
PyTeal

Apr 08, 2021

1	Overview	3
2	Install PyTeal	5
3	PyTeal Examples	7
4	Data Types and Constants	29
5	Arithmetic Operations	31
6	Byte Operators	35
7	Transaction Fields and Global Parameters	37
8	Cryptographic Primitives	41
9	Scratch Space	43
10	Control Flow	45
11	State Access and Manipulation	49
12	Asset Information	55
13	TEAL Versions	59
14	PyTeal Package	61
15	Indices and tables	91
	Python Module Index	93
	Index	95

PyTeal is a Python language binding for [Algorand Smart Contracts \(ASC1s\)](#).

Algorand Smart Contracts are implemented using a new language that is stack-based, called [Transaction Execution Approval Language \(TEAL\)](#). This is a non-Turing complete language that allows branch forwards but prevents recursive logic to maximize safety and performance.

However, TEAL is essentially an assembly language. With PyTeal, developers can express smart contract logic purely using Python. PyTeal provides high level, functional programming style abstractions over TEAL and does type checking at construction time.

The [User Guide](#) describes many useful features in PyTeal, and the complete documentation for every expression and operation can be found in the [PyTeal Package API documentation](#).

PyTeal **hasn't been security audited**. Use it at your own risk.

CHAPTER 1

Overview

With PyTeal, developers can easily write **Algorand Smart Contracts (ASC1s)** in Python. PyTeal supports both stateless and statefull smart contracts.

Below is an example of writing a basic stateless smart contract that allows a specific receiver to withdraw funds from an account.

```
# This example is provided for informational purposes only and has not been audited,
↳ for security.

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
```

(continues on next page)

(continued from previous page)

```
)

if __name__ == "__main__":
    program = bank_for_account(
        → "ZZAF5ARA4MEC5PVDOP64JM5O5MQST63Q2KOY2FLYFLXXD3PFSNJJBYPFZM")
    print(compileTeal(program, mode=Mode.Signature, version=3))
```

As shown in this example, the logic of smart contract is expressed using PyTeal expressions constructed in Python. PyTeal overloads Python's arithmetic operators such as `<` and `==` (more overloaded operators can be found in *Arithmetic Operations*), allowing Python developers express smart contract logic more naturally.

Lastly, `compileTeal` is called to convert an PyTeal expression to a TEAL program, consisting of a sequence of TEAL opcodes. The output of the above example is:

```
#pragma version 3
txn TypeEnum
int pay
==
global GroupSize
int 1
==
&&
txn Receiver
addr ZZAF5ARA4MEC5PVDOP64JM5O5MQST63Q2KOY2FLYFLXXD3PFSNJJBYPFZM
==
&&
txn CloseRemainderTo
global ZeroAddress
==
&&
txn RekeyTo
global ZeroAddress
==
&&
txn Fee
int 1000
<=
&&
```

CHAPTER 2

Install PyTeal

The easiest way of installing PyTeal is using `pip` :

```
$ pip3 install pyteal
```

Alternatively, choose a [distribution file](#), and run

```
$ pip3 install [file name]
```


Here are some additional PyTeal example programs:

3.1 Signature Mode

3.1.1 Atomic Swap

Atomic Swap allows the transfer of Algos from a buyer to a seller in exchange for a good or service. This is done using a *Hashed Time Locked Contract*. In this scheme, the buyer funds a TEAL account with the sale price. The buyer also picks a secret value and encodes a secure hash of this value in the TEAL program. The TEAL program will transfer its balance to the seller if the seller is able to provide the secret value that corresponds to the hash in the program. When the seller renders the good or service to the buyer, the buyer discloses the secret from the program. The seller can immediately verify the secret and withdraw the payment.

```
# This example is provided for informational purposes only and has not been audited_
↪ for security.

from pyteal import *

"""Atomic Swap"""

alice = Addr("6ZHGHH5Z5CTPCF5WCESXMGRSVK7QJETR63M3NY5FJCUYDHO57VTCMJOBGY")
bob = Addr("7Z5PW02C6LFNQFGHWKSK5H47IQP5OJW2M3HA2QPXTY3WTNP5NU2MHBW27M")
secret = Bytes("base32", "2323232323232323")
timeout = 3000

def htlc(tmpl_seller=alice,
         tmpl_buyer=bob,
         tmpl_fee=1000,
         tmpl_secret=secret,
         tmpl_hash_fn=Sha256,
         tmpl_timeout=timeout):
```

(continues on next page)

(continued from previous page)

```

    fee_cond = Txn.fee() < Int(templ_fee)
    safety_cond = And(
        Txn.type_enum() == TxnType.Payment,
        Txn.close_remainder_to() == Global.zero_address(),
        Txn.rekey_to() == Global.zero_address(),
    )

    recv_cond = And(
        Txn.receiver() == templ_seller,
        templ_hash_fn(Arg(0)) == templ_secret
    )

    esc_cond = And(
        Txn.receiver() == templ_buyer,
        Txn.first_valid() > Int(templ_timeout)
    )

    return And(
        fee_cond,
        safety_cond,
        Or(recv_cond, esc_cond)
    )

if __name__ == "__main__":
    print(compileTeal(htlc(), mode=Mode.Signature, version=2))

```

3.1.2 Split Payment

Split Payment splits payment between `templ_rcv1` and `templ_rcv2` on the ratio of `templ_ratn` / `templ_ratd`.

```

# This example is provided for informational purposes only and has not been audited_
↳ for security.

from pyteal import *

"""Split Payment"""

templ_fee = Int(1000)
templ_rcv1 = Addr("6ZHGH5Z5CTPCF5WCESXMGRSVK7QJETR63M3NY5FJCUDHO57VTCMJOBGY")
templ_rcv2 = Addr("7Z5PWO2C6LFNQFGHWKSK5H47IQP5OJW2M3HA2QPXTY3WTNP5NU2MHBW27M")
templ_own = Addr("5MK5NGBRT5RL6IGUSYDIX5P7TNNZKRVXKT6FGVI6UVK6IZAWTYQGE4RZIQ")
templ_ratn = Int(1)
templ_ratd = Int(3)
templ_min_pay = Int(1000)
templ_timeout = Int(3000)

def split(templ_fee=templ_fee,
          templ_rcv1=templ_rcv1,
          templ_rcv2=templ_rcv2,
          templ_own=templ_own,
          templ_ratn=templ_ratn,
          templ_ratd=templ_ratd,
          templ_min_pay=templ_min_pay,
          templ_timeout=templ_timeout):

```

(continues on next page)

(continued from previous page)

```

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
)

split_program = And(
    split_core,
    If(Global.group_size() == Int(2),
        split_transfer,
        split_close
    )
)

return split_program

if __name__ == "__main__":
    print(compileTeal(split(), mode=Mode.Signature, version=2))

```

3.1.3 Periodic Payment

Periodic Payment allows some account to execute periodic withdrawal of funds. This PyTeal program creates an contract account that allows `tmpl_rcv` to withdraw `tmpl_amt` every `tmpl_period` rounds for `tmpl_dur` after every multiple of `tmpl_period`.

After `tmpl_timeout`, all remaining funds in the escrow are available to `tmpl_rcv`.

```

# This example is provided for informational purposes only and has not been audited.
↪for security.

from pyteal import *

"""Periodic Payment"""

tmpl_fee = Int(1000)
tmpl_period = Int(50)
tmpl_dur = Int(5000)

```

(continues on next page)

(continued from previous page)

```

tmpl_lease = Bytes("base64", "023sdDE2")
tmpl_amt = Int(2000)
tmpl_rcv = Addr("6ZHGHH5Z5CTPCF5WCESXMGRSVK7QJETR63M3NY5FJCUYDHO57VTCMJOBGY")
tmpl_timeout = Int(30000)

def periodic_payment(tmpl_fee=tmpl_fee,
                    tmpl_period=tmpl_period,
                    tmpl_dur=tmpl_dur,
                    tmpl_lease=tmpl_lease,
                    tmpl_amt=tmpl_amt,
                    tmpl_rcv=tmpl_rcv,
                    tmpl_timeout=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)
    )

    periodic_pay_escrow = periodic_pay_core.And(periodic_pay_transfer.Or(periodic_pay_
→close))

    return periodic_pay_escrow

if __name__ == "__main__":
    print(compileTeal(periodic_payment(), mode=Mode.Signature, version=2))

```

3.2 Application Mode

3.2.1 Voting

Voting allows accounts to register and vote for arbitrary choices. Here a *choice* is any byte slice and anyone is allowed to register to vote.

This example has a configurable *registration period* defined by the global state `RegBegin` and `RegEnd` which restrict when accounts can register to vote. There is also a separate configurable *voting period* defined by the global state `VotingBegin` and `VotingEnd` which restrict when voting can take place.

