encode CLUB

Algorand Bootcamp

Any questions from last week?

Any issues with the assignment?

Recap

- Development Environment Setup
- Introduction to Algorand Infrastructure (Tools, SDKs, APIs)
- DApp overview
- Teal and PyTeal overview

Today

- Pyteal overview
- Data Types and Constants
- Arithmetic and Byte Operators
- Transaction Fields
- Global Parameters
- Scratch Space

PyTeal Overview

Py

- With) in Pyth

```python
from pyteal import *

"""Basic Bank"""


def bank_for_account(receiver):
    """Only allow receiver to withdraw funds from this contract account.

    Args:
        receiver (str): Base 32 Algorand address of the receiver.
    """

    is_payment = Txn.type_enum() == TxnType.Payment
    is_single_tx = Global.group_size() == Int(1)
    is_correct_receiver = Txn.receiver() == Addr(receiver)
    no_close_out_addr = Txn.close_remainder_to() == Global.zero_address()
    no_rekey_addr = Txn.rekey_to() == Global.zero_address()
    acceptable_fee = Txn.fee() <= Int(1000)

    return And(
        is_payment,
        is_single_tx,
        is_correct_receiver,
        no_close_out_addr,
        no_rekey_addr,
        acceptable_fee,
    )


if __name__ == "__main__":
    program = bank_for_account(
        "ZZAF5ARA4MEC5PVDOP64JM5O5MQST63Q2KOY2FLYFLXXD3PFSNJJBYAFZM"
    )
    print(compileTeal(program, mode=Mode.Signature, version=3))
```

PyTeal vs Teal

```python
from pyteal import *

def approval_program():

    hello = Seq([
        App.globalPut(Txn.sender(), Txn.application_args[1]),
        Return(Int(1))
    ])

    program = Cond(
        [Txn.application_id() == Int(0), Return(Int(1))],
        [Txn.on_completion() == OnComplete.DeleteApplication, Return(Int(0))],
        [Txn.on_completion() == OnComplete.UpdateApplication, Return(Int(0))],
        [Txn.on_completion() == OnComplete.OptIn, Return(Int(0))],
        [Txn.application_args[0] == Bytes("hello"), hello]
    )

    return program


def clear_state_program():
    return Int(1)

if __name__ == "__main__":
    with open("approval.teal", "w") as f:
        compiled = compileTeal(approval_program(), mode=Mode.Application, version=6)
        f.write(compiled)

    with open("clear_state.teal", "w") as f:
        compiled = compileTeal(clear_state_program(), mode=Mode.Application, version=6)
        f.write(compiled)
```

```
#pragma version 6
txn ApplicationID
int 0
==
bnz main_l10
txn OnCompletion
int DeleteApplication
==
bnz main_l9
txn OnCompletion
int UpdateApplication
==
bnz main_l8
txn OnCompletion
int OptIn
==
bnz main_l7
txna ApplicationArgs 0
byte "hello"
==
bnz main_l6
err
main_l6:
txn Sender
txna ApplicationArgs 1
app_global_put
int 1
return
main_l7:
int 0
return
main_l8:
int 0
return
main_l9:
```

Data Types and Constants

Data Types

A PyTeal expression has one of the following two data types:

- TealType.uint64, 64 bit unsigned integer

  Int(n) creates a TealType.uint64 constant, where n >= 0 and n < 2 ** 64.

- TealType.bytes, a slice of bytes

  A byte slice is a binary string. There are several ways to encode a byte slice in PyTeal:

  Can be encoded into Base16, Base32, Base64

Conversion

Converting a value to its corresponding value in the other data type is supported by the following two operators:

- Itob(n): generate a TealType.bytes value from a TealType.uint64 value n
- Btoi(b): generate a TealType.uint64 value from a TealType.bytes value b

Note: These operations are **not** meant to convert between human-readable strings and numbers. Itob produces a big-endian 8-byte encoding of an unsigned integer, not a human readable string. For example, Itob(Int(1)) will produce the string "x00x00x00x00x00x00x00x01" not the string "1".

