetEx(Txn.accounts[0], Int(0), Bytes("role"))
program = Seq([
    myAppSenderRole,
    If(myAppSenderRole.hasValue(), myAppSenderRole.value(), Bytes("none"))
])

# get "role" from the local state of Txn.accounts[1] for the current app
# if "role" has not been set, returns "none"
myAppOtherAccountRole = App.localGetEx(Txn.accounts[1], Int(0), Bytes("role"))
program = Seq([
    myAppOtherAccountRole,
    If(myAppOtherAccountRole.hasValue(), myAppOtherAccountRole.value(), Bytes("none"))
])

# get "role" from the local state of Txn.accounts[0] (the sender) for the app with ID 31
# if "role" has not been set, returns "none"
otherAppSenderRole = App.localGetEx(Txn.accounts[0], Int(31), Bytes("role"))
program = Seq([
    otherAppSenderRole,
    If(otherAppSenderRole.hasValue(), otherAppSenderRole.value(), Bytes("none"))
])

# get "role" from the local state of Txn.accounts[1] for the app with ID 31
# if "role" has not been set, returns "none"
otherAppOtherAccountRole = App.localGetEx(Txn.accounts[1], Int(31), Bytes("role"))
program = Seq([
    otherAppOtherAccountRole,
    If(otherAppOtherAccountRole.hasValue(), otherAppOtherAccountRole.value(), Bytes("none"))
])
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#external-local

Box Storage

Box Storage

Allocation

- App a can allocate as many boxes as it needs, when it needs them.

- App a allocates a box using the box_create opcode in its TEAL program, specifying the name and the size of the box being allocated.
    - Boxes can be any size from 0 to 32K bytes.
    - Box names must be at least 1 byte, at most 64 bytes, and must be unique within app a.

Box Storage

Allocation

- The app account(the smart contract) is responsible for funding the box storage (with an increase to its minimum balance requirement, see below for details).

- A box name and app id must be referenced in the boxes array of the app call to be allocated.

Box Storage

Reading

- App a is the only app that can read the contents of its boxes on-chain. This on-chain privacy is unique to box storage.

- Recall that everything can be read by anybody from off-chain using the algod or indexer APIs.

- To read box b from app a, the app call must include b in its boxes array.

Box Storage

Reading

```
# extract a segment of length 10 starting at the 5th byte in a box named `NoteBook`
App.box_extract(Bytes("NoteBook"), Int(5), Int(10))
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#reading-from-a-box

Box Storage

Writing

- App a is the only app that can write the contents of its boxes.

- As with reading, each box ref in the boxes array allows an app call to write 1kb of box state - 1kb of “box write budget”.

Box Storage

Deletion

- App a is the only app that can delete its boxes.

- If an app is deleted, its boxes are not deleted. The boxes will not be modifiable but still can be queried using the SDKs. The minimum balance will also be locked.

- The correct cleanup design is to look up the boxes from off-chain and call the app to delete all its boxes before deleting the app itself.

Box Storage

Writing

```
# create a 42 bytes length box called `poemLine` with content
App.box_put(Bytes("poemLine"), Bytes("Of that colossal wreck, boundless and bare"))

# write to box `poemLine` with new value
App.box_put(Bytes("poemLine"), Bytes("The lone and level sands stretch far away."))
```

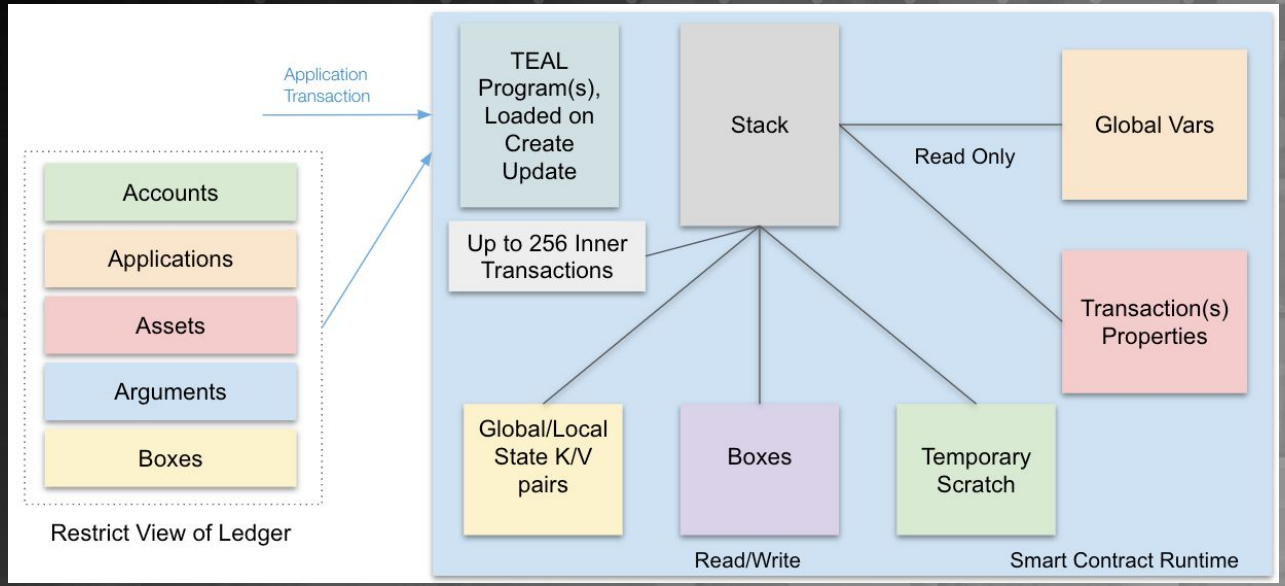
Deleting

```
# delete the box `boxToRemove`, asserting that it existed prior to this
Assert(App.box_delete(Bytes("boxToRemove")))

# delete the box `mightExist` and ignore the return value
Pop(App.box_delete(Bytes("mightExist")))
```

Source: https://pyteal.readthedocs.io/en/stable/state.html#reading-from-a-box

Arrays



Source: https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/

Assignment

1. Review the slides and recordings of this week
2. Google Forms Quiz (Test of Knowledge)
3. Coding assignment is due next week:
    a. **Hint**: Use AssetHolding to read ASA balance of an account on-chain: (https://pyteal.readthedocs.io/en/stable/api.html?highlight=AssetHolding#pyteal.AssetHolding)
4. Assignment must be submitted using the same typeform link shared for week-2 assignment

Next Week

1. Asset Information
2. App Information
3. Indexer
4. Smart Contract Testing (one of the few methods to do it)

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