An account must register in order to vote. Accounts cannot vote more than once, and if an account opts out of the application before the voting period has concluded, their vote is discarded. The results are visible in the global state of the application, and the winner is the candidate with the highest number of votes.

```
# This example is provided for informational purposes only and has not been audited,
↳ for security.

from pyteal import *

def approval_program():
    on_creation = Seq([
        App.globalPut(Bytes("Creator"), Txn.sender()),
        Assert(Txn.application_args.length() == Int(4)),
        App.globalPut(Bytes("RegBegin"), Btoi(Txn.application_args[0])),
        App.globalPut(Bytes("RegEnd"), Btoi(Txn.application_args[1])),
        App.globalPut(Bytes("VoteBegin"), Btoi(Txn.application_args[2])),
        App.globalPut(Bytes("VoteEnd"), Btoi(Txn.application_args[3])),
        Return(Int(1))
    ])

    is_creator = Txn.sender() == App.globalGet(Bytes("Creator"))

    get_vote_of_sender = App.localGetEx(Int(0), App.id(), Bytes("voted"))

    on_closeout = Seq([
        get_vote_of_sender,
        If(And(Global.round() <= App.globalGet(Bytes("VoteEnd")), get_vote_of_sender.
↳ hasValue()),
            App.globalPut(get_vote_of_sender.value(), App.globalGet(get_vote_of_
↳ sender.value()) - Int(1))
        ),
        Return(Int(1))
    ])

    on_register = Return(And(
        Global.round() >= App.globalGet(Bytes("RegBegin")),
        Global.round() <= App.globalGet(Bytes("RegEnd"))
    ))

    choice = Txn.application_args[1]
    choice_tally = App.globalGet(choice)
    on_vote = Seq([
        Assert(And(
            Global.round() >= App.globalGet(Bytes("VoteBegin")),
            Global.round() <= App.globalGet(Bytes("VoteEnd"))
        )),
        get_vote_of_sender,
        If(get_vote_of_sender.hasValue(),
            Return(Int(0))
        ),
        App.globalPut(choice, choice_tally + Int(1)),
        App.localPut(Int(0), Bytes("voted"), choice),
        Return(Int(1))
    ])

    program = Cond(
        [Txn.application_id() == Int(0), on_creation],
        [Txn.on_completion() == OnComplete.DeleteApplication, Return(is_creator)],
```

(continues on next page)

(continued from previous page)

```

        [Txn.on_completion() == OnComplete.UpdateApplication, Return(is_creator)],
        [Txn.on_completion() == OnComplete.CloseOut, on_closeout],
        [Txn.on_completion() == OnComplete.OptIn, on_register],
        [Txn.application_args[0] == Bytes("vote"), on_vote]
    )

    return program

def clear_state_program():
    get_vote_of_sender = App.localGetEx(Int(0), App.id(), Bytes("voted"))
    program = Seq([
        get_vote_of_sender,
        If(And(Global.round() <= App.globalGet(Bytes("VoteEnd")), get_vote_of_sender.
↪hasValue()),
            App.globalPut(get_vote_of_sender.value(), App.globalGet(get_vote_of_
↪sender.value()) - Int(1))
        ),
        Return(Int(1))
    ])

    return program

if __name__ == "__main__":
    with open('vote_approval.teal', 'w') as f:
        compiled = compileTeal(approval_program(), mode=Mode.Application, version=2)
        f.write(compiled)

    with open('vote_clear_state.teal', 'w') as f:
        compiled = compileTeal(clear_state_program(), mode=Mode.Application, ↪
↪version=2)
        f.write(compiled)

```

A reference script that deploys the voting application is below:

```

# based off https://github.com/algorand/docs/blob/
↪cdf11d48a4b1168752e6ccaf77c8b9e8e599713a/examples/smart_contracts/v2/python/
↪stateful_smart_contracts.py

import base64
import datetime

from algosdk.future import transaction
from algosdk import account, mnemonic
from algosdk.v2client import algod
from pyteal import compileTeal, Mode
from vote import approval_program, clear_state_program

# user declared account mnemonics
creator_mnemonic = "Your 25-word mnemonic goes here"
user_mnemonic = "A second distinct 25-word mnemonic goes here"

# user declared algod connection parameters. Node must have EnableDeveloperAPI set to ↪
↪true in its config
algod_address = "http://localhost:4001"
algod_token = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"

# helper function to compile program source

```

(continues on next page)

(continued from previous page)

```

def compile_program(client, source_code):
    compile_response = client.compile(source_code)
    return base64.b64decode(compile_response['result'])

# helper function that converts a mnemonic passphrase into a private signing key
def get_private_key_from_mnemonic(mn):
    private_key = mnemonic.to_private_key(mn)
    return private_key

# helper function that waits for a given txid to be confirmed by the network
def wait_for_confirmation(client, txid):
    last_round = client.status().get('last-round')
    txinfo = client.pending_transaction_info(txid)
    while not (txinfo.get('confirmed-round') and txinfo.get('confirmed-round') > 0):
        print("Waiting for confirmation...")
        last_round += 1
        client.status_after_block(last_round)
        txinfo = client.pending_transaction_info(txid)
    print("Transaction {} confirmed in round {}".format(txid, txinfo.get('confirmed-
↳round')))
    return txinfo

def wait_for_round(client, round):
    last_round = client.status().get('last-round')
    print(f"Waiting for round {round}")
    while last_round < round:
        last_round += 1
        client.status_after_block(last_round)
        print(f"Round {last_round}")

# create new application
def create_app(client, private_key, approval_program, clear_program, global_schema,
↳local_schema, app_args):
    # define sender as creator
    sender = account.address_from_private_key(private_key)

    # declare on_complete as NoOp
    on_complete = transaction.OnComplete.NoOpOC.real

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationCreateTxn(sender, params, on_complete, \
                                           approval_program, clear_program, \
                                           global_schema, local_schema, app_args)

    # sign transaction
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()

    # send transaction
    client.send_transactions([signed_txn])

```

(continues on next page)

(continued from previous page)

```

    # await confirmation
    wait_for_confirmation(client, tx_id)

    # display results
    transaction_response = client.pending_transaction_info(tx_id)
    app_id = transaction_response['application-index']
    print("Created new app-id:", app_id)

    return app_id

# opt-in to application
def opt_in_app(client, private_key, index):
    # declare sender
    sender = account.address_from_private_key(private_key)
    print("OptIn from account: ", sender)

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationOptInTxn(sender, params, index)

    # sign transaction
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()

    # send transaction
    client.send_transactions([signed_txn])

    # await confirmation
    wait_for_confirmation(client, tx_id)

    # display results
    transaction_response = client.pending_transaction_info(tx_id)
    print("OptIn to app-id:", transaction_response['txn']['txn']['apid'])

# call application
def call_app(client, private_key, index, app_args):
    # declare sender
    sender = account.address_from_private_key(private_key)
    print("Call from account:", sender)

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationNoOpTxn(sender, params, index, app_args)

    # sign transaction
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()

```

(continues on next page)

(continued from previous page)

```

    # send transaction
    client.send_transactions([signed_txn])

    # await confirmation
    wait_for_confirmation(client, tx_id)

def format_state(state):
    formatted = {}
    for item in state:
        key = item['key']
        value = item['value']
        formatted_key = base64.b64decode(key).decode('utf-8')
        if value['type'] == 1:
            # byte string
            if formatted_key == 'voted':
                formatted_value = base64.b64decode(value['bytes']).decode('utf-8')
            else:
                formatted_value = value['bytes']
            formatted[formatted_key] = formatted_value
        else:
            # integer
            formatted[formatted_key] = value['uint']
    return formatted

# read user local state
def read_local_state(client, addr, app_id):
    results = client.account_info(addr)
    for local_state in results['apps-local-state']:
        if local_state['id'] == app_id:
            if 'key-value' not in local_state:
                return {}
            return format_state(local_state['key-value'])
    return {}

# read app global state
def read_global_state(client, addr, app_id):
    results = client.account_info(addr)
    apps_created = results['created-apps']
    for app in apps_created:
        if app['id'] == app_id:
            return format_state(app['params']['global-state'])
    return {}

# delete application
def delete_app(client, private_key, index):
    # declare sender
    sender = account.address_from_private_key(private_key)

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationDeleteTxn(sender, params, index)

```

(continues on next page)

(continued from previous page)

```

    # sign transaction
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()

    # send transaction
    client.send_transactions([signed_txn])

    # await confirmation
    wait_for_confirmation(client, tx_id)

    # display results
    transaction_response = client.pending_transaction_info(tx_id)
    print("Deleted app-id:", transaction_response['txn']['txn']['apid'])

# close out from application
def close_out_app(client, private_key, index):
    # declare sender
    sender = account.address_from_private_key(private_key)

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationCloseOutTxn(sender, params, index)

    # sign transaction
    signed_txn = txn.sign(private_key)
    tx_id = signed_txn.transaction.get_txid()

    # send transaction
    client.send_transactions([signed_txn])

    # await confirmation
    wait_for_confirmation(client, tx_id)

    # display results
    transaction_response = client.pending_transaction_info(tx_id)
    print("Closed out from app-id: ", transaction_response['txn']['txn']['apid'])

# clear application
def clear_app(client, private_key, index):
    # declare sender
    sender = account.address_from_private_key(private_key)

    # get node suggested parameters
    params = client.suggested_params()
    # comment out the next two (2) lines to use suggested fees
    params.flat_fee = True
    params.fee = 1000

    # create unsigned transaction
    txn = transaction.ApplicationClearStateTxn(sender, params, index)

```

(continues on next page)