Operators

- Lt (a, b) => Ex: Int(1) < Int(5)
- Gt (a, b) => Ex: Int(1) > Int(5)
- Le (a, b) => Ex: Int(1) <= Int(5)
- Ge (a, b) => Ex: Int(1) <= Int(5)
- Add (a, b) => Ex: Int(1) + Int(5)
- Minus (a, b) => Ex: Int(5) - Int(1)
- Div (a, b) => Ex: Int(5) / Int(1)
- Mod (a, b) => Ex: Int(5) % Int(2)

Operators

- Exp (a, b) => Ex: Int(5) ** Int(2)
- Eq (a, b) => Ex: Int(1) == Int(1)
- Neq (a, b) => Ex: Int(5) != Int(4)
- And (a, b) => Ex: And(Int(1), Int(1)) *Note: Returns 1 only if a > 0 and b > 0, 0 otherwise*
- Or (a, b) => Ex: And(Int(1), Int(0)) *Note: Returns 1 if a > 0 or b > 0, 0 otherwise*
- Not (a) => Ex: Not(Int(0)) *Note: Returns 1 if a = 0*
- *BitwiseAnd (a,b), BitwiseOr (a,b), BitwiseXor (a,b), BitwiseNot (a)*

Byte Operators

- In addition to the standard arithmetic operators above, PyTeal also supports operations that manipulate the individual bits and bytes of PyTeal values.

- To use these operations, you'll need to provide an index specifying which bit or byte to access. These indexes have different meanings depending on whether you are manipulating integers or byte slices.

Byte Operators

- In addition to the standard arithmetic operators above, PyTeal also supports operations that manipulate the individual bits and bytes of PyTeal values.

- To use these operations, you'll need to provide an index specifying which bit or byte to access. These indexes have different meanings depending on whether you are manipulating integers or byte slices.

Bit Manipulation

- GetBit
  - The GetBit expression can extract individual bit values from integers and byte strings

- SetBit
  - The SetBit expression can modify individual bit values from integers and byte strings.

Example

```
GetBit(Int(16), Int(0)) # get the 0th bit of 16, produces 0
GetBit(Int(16), Int(4)) # get the 4th bit of 16, produces 1
GetBit(Int(16), Int(63)) # get the 63rd bit of 16, produces 0
GetBit(Int(16), Int(64)) # get the 64th bit of 16, invalid index

GetBit(Bytes("base16", "0xf0"), Int(0)) # get the 0th bit of 0xf0, produces 1
GetBit(Bytes("base16", "0xf0"), Int(7)) # get the 7th bit of 0xf0, produces 0
GetBit(Bytes("base16", "0xf0"), Int(8)) # get the 8th bit of 0xf0, invalid index
```

```
SetBit(Int(0), Int(4), Int(1)) # set the 4th bit of 0 to 1, produces 16
SetBit(Int(4), Int(0), Int(1)) # set the 0th bit of 4 to 1, produces 5
SetBit(Int(4), Int(0), Int(0)) # set the 0th bit of 4 to 0, produces 4

SetBit(Bytes("base16", "0x00"), Int(0), Int(1)) # set the 0th bit of 0x00 to 1, produces 0x80
SetBit(Bytes("base16", "0x00"), Int(3), Int(1)) # set the 3rd bit of 0x00 to 1, produces 0x10
SetBit(Bytes("base16", "0x00"), Int(7), Int(1)) # set the 7th bit of 0x00 to 1, produces 0x01
```

Source:
https://pyteal.readthedocs.io/en/stable/arithmetic_expression.html#bit-and-byte-operations

Byte Manipulation

- GetByte
  - The GetByte expression can extract individual bytes from byte strings.

- SetByte
  - The SetByte expression can modify individual bytes in byte strings.

Example

```
GetByte(Bytes("abc"), Int(0)) # get the 0th byte of "abc", produces 97 (ASCII 'a')
GetByte(Bytes("abc"), Int(1)) # get the 1st byte of "abc", produces 98 (ASCII 'b')
GetByte(Bytes("abc"), Int(2)) # get the 2nd byte of "abc", produces 99 (ASCII 'c')
```

```
SetByte(Bytes("abc"), Int(0), Int(98)) # set the 0th byte of "abc" to 98 (ASCII 'b')
SetByte(Bytes("abc"), Int(1), Int(66)) # set the 1st byte of "abc" to 66 (ASCII 'B')
```

Source:
https://pyteal.readthedocs.io/en/stable/arithmetic_expression.html#bit-and-byte-operations

Byte Operators

- **Length**
  - The length of a byte slice can be obtained using the Len
- **Concatenation**
  - Byte slices can be combined using the Concat expression.
- **Substring Extraction**
  - Byte slices can be extracted from other byte slices using the Substring and Extract expressions.
- **Extract**
  - The Extract expression can extract part of a byte slice given the start index and length.

