Intro to PyTeal

Getting started with PyTeal

September 16, 2022 Joe Polny

Algorand™ FOUNDATION

TEAL vs PyTeal

- TEAL
  - Assembly-like language
  - Cumbersome to write
  - Hard to read
- PyTeal
  - Familiar for Python devs
  - Easier to read and write
  - Access to abstractions such as types, loops, etc.

Hello World

```
from pyteal import *

def approval_program():
    return Seq(
        Log(Bytes("Hello World!")),
        Return(Int(1))
    )

print(compileTeal(approval_program(), mode=Mode.Application, version=7))
```

Seq

```
    return Seq(
```

- `Seq` stands for sequence, meaning a sequence of PyTeal expressions
- `compileTeal` ultimately expects a PyTeal expression
- Expressions are methods that compile down to TEAL
- All other methods, such as `Log` or `Return`, also expect PyTeal expression

Bytes and Int

```
        Log(Bytes("Hello World!")),
```

- Why not `Log("Hello World")` or `Return(1)`?
- `"Hello World"` and `1` are python `str` and `int` objects, which is not a PyTeal expression

Return, Approve, Reject

```
Return(Int(1))
```

- We want our `approval_program()` function to return the `Seq`
  - Every argument of the `Seq` must be a PyTeal expression
- `Return()` is the PyTeal expression for the `return` opcode
- Helper methods
  - `Approve()` == `Return(Int(1))`
  - `Reject()` == `Return(Int(0))`

Program Flow - If

```
from pyteal import *

def approval_program():
    return If(Int(1), Log(Bytes("Expression is true!")), Log(Bytes("Expression is false!")))

print(compileTeal(approval_program(), mode=Mode.Application, version=7))
```

Program Flow - Cond

```
from pyteal import *

def approval_program():
    option = Txn.application_args[0]
    return Cond(
        [option == Bytes("A"), Log(Bytes("Option A was selected!"))],
        [option == Bytes("B"), Log(Bytes("Option B was selected!"))],
        [option == Bytes("C"), Log(Bytes("Option C was selected!"))]
        # An error will occur if option != A, B, or C
    )

print(compileTeal(approval_program(), mode=Mode.Application, version=7))
```

App Arrays

```
def approval_program():
    return Seq(
        Log(Concat(Bytes("The first account is "), Txn.accounts[1])), # Txn.accounts[0] is always sender
        Log(Concat(Bytes("The first asset ID is "), Itob(Txn.assets[0]))),
        Log(Concat(Bytes("The first app ID is "), Itob(Txn.applications[0]))),
        Approve()
    )
```

Global/Local State

```
def approval_program():
    return Seq(
        App.globalPut(Bytes("Last Caller"), Txn.sender()),
        # Note: Before using local state Txn.sender() must opt into app
        App.localPut(Txn.sender(), Bytes("Last Call"), Global.latest_timestamp()),
        Approve()
    )
```

Inner Txns

```
def axfer_to_sender():
    return Seq(
        InnerTxnBuilder.Begin(),
        InnerTxnBuilder.SetFields(
            {
                TxnField.type_enum: TxnType.AssetTransfer,
                # Generally set fee to 0 and use fee pooling to cover.
                # Doing so prevents unpredictable fees from having unintended consequences during congestion.
                TxnField.fee: Int(0)
                TxnField.receiver: Txn.sender(),
                TxnField.amount: Int(1),
                TxnField.asset_sender: Global.current_application_address(),
                TxnField.xfer_asset: Txn.assets[0],
            }
        )
    )
```

Maybe Values

```
def approval_program():
    # Some opcodes, like app_global_get_ex, return two values:
    #   1. The actual value (.value())
    #   2. Boolean to signal whether there actually is a value or not (.hasValue())
    myStatus = App.globalGetEx(Txn.applications[1], Bytes("status"))

    return Seq(
        myStatus, # the function that returns the maybeValue must be in the sequence
        Log(If(myStatus.hasValue(), myStatus.value(), Bytes("none"))),
        Approve()
    )
```

ABI Router

```
# Main router class
router = Router(
    "demo-abi", # Name of the contract
    BareCallActions( # Bare call means no arguments
        # On create only, just approve
        no_op=OnCompleteAction.create_only(Approve()),
        # Always let creator update/delete but only by the creator of this contract
        update_application=OnCompleteAction.always(Return(is_creator)),
        delete_application=OnCompleteAction.always(Return(is_creator)),
        # clear_state must be defined for clear program
        clear_state=OnCompleteAction.never(),
    ),
)
```

ABI Methods

```
@router.method
def add(a: abi.Uint64, b: abi.Uint64, *, output: abi.Uint64) -> Expr:
    # Note the use of .set() and .get()
    # The return value will be in output
    return output.set(a.get() + b.get())
```

Resources

- PyTeal on GitHub
- PyTeal Documentation
- PyTeal on Developer Portal
- Demo ABI Repository