(continued from previous page)

```

# sign transaction
signed_txn = txn.sign(private_key)
tx_id = signed_txn.transaction.get_txid()

# send transaction
client.send_transactions([signed_txn])

# await confirmation
wait_for_confirmation(client, tx_id)

# display results
transaction_response = client.pending_transaction_info(tx_id)
print("Cleared app-id:", transaction_response['txn']['txn']['apid'])

# convert 64 bit integer i to byte string
def intToBytes(i):
    return i.to_bytes(8, "big")

def main():
    # initialize an algodClient
    algod_client = algod.AlgodClient(algod_token, algod_address)

    # define private keys
    creator_private_key = get_private_key_from_mnemonic(creator_mnemonic)
    user_private_key = get_private_key_from_mnemonic(user_mnemonic)

    # declare application state storage (immutable)
    local_ints = 0
    local_bytes = 1
    global_ints = 24 # 4 for setup + 20 for choices. Use a larger number for more_
    ↪choices.
    global_bytes = 1
    global_schema = transaction.StateSchema(global_ints, global_bytes)
    local_schema = transaction.StateSchema(local_ints, local_bytes)

    # get PyTeal approval program
    approval_program_ast = approval_program()
    # compile program to TEAL assembly
    approval_program_teal = compileTeal(approval_program_ast, mode=Mode.Application,
    ↪version=2)
    # compile program to binary
    approval_program_compiled = compile_program(algod_client, approval_program_teal)

    # get PyTeal clear state program
    clear_state_program_ast = clear_state_program()
    # compile program to TEAL assembly
    clear_state_program_teal = compileTeal(clear_state_program_ast, mode=Mode.
    ↪Application, version=2)
    # compile program to binary
    clear_state_program_compiled = compile_program(algod_client, clear_state_program_
    ↪teal)

    # configure registration and voting period
    status = algod_client.status()
    regBegin = status['last-round'] + 10
    regEnd = regBegin + 10
    voteBegin = regEnd + 1

```

(continues on next page)

(continued from previous page)

```

voteEnd = voteBegin + 10

print(f"Registration rounds: {regBegin} to {regEnd}")
print(f"Vote rounds: {voteBegin} to {voteEnd}")

# create list of bytes for app args
app_args = [
    intToBytes(regBegin),
    intToBytes(regEnd),
    intToBytes(voteBegin),
    intToBytes(voteEnd)
]

# create new application
app_id = create_app(algod_client, creator_private_key, approval_program_compiled,
↳clear_state_program_compiled, global_schema, local_schema, app_args)

# read global state of application
print("Global state:", read_global_state(algod_client, account.address_from_
↳private_key(creator_private_key), app_id))

# wait for registration period to start
wait_for_round(algod_client, regBegin)

# opt-in to application
opt_in_app(algod_client, user_private_key, app_id)

wait_for_round(algod_client, voteBegin)

# call application without arguments
call_app(algod_client, user_private_key, app_id, [b'vote', b'choiceA'])

# read local state of application from user account
print("Local state:", read_local_state(algod_client, account.address_from_private_
↳key(user_private_key), app_id))

# wait for registration period to start
wait_for_round(algod_client, voteEnd)

# read global state of application
global_state = read_global_state(algod_client, account.address_from_private_
↳key(creator_private_key), app_id)
print("Global state:", global_state)

max_votes = 0
max_votes_choice = None
for key,value in global_state.items():
    if key not in ('RegBegin', 'RegEnd', 'VoteBegin', 'VoteEnd', 'Creator') and
↳isinstance(value, int):
        if value > max_votes:
            max_votes = value
            max_votes_choice = key

print("The winner is:", max_votes_choice)

# delete application
delete_app(algod_client, creator_private_key, app_id)

```

(continues on next page)

(continued from previous page)

```

# clear application from user account
clear_app(algod_client, user_private_key, app_id)

if __name__ == "__main__":
    main()

```

Example output for deployment would be:

```

Registration rounds: 592 to 602
Vote rounds: 603 to 613
Waiting for confirmation...
Transaction KXJHR6J4QSCAH036L77DPJ53CLZBCCSPSBAOGTGQDRA7WECDXUEA confirmed in round_
↳584.
Created new app-id: 29
Global state: {'RegEnd': 602, 'VoteBegin': 603, 'VoteEnd': 613, 'Creator':
↳'49y8gDrKSnm77cgRyFzYdlkw18SDVnKhhoiS6NVVH8U=', 'RegBegin': 592}
Waiting for round 592
Round 585
Round 586
Round 587
Round 588
Round 589
Round 590
Round 591
Round 592
OptIn from account: FVQEFNOSD25TDBTTIU2I5KW5DHR6PADYMZESTOCQ2O3ME4OWXEI7OHVRY
Waiting for confirmation...
Transaction YWXOAREFSUYID6QLWQHANTXK3NR2XOVTIQYKMD27F3VXJKP7CMYQ confirmed in round_
↳595.
OptIn to app-id: 29
Waiting for round 603
Round 596
Round 597
Round 598
Round 599
Round 600
Round 601
Round 602
Round 603
Call from account: FVQEFNOSD25TDBTTIU2I5KW5DHR6PADYMZESTOCQ2O3ME4OWXEI7OHVRY
Waiting for confirmation...
Transaction WNV4DTPEMVGUXNRZHMWN SCUU7AQJOCFTBKJT6NV2KN6THT4QGKNQ confirmed in round_
↳606.
Local state: {'voted': 'choiceA'}
Waiting for round 613
Round 607
Round 608
Round 609
Round 610
Round 611
Round 612
Round 613
Global state: {'RegBegin': 592, 'RegEnd': 602, 'VoteBegin': 603, 'VoteEnd': 613,
↳'choiceA': 1, 'Creator': '49y8gDrKSnm77cgRyFzYdlkw18SDVnKhhoiS6NVVH8U='}
The winner is: choiceA
Waiting for confirmation...

```

(continues on next page)

(continued from previous page)

```

Transaction 535KBWJ7RQX4ISV763IUUICQWI6VERYBJ7J6X7HPMAMFNKJPSNPQ confirmed in round_
↳616.
Deleted app-id: 29
Waiting for confirmation...
Transaction Z56HDAJYARUC4PWGWQLCBA6TZYQOOLNOXY5XRM3IYUEEUCT5DRMA confirmed in round_
↳618.
Cleared app-id: 29

```

3.2.2 Asset

Asset is an implementation of a custom asset type using smart contracts. While Algorand has *ASAs*, in some blockchains the only way to create a custom asset is through smart contracts.

At creation, the creator specifies the total supply of the asset. Initially this supply is placed in a reserve and the creator is made an admin. Any admin can move funds from the reserve into the balance of any account that has opted into the application using the *mint* argument. Additionally, any admin can move funds from any account's balance into the reserve using the *burn* argument.

Accounts are free to transfer funds in their balance to any other account that has opted into the application. When an account opts out of the application, their balance is added to the reserve.

```

# This example is provided for informational purposes only and has not been audited_
↳for security.

from pyteal import *

def approval_program():
    on_creation = Seq([
        Assert(Txn.application_args.length() == Int(1)),
        App.globalPut(Bytes("total supply"), Btoi(Txn.application_args[0])),
        App.globalPut(Bytes("reserve"), Btoi(Txn.application_args[0])),
        App.localPut(Int(0), Bytes("admin"), Int(1)),
        App.localPut(Int(0), Bytes("balance"), Int(0)),
        Return(Int(1))
    ])

    is_admin = App.localGet(Int(0), Bytes("admin"))

    on_closeout = Seq([
        App.globalPut(
            Bytes("reserve"),
            App.globalGet(Bytes("reserve")) + App.localGet(Int(0), Bytes("balance"))
        ),
        Return(Int(1))
    ])

    register = Seq([
        App.localPut(Int(0), Bytes("balance"), Int(0)),
        Return(Int(1))
    ])

    # configure the admin status of the account Txn.accounts[1]
    # sender must be admin
    new_admin_status = Btoi(Txn.application_args[1])
    set_admin = Seq([

```

(continues on next page)

(continued from previous page)

```

    Assert(And(is_admin, Txn.application_args.length() == Int(2))),
    App.localPut(Int(1), Bytes("admin"), new_admin_status),
    Return(Int(1))
])
# NOTE: The above set_admin code is carefully constructed. If instead we used the
↪ following code:
# Seq([
#     Assert(Txn.application_args.length() == Int(2)),
#     App.localPut(Int(1), Bytes("admin"), new_admin_status),
#     Return(is_admin)
# ])
# It would be vulnerable to the following attack: a sender passes in their own
↪ address as
# Txn.accounts[1], so then the line App.localPut(Int(1), Bytes("admin"), new_
↪ admin_status)
# changes the sender's admin status, meaning the final Return(is_admin) can
↪ return anything the
# sender wants. This allows anyone to become an admin!

# move assets from the reserve to Txn.accounts[1]
# sender must be admin
mint_amount = Btoi(Txn.application_args[1])
mint = Seq([
    Assert(Txn.application_args.length() == Int(2)),
    Assert(mint_amount <= App.globalGet(Bytes("reserve"))),
    App.globalPut(Bytes("reserve"), App.globalGet(Bytes("reserve")) - mint_
↪ amount),
    App.localPut(Int(1), Bytes("balance"), App.localGet(Int(1), Bytes("balance"))
↪ + mint_amount),
    Return(is_admin)
])

# transfer assets from the sender to Txn.accounts[1]
transfer_amount = Btoi(Txn.application_args[1])
transfer = Seq([
    Assert(Txn.application_args.length() == Int(2)),
    Assert(transfer_amount <= App.localGet(Int(0), Bytes("balance"))),
    App.localPut(Int(0), Bytes("balance"), App.localGet(Int(0), Bytes("balance"))
↪ - transfer_amount),
    App.localPut(Int(1), Bytes("balance"), App.localGet(Int(1), Bytes("balance"))
↪ + transfer_amount),
    Return(Int(1))
])

program = Cond(
    [Txn.application_id() == Int(0), on_creation],
    [Txn.on_completion() == OnComplete.DeleteApplication, Return(is_admin)],
    [Txn.on_completion() == OnComplete.UpdateApplication, Return(is_admin)],
    [Txn.on_completion() == OnComplete.CloseOut, on_closeout],
    [Txn.on_completion() == OnComplete.OptIn, register],
    [Txn.application_args[0] == Bytes("set admin"), set_admin],
    [Txn.application_args[0] == Bytes("mint"), mint],
    [Txn.application_args[0] == Bytes("transfer"), transfer]
)

return program