Example

```
Len(Bytes("")) # will produce 0
Len(Bytes("algorand")) # will produce 8
```

```
Concat(Bytes("a"), Bytes("b"), Bytes("c")) # will produce "abc"
```

```
Substring(Bytes("algorand"), Int(2), Int(8)) # will produce "gorand"
```

```
Extract(Bytes("algorand"), Int(2), Int(6)) # will produce "gorand"
```

Source:
https://pyteal.readthedocs.io/en/stable/arithmetic_expression.html#bit-and-byte-operations

Transaction Fields

Transaction Fields

- PyTeal smart contracts can access properties of the current transaction and the state of the blockchain when they are running.

- Information about the current transaction being evaluated can be obtained using the Txn object

Common Fields

- Txn.type_enum()
- Txn.sender()
- Txn.receiver()
- Txn.fee()
- Txn.rekey_to()

Examples

```
safety_cond = And(
    Txn.type_enum() == TxnType.Payment,
    Txn.close_remainder_to() == Global.zero_address(),
    Txn.rekey_to() == Global.zero_address(),
)
```

```
split_core = And(
    Txn.type_enum() == TxnType.Payment,
    Txn.fee() < tmpl_fee,
    Txn.rekey_to() == Global.zero_address(),
)

split_transfer = And(
    Gtxn[0].sender() == Gtxn[1].sender(),
    Txn.close_remainder_to() == Global.zero_address(),
    Gtxn[0].receiver() == tmpl_rcv1,
    Gtxn[1].receiver() == tmpl_rcv2,
    Gtxn[0].amount()
    == ((Gtxn[0].amount() + Gtxn[1].amount()) * tmpl_ratn) / tmpl_ratd,
    Gtxn[0].amount() == tmpl_min_pay,
)

split_close = And(
    Txn.close_remainder_to() == tmpl_own,
    Txn.receiver() == Global.zero_address(),
    Txn.amount() == Int(0),
    Txn.first_valid() > tmpl_timeout,
)
```

```
periodic_pay_core = And(
    Txn.type_enum() == TxnType.Payment,
    Txn.fee() < tmpl_fee,
    Txn.first_valid() % tmpl_period == Int(0),
    Txn.last_valid() == tmpl_dur + Txn.first_valid(),
    Txn.lease() == tmpl_lease,
)

periodic_pay_transfer = And(
    Txn.close_remainder_to() == Global.zero_address(),
    Txn.rekey_to() == Global.zero_address(),
    Txn.receiver() == tmpl_rcv,
    Txn.amount() == tmpl_amt,
)

periodic_pay_close = And(
    Txn.close_remainder_to() == tmpl_rcv,
    Txn.rekey_to() == Global.zero_address(),
    Txn.receiver() == Global.zero_address(),
    Txn.first_valid() == tmpl_timeout,
    Txn.amount() == Int(0),
)
```

Transaction Types

- TxnType.Payment
- TxnType.KeyRegistration
- TxnType.AssetConfig
- TxnType.AssetTransfer

Transaction Types

- TxnType.AssetFreeze
- TxnType.ApplicationCall
- TxnType.AssetConfig
- TxnType.AssetTransfer

Application Call Txns

- Txn.application_id()
- Txn.on_completion()
- Txn.accounts()
- Txn.assets()

Application Call Txns

- Txn.application_args()
- Txn.created_application_id()
- Txn.approval_program()
- Txn.clear_state_program()

Payment

- Txn.receiver()
- Txn.amount()
- Txn.close_remainder_to()

Asset Transfer

- Txn.xfer_asset()
- Txn.asset_amount()
- Txn.asset_sender()
- Txn.asset_receiver()

Scratch Space

- Scratch space is a temporary place to store values for later use in your program.
- Any changes to scratch space do not persist beyond the current transaction.
- Available in both Application and Signature mode.

Scratch Space

```
myvar = ScratchVar(TealType.uint64) # assign a scratch slot in any available slot
program = Seq([
    myvar.store(Int(5)),
    Assert(myvar.load() == Int(5))
])
anotherVar = ScratchVar(TealType.bytes, 4) # assign this scratch slot to slot #4
```

Source:
https://pyteal.readthedocs.io/en/stable/scratch.html#scratchvar-writing-and-reading-to-from-scratch-space

Next Week

- Control Flow
- Seq, Cond Expressions
- If, If-Else, If-ElseIf-Else, For, While
- Subroutines

Assignment

1. Review the slides and recordings of this week
2. It is an assignment of knowledge
3. Assignment Link:

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