```

(continues on next page)

(continued from previous page)

```

def clear_state_program():
    program = Seq([
        App.globalPut(
            Bytes("reserve"),
            App.globalGet(Bytes("reserve")) + App.localGet(Int(0), Bytes("balance"))
        ),
        Return(Int(1))
    ])

    return program

if __name__ == "__main__":
    with open('asset_approval.teal', 'w') as f:
        compiled = compileTeal(approval_program(), mode=Mode.Application, version=2)
        f.write(compiled)

    with open('asset_clear_state.teal', 'w') as f:
        compiled = compileTeal(clear_state_program(), mode=Mode.Application,
↪version=2)
        f.write(compiled)

```

3.2.3 Security Token

Security Token is an extension of the *Asset* example with more features and restrictions. There are two types of admins, *contract admins* and *transfer admins*.

Contract admins can delete the smart contract if the entire supply is in the reserve. They can promote accounts to transfer or contract admins. They can also *mint* and *burn* funds.

Transfer admins can impose maximum balance limitations on accounts, temporarily lock accounts, assign accounts to transfer groups, and impose transaction restrictions between transaction groups.

Both contract and transfer admins can pause trading of funds and freeze individual accounts.

Accounts can only transfer funds if trading is not paused, both the sender and receive accounts are not frozen or temporarily locked, transfer group restrictions are not in place between them, and the receiver's account does not have a maximum balance restriction that would be invalidated.

```

# This example is provided for informational purposes only and has not been audited_
↪for security.

from pyteal import *

def approval_program():
    on_creation = Seq([
        Assert(Txn.application_args.length() == Int(1)),
        App.globalPut(Bytes("total supply"), Btoi(Txn.application_args[0])),
        App.globalPut(Bytes("reserve"), Btoi(Txn.application_args[0])),
        App.globalPut(Bytes("paused"), Int(0)),
        App.localPut(Int(0), Bytes("contract admin"), Int(1)),
        App.localPut(Int(0), Bytes("transfer admin"), Int(1)),
        App.localPut(Int(0), Bytes("balance"), Int(0)),
        Return(Int(1))
    ])

    is_contract_admin = App.localGet(Int(0), Bytes("contract admin"))

```

(continues on next page)

(continued from previous page)

```

is_transfer_admin = App.localGet(Int(0), Bytes("transfer admin"))
is_any_admin = is_contract_admin.Or(is_transfer_admin)

can_delete = And(
    is_contract_admin,
    App.globalGet(Bytes("total supply")) == App.globalGet(Bytes("reserve"))
)

on_closeout = Seq([
    App.globalPut(
        Bytes("reserve"),
        App.globalGet(Bytes("reserve")) + App.localGet(Int(0), Bytes("balance"))
    ),
    Return(Int(1))
])

register = Seq([
    App.localPut(Int(0), Bytes("balance"), Int(0)),
    Return(Int(1))
])

# pause all transfers
# sender must be any admin
new_pause_value = Btoi(Txn.application_args[1])
pause = Seq([
    Assert(Txn.application_args.length() == Int(2)),
    App.globalPut(Bytes("paused"), new_pause_value),
    Return(is_any_admin)
])

# configure the admin status of the account Txn.accounts[1]
# sender must be contract admin
new_admin_type = Txn.application_args[1]
new_admin_status = Btoi(Txn.application_args[2])
set_admin = Seq([
    Assert(And(
        is_contract_admin,
        Txn.application_args.length() == Int(3),
        Or(new_admin_type == Bytes("contract admin"), new_admin_type == Bytes(
↪ "transfer admin")),
        Txn.accounts.length() == Int(1)
    )),
    App.localPut(Int(1), new_admin_type, new_admin_status),
    Return(Int(1))
])

# NOTE: The above set_admin code is carefully constructed. If instead we used the
↪ following code:
    # Seq([
    #     Assert(And(
    #         Txn.application_args.length() == Int(3),
    #         Or(new_admin_type == Bytes("contract admin"), new_admin_type == Bytes(
↪ "transfer admin")),
    #         Txn.accounts.length() == Int(1)
    #     )),
    #     App.localPut(Int(1), new_admin_type, new_admin_status),
    #     Return(is_contract_admin)
    # ])

```

(continues on next page)

(continued from previous page)

```

    # It would be vulnerable to the following attack: a sender passes in their own_
    ↪address as
    # Txn.accounts[1], so then the line App.localPut(Int(1), new_admin_type, new_
    ↪admin_status)
    # changes the sender's admin status, meaning the final Return(is_contract_admin)_
    ↪can return
    # anything the sender wants. This allows anyone to become an admin!

    # freeze Txn.accounts[1]
    # sender must be any admin
    new_freeze_value = Btoi(Txn.application_args[1])
    freeze = Seq([
        Assert(And(
            Txn.application_args.length() == Int(2),
            Txn.accounts.length() == Int(1)
        )),
        App.localPut(Int(1), Bytes("frozen"), new_freeze_value),
        Return(is_any_admin)
    ])

    # modify the max balance of Txn.accounts[1]
    # if max_balance_value is 0, will delete the existing max balance limitation on_
    ↪the account
    # sender must be transfer admin
    max_balance_value = Btoi(Txn.application_args[1])
    max_balance = Seq([
        Assert(And(
            Txn.application_args.length() == Int(2),
            Txn.accounts.length() == Int(1)
        )),
        If(max_balance_value == Int(0),
            App.localDel(Int(1), Bytes("max balance")),
            App.localPut(Int(1), Bytes("max balance"), max_balance_value)
        ),
        Return(is_transfer_admin)
    ])

    # lock Txn.accounts[1] until a UNIX timestamp
    # sender must be transfer admin
    lock_until_value = Btoi(Txn.application_args[1])
    lock_until = Seq([
        Assert(And(
            Txn.application_args.length() == Int(2),
            Txn.accounts.length() == Int(1)
        )),
        If(lock_until_value == Int(0),
            App.localDel(Int(1), Bytes("lock until")),
            App.localPut(Int(1), Bytes("lock until"), lock_until_value)
        ),
        Return(is_transfer_admin)
    ])

    set_transfer_group = Seq([
        Assert(And(
            Txn.application_args.length() == Int(3),
            Txn.accounts.length() == Int(1)
        )),

```

(continues on next page)

(continued from previous page)

```

    App.localPut(Int(1), Bytes("transfer_group"), Btoi(Txn.application_args[2]))
})

def getRuleKey(sendGroup, receiveGroup):
    return Concat(Bytes("rule"), Itob(sendGroup), Itob(receiveGroup))

lock_transfer_key = getRuleKey(Btoi(Txn.application_args[2]), Btoi(Txn.
↪application_args[3]))
lock_transfer_until = Btoi(Txn.application_args[4])
lock_transfer_group = Seq([
    Assert(Txn.application_args.length() == Int(5)),
    If(lock_transfer_until == Int(0),
        App.globalDel(lock_transfer_key),
        App.globalPut(lock_transfer_key, lock_transfer_until)
    )
])

# sender must be transfer admin
transfer_group = Seq([
    Assert(Txn.application_args.length() > Int(2)),
    Cond(
        [Txn.application_args[1] == Bytes("set"), set_transfer_group],
        [Txn.application_args[1] == Bytes("lock"), lock_transfer_group]
    ),
    Return(is_transfer_admin)
])

# move assets from the reserve to Txn.accounts[1]
# sender must be contract admin
mint_amount = Btoi(Txn.application_args[1])
mint = Seq([
    Assert(And(
        Txn.application_args.length() == Int(2),
        Txn.accounts.length() == Int(1),
        mint_amount <= App.globalGet(Bytes("reserve"))
    )),
    App.globalPut(Bytes("reserve"), App.globalGet(Bytes("reserve")) - mint_
↪amount),
    App.localPut(Int(1), Bytes("balance"), App.localGet(Int(1), Bytes("balance"))_
↪+ mint_amount),
    Return(is_contract_admin)
])

# move assets from Txn.accounts[1] to the reserve
# sender must be contract admin
burn_amount = Btoi(Txn.application_args[1])
burn = Seq([
    Assert(And(
        Txn.application_args.length() == Int(2),
        Txn.accounts.length() == Int(1),
        burn_amount <= App.localGet(Int(1), Bytes("balance"))
    )),
    App.globalPut(Bytes("reserve"), App.globalGet(Bytes("reserve")) + burn_
↪amount),
    App.localPut(Int(1), Bytes("balance"), App.localGet(Int(1), Bytes("balance"))_
↪- burn_amount),
    Return(is_contract_admin)
])

```

(continues on next page)

(continued from previous page)

```

    ])

    # transfer assets from the sender to Txn.accounts[1]
    transfer_amount = Btoi(Txn.application_args[1])
    receiver_max_balance = App.localGetEx(Int(1), App.id(), Bytes("max balance"))
    transfer = Seq([
        Assert(And(
            Txn.application_args.length() == Int(2),
            Txn.accounts.length() == Int(1),
            transfer_amount <= App.localGet(Int(0), Bytes("balance"))
        )),
        receiver_max_balance,
        If(
            Or(
                App.globalGet(Bytes("paused")),
                App.localGet(Int(0), Bytes("frozen")),
                App.localGet(Int(1), Bytes("frozen")),
                App.localGet(Int(0), Bytes("lock until")) >= Global.latest_
↪timestamp(),
                App.localGet(Int(1), Bytes("lock until")) >= Global.latest_
↪timestamp(),
                App.globalGet(getRuleKey(App.localGet(Int(0), Bytes("transfer group
↪")), App.localGet(Int(1), Bytes("transfer group"))) >= Global.latest_timestamp(),
                And(
                    receiver_max_balance.hasValue(),
                    receiver_max_balance.value() < App.localGet(Int(1), Bytes("balance
↪")) + transfer_amount
                )
            ),
            Return(Int(0))
        ),
        App.localPut(Int(0), Bytes("balance"), App.localGet(Int(0), Bytes("balance"))_
↪- transfer_amount),
        App.localPut(Int(1), Bytes("balance"), App.localGet(Int(1), Bytes("balance"))_
↪+ transfer_amount),
        Return(Int(1))
    ])

    program = Cond(
        [Txn.application_id() == Int(0), on_creation],
        [Txn.on_completion() == OnComplete.DeleteApplication, Return(can_delete)],
        [Txn.on_completion() == OnComplete.UpdateApplication, Return(is_contract_
↪admin)],
        [Txn.on_completion() == OnComplete.CloseOut, on_closeout],
        [Txn.on_completion() == OnComplete.OptIn, register],
        [Txn.application_args[0] == Bytes("pause"), pause],
        [Txn.application_args[0] == Bytes("set admin"), set_admin],
        [Txn.application_args[0] == Bytes("freeze"), freeze],
        [Txn.application_args[0] == Bytes("max balance"), max_balance],
        [Txn.application_args[0] == Bytes("lock until"), lock_until],
        [Txn.application_args[0] == Bytes("transfer group"), transfer_group],
        [Txn.application_args[0] == Bytes("mint"), mint],
        [Txn.application_args[0] == Bytes("burn"), burn],
        [Txn.application_args[0] == Bytes("transfer"), transfer],
    )

    return program

```

(continues on next page)

(continued from previous page)

```
def clear_state_program():
    program = Seq([
        App.globalPut(
            Bytes("reserve"),
            App.globalGet(Bytes("reserve")) + App.localGet(Int(0), Bytes("balance"))
        ),
        Return(Int(1))
    ])

    return program

if __name__ == "__main__":
    with open('security_token_approval.teal', 'w') as f:
        compiled = compileTeal(approval_program(), mode=Mode.Application, version=2)
        f.write(compiled)

    with open('security_token_clear_state.teal', 'w') as f:
        compiled = compileTeal(clear_state_program(), mode=Mode.Application,
↪version=2)
        f.write(compiled)
```

Data Types and Constants

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

- `TealType.uint64`, 64 bit unsigned integer
- `TealType.bytes`, a slice of bytes

For example, all the transaction arguments (e.g. `Arg(0)`) are of type `TealType.bytes`. The first valid round of current transaction (`Txn.first_valid()`) is typed `TealType.uint64`.

4.1 Integers

`Int(n)` creates a `TealType.uint64` constant, where $n \geq 0$ and $n < 2^{64}$.

4.2 Bytes

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

4.2.1 UTF-8

Byte slices can be created from UTF-8 encoded strings. For example:

```
Bytes("hello world")
```

4.2.2 Base16

Byte slices can be created from a [RFC 4648#section-8](#) base16 encoded binary string, e.g. `"0xA21212EF"` or `"A21212EF"`. For example:

```
Bytes("base16", "0xA21212EF")
Bytes("base16", "A21212EF") # "0x" is optional
```

4.2.3 Base32

Byte slices can be created from a [RFC 4648#section-6](#) base32 encoded binary string with or without padding, e.g. "7Z5PWO2C6LFNQFGHWKSK5H47IQP5OJW2M3HA2QPXTY3WTNP5NU2MHBW27M".

```
Bytes("base32", "7Z5PWO2C6LFNQFGHWKSK5H47IQP5OJW2M3HA2QPXTY3WTNP5NU2MHBW27M")
```

4.2.4 Base64

Byte slices can be created from a [RFC 4648#section-4](#) base64 encoded binary string, e.g. "Zm9vYmE=".

```
Bytes("base64", "Zm9vYmE=")
```

4.3 Type Checking

All PyTeal expressions are type checked at construction time, for example, running the following code triggers a `TealTypeError`:

```
Int(0) < Arg(0)
```

Since `<` (overloaded Python operator, see [Arithmetic Operations](#) for more details) requires both operands of type `TealType.uint64`, while `Arg(0)` is of type `TealType.bytes`.

4.4 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 `"\x00\x00\x00\x00\x00\x00\x00\x01"` not the string `"1"`.

Arithmetic Operations

An arithmetic expression is an expression that results in a `TealType.uint64` value. In PyTeal, arithmetic expressions include integer and boolean operators (booleans are the integers `0` or `1`). The table below summarized all arithmetic expressions in PyTeal.

Operator	Overloaded	Semantics	Example
<i>Lt</i> (<i>a</i> , <i>b</i>)	<i>a</i> < <i>b</i>	<i>1</i> if <i>a</i> is less than <i>b</i> , <i>0</i> otherwise	<code>Int(1) < Int(5)</code>
<i>Gt</i> (<i>a</i> , <i>b</i>)	<i>a</i> > <i>b</i>	<i>1</i> if <i>a</i> is greater than <i>b</i> , <i>0</i> otherwise	<code>Int(1) > Int(5)</code>
<i>Le</i> (<i>a</i> , <i>b</i>)	<i>a</i> <= <i>b</i>	<i>1</i> if <i>a</i> is no greater than <i>b</i> , <i>0</i> otherwise	<code>Int(1) <= Int(5)</code>
<i>Ge</i> (<i>a</i> , <i>b</i>)	<i>a</i> >= <i>b</i>	<i>1</i> if <i>a</i> is no less than <i>b</i> , <i>0</i> otherwise	<code>Int(1) >= Int(5)</code>
<i>Add</i> (<i>a</i> , <i>b</i>)	<i>a</i> + <i>b</i>	<i>a</i> + <i>b</i> , error (panic) if overflow	<code>Int(1) + Int(5)</code>
<i>Minus</i> (<i>a</i> , <i>b</i>)	<i>a</i> - <i>b</i>	<i>a</i> - <i>b</i> , error if underflow	<code>Int(5) - Int(1)</code>
<i>Mul</i> (<i>a</i> , <i>b</i>)	<i>a</i> * <i>b</i>	<i>a</i> * <i>b</i> , error if overflow	<code>Int(2) * Int(3)</code>
<i>Div</i> (<i>a</i> , <i>b</i>)	<i>a</i> / <i>b</i>	<i>a</i> / <i>b</i> , error if divided by zero	<code>Int(3) / Int(2)</code>
<i>Mod</i> (<i>a</i> , <i>b</i>)	<i>a</i> % <i>b</i>	<i>a</i> % <i>b</i> , modulo operation	<code>Int(7) % Int(3)</code>
<i>Eq</i> (<i>a</i> , <i>b</i>)	<i>a</i> == <i>b</i>	<i>1</i> if <i>a</i> equals <i>b</i> , <i>0</i> otherwise	<code>Int(7) == Int(7)</code>
<i>Neq</i> (<i>a</i> , <i>b</i>)	<i>a</i> != <i>b</i>	<i>0</i> if <i>a</i> equals <i>b</i> , <i>1</i> otherwise	<code>Int(7) != Int(7)</code>
<i>And</i> (<i>a</i> , <i>b</i>)		<i>1</i> if <i>a</i> > 0 && <i>b</i> > 0, <i>0</i> otherwise	<code>And(Int(1), Int(1))</code>
<i>Or</i> (<i>a</i> , <i>b</i>)		<i>1</i> if <i>a</i> > 0 <i>b</i> > 0, <i>0</i> otherwise	<code>Or(Int(1), Int(0))</code>
<i>Not</i> (<i>a</i>)		<i>1</i> if <i>a</i> equals <i>0</i> , <i>0</i> otherwise	<code>Not(Int(0))</code>
<i>BitwiseAnd</i> (<i>a</i> , <i>b</i>)	<i>a</i> & <i>b</i>	<i>a</i> & <i>b</i> , bitwise and operation	<code>Int(1) & Int(3)</code>
<i>BitwiseOr</i> (<i>a</i> , <i>b</i>)	<i>a</i> <i>b</i>	<i>a</i> <i>b</i> , bitwise or operation	<code>Int(2) Int(5)</code>
<i>BitwiseXor</i> (<i>a</i> , <i>b</i>)	<i>a</i> ^ <i>b</i>	<i>a</i> ^ <i>b</i> , bitwise xor operation	<code>Int(3) ^ Int(7)</code>
<i>BitwiseNot</i> (<i>a</i>)	~ <i>a</i>	~ <i>a</i> , bitwise complement operation	<code>~Int(1)</code>

Most of the above operations take two `TealType.uint64` values as inputs. In addition, `Eq(a, b)` (`==`) and `Neq(a, b)` (`!=`) also work for byte slices. For example, `Arg(0) == Arg(1)` and `Arg(0) != Arg(1)` are valid PyTeal expressions.

Both `And` and `Or` also support more than 2 arguments when called as functions:

- `And(a, b, ...)`
- `Or(a, b, ...)`

The associativity and precedence of the overloaded Python arithmetic operators are the same as the [original python operators](#) . For example:

- `Int(1) + Int(2) + Int(3)` is equivalent to `Add(Add(Int(1), Int(2)), Int(3))`
- `Int(1) + Int(2) * Int(3)` is equivalent to `Add(Int(1), Mul(Int(2), Int(3)))`

5.1 Bit and Byte Operations

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:

- For integers, bit indexing begins with low-order bits. For example, the bit at index 4 of the integer 16 (000...0001000 in binary) is 1. Every other index has a bit value of 0. Any index less than 64 is valid, regardless of the integer's value.

Byte indexing is not supported for integers.

- For byte strings, bit indexing begins at the first bit. For example, the bit at index 0 of the base16 byte string 0xf0 (11110000 in binary) is 1. Any index less than 4 has a bit value of 1, and any index 4 or greater has a bit value of 0. Any index less than 8 times the length of the byte string is valid.

Likewise, byte indexing begins at the first byte of the string. For example, the byte at index 0 of that the base16 string 0xff00 (1111111100000000 in binary) is 255 (11111111 in binary), and the byte at index 1 is 0. Any index less than the length of the byte string is valid.

5.1.1 Bit Manipulation

The *GetBit* expression can extract individual bit values from integers and byte strings. For 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
```

Additionally, the *SetBit* expression can modify individual bit values from integers and byte strings. For example,

```
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
```

5.1.2 Byte Manipulation

In addition to manipulating bits, individual bytes in byte strings can be manipulated.

The `GetByte` expression can extract individual bytes from byte strings. For example,

```
GetByte(Bytes("base16", "0xff00"), Int(0)) # get the 0th byte of 0xff00, produces 255
GetByte(Bytes("base16", "0xff00"), Int(1)) # get the 1st byte of 0xff00, produces 0
GetByte(Bytes("base16", "0xff00"), Int(2)) # get the 2nd byte of 0xff00, invalid index

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')
```

Additionally, the `SetByte` expression can modify individual bytes in byte strings. For example,

```
SetByte(Bytes("base16", "0xff00"), Int(0), Int(0)) # set the 0th byte of 0xff00 to 0,
↳ produces 0x0000
SetByte(Bytes("base16", "0xff00"), Int(0), Int(128)) # set the 0th byte of 0xff00 to
↳ 128, produces 0x8000

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


TEAL byte slices are similar to strings and can be manipulated in the same way.

6.1 Length

The length of a byte slice can be obtained using the *Len* expression. For example:

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

6.2 Concatenation

Byte slices can be combined using the *Concat* expression. This expression takes at least two arguments and produces a new byte slice consisting of each argument, one after another. For example:

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

6.3 Substring Extraction

Byte slices can be extracted from other byte slices using the *Substring* expression. This expression can extract part of a byte slicing given start and end indices. For example:

```
Substring(Bytes("algorand"), Int(0), Int(4)) # will produce "algo"
```

6.4 Manipulating Individual Bits and Bytes

The individual bits and bytes in a byte string can be extracted and changed. See *Bit and Byte Operations* for more information.

Transaction Fields and Global Parameters

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

7.1 Transaction Fields

Information about the current transaction being evaluated can be obtained using the `Txn` object. Below are the PyTeal expressions that refer to transaction fields:

Operator	Type	Notes
<code>Txn.sender()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.fee()</code>	<code>TealType.uint64</code>	in microAlgos
<code>Txn.first_valid()</code>	<code>TealType.uint64</code>	round number
<code>Txn.last_valid()</code>	<code>TealType.uint64</code>	round number
<code>Txn.note()</code>	<code>TealType.bytes</code>	transaction note in bytes
<code>Txn.lease()</code>	<code>TealType.bytes</code>	transaction lease in bytes
<code>Txn.receiver()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.amount()</code>	<code>TealType.uint64</code>	in microAlgos
<code>Txn.close_remainder_to()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.vote_pk()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.selection_pk()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.vote_first()</code>	<code>TealType.uint64</code>	
<code>Txn.vote_last()</code>	<code>TealType.uint64</code>	
<code>Txn.vote_key_dilution()</code>	<code>TealType.uint64</code>	
<code>Txn.type()</code>	<code>TealType.bytes</code>	
<code>Txn.type_enum()</code>	<code>TealType.uint64</code>	see table below
<code>Txn.xfer_asset()</code>	<code>TealType.uint64</code>	ID of asset being transferred
<code>Txn.asset_amount()</code>	<code>TealType.uint64</code>	value in Asset's units
<code>Txn.asset_sender()</code>	<code>TealType.bytes</code>	32 byte address, causes clawback of all value if send
<code>Txn.asset_receiver()</code>	<code>TealType.bytes</code>	32 byte address

Continu

Table 1 – continued from previous page

Operator	Type	Notes
<code>Txn.asset_close_to()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.group_index()</code>	<code>TealType.uint64</code>	position of this transaction within a transaction group
<code>Txn.tx_id()</code>	<code>TealType.bytes</code>	the computed ID for this transaction, 32 bytes
<code>Txn.application_id()</code>	<code>TealType.uint64</code>	
<code>Txn.on_completion()</code>	<code>TealType.uint64</code>	
<code>Txn.approval_program()</code>	<code>TealType.bytes</code>	
<code>Txn.clear_state_program()</code>	<code>TealType.bytes</code>	
<code>Txn.rekey_to()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.config_asset()</code>	<code>TealType.uint64</code>	ID of asset being configured
<code>Txn.config_asset_total()</code>	<code>TealType.uint64</code>	
<code>Txn.config_asset_decimals()</code>	<code>TealType.uint64</code>	
<code>Txn.config_asset_default_frozen()</code>	<code>TealType.uint64</code>	
<code>Txn.config_asset_unit_name()</code>	<code>TealType.bytes</code>	
<code>Txn.config_asset_name()</code>	<code>TealType.bytes</code>	
<code>Txn.config_asset_url()</code>	<code>TealType.bytes</code>	
<code>Txn.config_asset_metadata_hash()</code>	<code>TealType.bytes</code>	
<code>Txn.config_asset_manager()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.config_asset_reserve()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.config_asset_freeze()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.config_asset_clawback()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.freeze_asset()</code>	<code>TealType.uint64</code>	
<code>Txn.freeze_asset_account()</code>	<code>TealType.bytes</code>	32 byte address
<code>Txn.freeze_asset_frozen()</code>	<code>TealType.uint64</code>	
<code>Txn.global_num_uints()</code>	<code>TealType.uint64</code>	Maximum global integers in app schema
<code>Txn.global_num_byte_slices()</code>	<code>TealType.uint64</code>	Maximum global byte strings in app schema
<code>Txn.local_num_uints()</code>	<code>TealType.uint64</code>	Maximum local integers in app schema
<code>Txn.local_num_byte_slices()</code>	<code>TealType.uint64</code>	Maximum local byte strings in app schema
<code>Txn.application_args</code>	<code>TealType.bytes[]</code>	Array of application arguments
<code>Txn.accounts</code>	<code>TealType.bytes[]</code>	Array of application accounts
<code>Txn.assets</code>	<code>TealType.uint64[]</code>	Array of application assets
<code>Txn.applications</code>	<code>TealType.uint64[]</code>	Array of applications

7.1.1 Transaction Type

The `Txn.type_enum()` values can be checked using the `TxnType` enum:

Value	Numerical Value	Type String	Description
<code>TxnType.Unknown</code>	0	unkown	unknown type, invalid
<code>TxnType.Payment</code>	1	pay	payment
<code>TxnType.KeyRegistration</code>	2	keyreg	key registration
<code>TxnType.AssetConfig</code>	3	acfg	asset config
<code>TxnType.AssetTransfer</code>	4	axfer	asset transfer
<code>TxnType.AssetFreeze</code>	5	afrz	asset freeze
<code>TxnType.ApplicationCall</code>	6	appl	application call

7.1.2 Transaction Array Fields

Some of the exposed transaction fields are arrays with the type `TealType.uint64[]` or `TealType.bytes[]`. These fields are `Txn.application_args`, `Txn.assets`, `Txn.accounts`, and `Txn.applications`.

The length of these array fields can be found using the `.length()` method, and individual items can be accessed using bracket notation. For example:

```
Txn.application_args.length() # get the number of application arguments in the_
    ↳ transaction
Txn.application_args[0] # get the first application argument
Txn.application_args[1] # get the second application argument
```

Special case: `Txn.accounts` and `Txn.applications`

The `Txn.accounts` and `Txn.applications` arrays are special cases. Normal arrays in PyTeal are 0-indexed, but these are 1-indexed with special values at index 0.

For the accounts array, `Txn.accounts[0]` is always equivalent to `Txn.sender()`.

For the applications array, `Txn.applications[0]` is always equivalent to `Txn.application_id()`.

IMPORTANT: Since these arrays are 1-indexed, their lengths are handled differently. For example, if `Txn.accounts.length()` or `Txn.applications.length()` is 2, then indexes 0, 1, and 2 will be present. In fact, the index 0 will always evaluate to the special values above, even when `length()` is 0.

7.2 Atomic Transfer Groups

Atomic Transfers are irreducible batch transactions that allow groups of transactions to be submitted at one time. If any of the transactions fail, then all the transactions will fail. PyTeal allows programs to access information about the transactions in an atomic transfer group using the `Gtxn` object. This object acts like a list of `TxnObject`, meaning all of the above transaction fields on `Txn` are available on the elements of `Gtxn`. For example:

```
Gtxn[0].sender() # get the sender of the first transaction in the atomic transfer_
    ↳ group
Gtxn[1].receiver() # get the receiver of the second transaction in the atomic_
    ↳ transfer group
```

`Gtxn` is zero-indexed and the maximum size of an atomic transfer group is 16. The size of the current transaction group is available as `Global.group_size()`. A standalone transaction will have a group size of 1.

To find the current transaction's index in the transfer group, use `Txn.group_index()`. If the current transaction is standalone, its group index will be 0.

7.3 Global Parameters

Information about the current state of the blockchain can be obtained using the following `Global` expressions:

Operator	Type	Notes
<code>Global.min_txn_fee()</code>	<code>TealType.uint64</code>	in microAlgos
<code>Global.min_balance()</code>	<code>TealType.uint64</code>	in mircoAlgos
<code>Global.max_txn_life()</code>	<code>TealType.uint64</code>	number of rounds
<code>Global.zero_address()</code>	<code>TealType.bytes</code>	32 byte address of all zero bytes
<code>Global.group_size()</code>	<code>TealType.uint64</code>	number of txns in this atomic transaction group, at least 1
<code>Global.logic_sig_version()</code>	<code>TealType.uint64</code>	the maximum supported TEAL version
<code>Global.round()</code>	<code>TealType.uint64</code>	the current round number
<code>Global.latest_timestamp()</code>	<code>TealType.uint64</code>	the latest confirmed block UNIX timestamp
<code>Global.current_application_id()</code>	<code>TealType.uint64</code>	the ID of the current application executing

Cryptographic Primitives

Algorand Smart Contracts support 4 cryptographic primitives, including 3 cryptographic hash functions and 1 digital signature verification. Each of these cryptographic primitives is associated with a cost, which is a number indicating its relative performance overhead comparing with simple teal operations such as addition and subtraction. All TEAL opcodes except crypto primitives have cost *1*. Below is how you express cryptographic primitives in PyTeal:

Operator	Cost	Description
Sha256(e)	35	<i>SHA-256</i> hash function, produces 32 bytes
Keccak256(e)	130	<i>Keccak-256</i> hash function, produces 32 bytes
Sha512_256(e)	45	<i>SHA512-256</i> hash function, produces 32 bytes
Ed25519Verify(d, s, p)	1900*	1 if s is the signature of d signed by p (PK), else 0

* Ed25519Verify is only available in signature mode.

These cryptographic primitives cover the most used ones in blockchains and cryptocurrencies. For example, Bitcoin uses *SHA-256* for creating Bitcoin addresses; Algorand uses *ed25519* signature scheme for authorization and uses *SHA512-256* hash function for creating contract account addresses from TEAL bytecode.

Scratch Space

Scratch space is a temporary place to store values for later use in your program. It is temporary because any changes to scratch space do not persist beyond the current transaction. Scratch space can be used in both Application and Signature mode.

Scratch space consists of 256 scratch slots, each capable of storing one integer or byte slice. When using the *ScratchVar* class to work with scratch space, a slot is automatically assigned to each variable.

9.1 Writing and Reading

To write to scratch space, first create a *ScratchVar* object and pass in the *TealType* of the values that you will store there. It is possible to create a *ScratchVar* that can store both integers and byte slices by passing no arguments to the *ScratchVar* constructor, but note that no type checking takes place in this situation.

To write or read values, use the corresponding *ScratchVar.store* or *ScratchVar.load* methods.

For example:

```
myvar = ScratchVar(TealType.uint64)
program = Seq([
    myvar.store(Int(5)),
    Assert(myvar.load() == Int(5))
])
```


PyTeal provides several control flow expressions to chain together multiple expressions and to conditionally evaluate expressions.

10.1 Exiting the Program: Return

The `Return` expression causes the program to exit immediately. It takes a single argument, which is the success value that the program should have. For example, `Return(Int(0))` causes the program to immediately fail, and `Return(Int(1))` causes the program to immediately succeed. Since the presence of a `Return` statement causes the program to exit, no operations after it will be executed.

10.2 Chaining Expressions: Seq

The `Seq` expression can be used to create a sequence of multiple expressions. It takes a single argument, which is a list of expressions to include in the sequence. For example:

```
Seq([
    App.globalPut(Bytes("creator"), Txn.sender()),
    Return(Int(1))
])
```

A `Seq` expression will take on the value of its last expression. Additionally, all expressions in a `Seq` expression, except the last one, must not return anything (e.g. evaluate to `TealType.none`). This restriction is in place because intermediate values must not add things to the TEAL stack. As a result, the following is an invalid sequence:

Listing 1: Invalid Seq expression

```
Seq([
    Txn.sender(),
    Return(Int(1))
])
```

If you must include an operation that returns a value in the earlier part of a sequence, you can wrap the value in a *Pop* expression to discard it. For example,

```
Seq([
    Pop(Txn.sender()),
    Return(Int(1))
])
```

10.3 Simple Branching: If

In an *If* expression,

```
If(test-expr, then-expr, else-expr)
```

the `test-expr` is always evaluated and needs to be typed `TealType.uint64`. If it results in a value greater than 0, then the `then-expr` is evaluated. Otherwise, `else-expr` is evaluated. Note that `then-expr` and `else-expr` must evaluate to the same type (e.g. both `TealType.uint64`).

You may also invoke an *If* expression without an `else-expr`:

```
If(test-expr, then-expr)
```

In this case, `then-expr` must be typed `TealType.none`.

10.4 Checking Conditions: Assert

The *Assert* expression can be used to ensure that conditions are met before continuing the program. The syntax for *Assert* is:

```
Assert(test-expr)
```

If `test-expr` is always evaluated and must be typed `TealType.uint64`. If `test-expr` results in a value greater than 0, the program continues. Otherwise, the program immediately exits and indicates that it encountered an error.

Example:

```
Assert(Txn.type_enum() == TxnType.Payment)
```

The above example will cause the program to immediately fail with an error if the transaction type is not a payment.

10.5 Chaining Tests: Cond

A *Cond* expression chains a series of tests to select a result expression. The syntax of *Cond* is:

```
Cond([test-expr-1, body-1],
     [test-expr-2, body-2],
     . . . )
```

Each `test-expr` is evaluated in order. If it produces 0, the paired `body` is ignored, and evaluation proceeds to the next `test-expr`. As soon as a `test-expr` produces a true value (> 0), its `body` is evaluated to produce the value

for this `Cond` expression. If none of `test-expr` s evaluates to a true value, the `Cond` expression will be evaluated to `err`, a TEAL opcode that causes the runtime panic.

In a `Cond` expression, each `test-expr` needs to be typed `TealType.uint64`. A `body` could be typed either `TealType.uint64` or `TealType.bytes`. However, all `body` s must have the same data type. Otherwise, a `TealTypeError` is triggered.

Example:

```
Cond([Global.group_size() == Int(5), bid],  
     [Global.group_size() == Int(4), redeem],  
     [Global.group_size() == Int(1), wrapup])
```

This PyTeal code branches on the size of the atomic transaction group.

State Access and Manipulation

PyTeal can be used to write [Stateful Algorand Smart Contracts](#) as well. Stateful contracts, also known as applications, can access and manipulate state on the Algorand blockchain.

State consists of key-value pairs, where keys are byte slices and values can be integers or byte slices. There are multiple types of state that an application can use.

11.1 State Operation Table

Context	Write	Read	Delete	Check If Exists
Current App Global	<code>App.globalPut</code>	<code>App.globalGet</code>	<code>App.globalDel</code>	<code>App.globalGetEx</code>
Current App Local	<code>App.localPut</code>	<code>App.localGet</code>	<code>App.localDel</code>	<code>App.localGetEx</code>
Other App Global		<code>App.globalGetEx</code>		<code>App.globalGetEx</code>
Other App Local		<code>App.localGetEx</code>		<code>App.localGetEx</code>

11.2 Global State

Global state consists of key-value pairs that are stored in the application's global context. It can be manipulated as follows:

11.2.1 Writing Global State

To write to global state, use the `App.globalPut` function. The first argument is the key to write to, and the second argument is the value to write. For example:

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

11.2.2 Reading Global State

To read from global state, use the `App.globalGet` function. The only argument it takes is the key to read from. For example:

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

If you try to read from a key that does not exist in your app's global state, the integer `0` is returned.

11.2.3 Deleting Global State

To delete a key from global state, use the `App.globalDel` function. The only argument it takes is the key to delete. For example:

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

If you try to delete a key that does not exist in your app's global state, nothing happens.

11.3 Local State

Local state consists of key-value pairs that are stored in a unique context for each account that has opted into your application. As a result, you will need to specify an account when manipulating local state. This is done by passing in an integer that corresponds to the index of the account in the `Txn.accounts` array.

In order to read or manipulate an account's local state, that account must be present in the application call transaction's `Txn.accounts` array.

Note: The `Txn.accounts` array does not behave like a normal array. It's actually a 1-indexed array with a special value at index `0`, the sender's account. See *Special case: Txn.accounts and Txn.applications* for more details.

11.3.1 Writing Local State

To write to the local state of an account, use the `App.localPut` function. The first argument is an integer corresponding to the account to write to, the second argument is the key to write to, and the third argument is the value to write. For example:

```
App.localPut(Int(0), Bytes("role"), Bytes("admin")) # write a byte slice to Txn.
↳accounts[0], the sender's account
App.localPut(Int(0), Bytes("balance"), Int(10)) # write a uint64 to Txn.accounts[0],
↳the sender's account
App.localPut(Int(1), Bytes("balance"), Int(10)) # write a uint64 to Txn.accounts[1]
```

Note: It is only possible to write to the local state of an account if that account has opted into your application. If the account has not opted in, the program will fail with an error. The function `App.optedIn` can be used to check if an account has opted into an app.

11.3.2 Reading Local State

To read from the local state of an account, use the `App.localGet` function. The first argument is an integer corresponding to the account to read from and the second argument is the key to read. For example:

```
App.localGet(Int(0), Bytes("role")) # read from Txn.accounts[0], the sender's account
App.localGet(Int(0), Bytes("balance")) # read from Txn.accounts[0], the the sender's
↪account
App.localGet(Int(1), Bytes("balance")) # read from Txn.accounts[1]
```

If you try to read from a key that does not exist in your app's global state, the integer 0 is returned.

11.3.3 Deleting Local State

To delete a key from local state of an account, use the `App.localDel` function. The first argument is an integer corresponding to the account and the second argument is the key to delete. For example:

```
App.localDel(Int(0), Bytes("role")) # delete "role" from Txn.accounts[0], the sender
↪'s account
App.localDel(Int(0), Bytes("balance")) # delete "balance" from Txn.accounts[0], the
↪the sender's account
App.localDel(Int(1), Bytes("balance")) # delete "balance" from Txn.accounts[1]
```

If you try to delete a key that does not exist in the account's local state, nothing happens.

11.4 External State

The above functions allow an app to read and write state in its own context. Additionally, 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`. This value 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.

Note: Even though the `MaybeValue` returned by `App.globalGetEx` and `App.localGetEx` cannot be used directly, it **must** be included in the application before `hasValue()` and `value()` are called on it. You will probably want to use `Seq` to do this.

Since these functions are the only way to check whether a key exists, it can be useful to use them in the current application's context too.

11.4.1 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 the index of an application in the `Txn.applications` array.

This means that in order to read or manipulate an external application's local state, that application must be present in the application call transaction's `Txn.applications` array.

Note: The `Txn.applications` array does not behave like a normal array. It's actually a 1-indexed array with a special value at index 0, the current application's ID. See *Special case: Txn.accounts and Txn.applications* for more details.

Now that you have an integer that represents an application to read from, pass this as the first argument to `App.globalGetEx`, and pass the key to read as the second argument. For example:

```
# get "status" from the global context of Txn.applications[0] (the current app)
# if "status" has not been set, returns "none"
myStatus = App.globalGetEx(Int(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(Int(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(Int(1), Bytes("total supply"))
program = Seq([
    otherSupply,
    otherSupply.value()
])
```

11.4.2 External Local

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

The first argument is an integer corresponding to 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.

Note: The second argument is the actual ID of the application to read from, not an index into `Txn.applications`. This means that you can read from any application that the account has opted into, not just applications included in `Txn.applications`. The ID 0 is still a special value that refers to the ID of the current application, but you could also use `Global.current_application_id()` or `Txn.application_id()` to refer to the current application.

For example:

```
# 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(Int(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(Int(1), Int(0), Bytes("role"))
```

(continues on next page)

(continued from previous page)

```

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(Int(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(Int(1), Int(31), Bytes("role"))
program = Seq([
    otherAppOtherAccountRole,
    If(otherAppOtherAccountRole.hasValue(), otherAppOtherAccountRole.value(), Bytes(
↪ "none"))
])

```


CHAPTER 12

Asset Information

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

12.1 Algo Balances

The *Balance* expression can be used to look up an account's balance in microAlgos (1 Algo = 1,000,000 microAlgos). For example,

```
senderBalance = Balance(Int(0)) # get the balance of Txn.accounts[0] (the sender)
account1Balance = Balance(Int(1)) # get the balance of Txn.accounts[1]
```

The *MinBalance* expression can be used to find an account's **minimum balance**. This amount is also in microAlgos. For example,

```
senderMinBalance = MinBalance(Int(0)) # get the minimum balance of Txn.accounts[0]
↳ (the sender)
account1MinBalance = MinBalance(Int(1)) # get the minimum balance of Txn.accounts[1]
```

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

```
senderSpendableBalance = Balance(Int(0)) - MinBalance(Int(0)) # calculate how many
↳ Algos Txn.accounts[0] (the sender) can spend
account1SpendableBalance = Balance(Int(1)) - MinBalance(Int(1)) # calculate how many
↳ Algos Txn.accounts[1] can spend
```

12.2 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 expressions return a *MaybeValue*. 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.

12.2.1 Balances

The *AssetHolding.balance* expression can be used to look up how many units of an asset an account holds. For example,

```
# get the balance of Txn.accounts[0] (the sender) for asset 31566704
# if the account is not opted into that asset, returns 0
senderAssetBalance = AssetHolding.balance(Int(0), Int(31566704))
program = Seq([
    senderAssetBalance,
    senderAssetBalance.value()
])

# get the balance of Txn.accounts[1] for asset 27165954
# if the account is not opted into that asset, exit with an error
account1AssetBalance = AssetHolding.balance(Int(1), Int(27165954))
program = Seq([
    account1AssetBalance,
    Assert(account1AssetBalance.hasValue()),
    account1AssetBalance.value()
])
```

12.2.2 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. For example,

```
# get the frozen status of Txn.accounts[0] (the sender) for asset 31566704
# if the account is not opted into that asset, returns 0
senderAssetFrozen = AssetHolding.frozen(Int(0), Int(31566704))
program = Seq([
    senderAssetFrozen,
    senderAssetFrozen.value()
])

# get the frozen status of Txn.accounts[1] for asset 27165954
# if the account is not opted into that asset, exit with an error
account1AssetFrozen = AssetHolding.frozen(Int(1), Int(27165954))
program = Seq([
    account1AssetFrozen,
    Assert(account1AssetFrozen.hasValue()),
    account1AssetFrozen.value()
])
```

12.3 Asset Parameters

Every ASA has parameters that contain information about the asset and how it behaves. These parameters can be read by TEAL applications for any asset in the `Txn.assets` array.

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).

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`).

The different parameters that can be accessed are summarized by the table below. More information about each parameter can be found on the [Algorand developer website](#).

Expression	Type	Description
<code>AssetParam.total()</code>	<code>TealType.uint64</code>	The total number of units of the asset.
<code>AssetParam.decimals()</code>	<code>TealType.uint64</code>	The number of decimals the asset should be formatted with.
<code>AssetParam.defaultFrozen()</code>	<code>TealType.uint64</code>	Whether the asset is frozen by default.
<code>AssetParam.unitName()</code>	<code>TealType.bytes</code>	The name of the asset's units.
<code>AssetParam.name()</code>	<code>TealType.bytes</code>	The name of the asset.
<code>AssetParam.url()</code>	<code>TealType.bytes</code>	A URL associated with the asset.
<code>AssetParam.metadataHash()</code>	<code>TealType.bytes</code>	A 32-byte hash associated with the asset.
<code>AssetParam.manager()</code>	<code>TealType.bytes</code>	The address of the asset's manager account.
<code>AssetParam.reserve()</code>	<code>TealType.bytes</code>	The address of the asset's reserve account.
<code>AssetParam.freeze()</code>	<code>TealType.bytes</code>	The address of the asset's freeze account.
<code>AssetParam.clawback()</code>	<code>TealType.bytes</code>	The address of the asset's clawback account.

Here's an example that uses an asset parameter:

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


CHAPTER 13

TEAL Versions

Each version of PyTeal compiles contracts for a specific version of TEAL. Newer versions of TEAL introduce new opcodes and transaction fields, so PyTeal must be updated to support these new features. Below is a table which shows the relationship between TEAL and PyTeal versions.

TEAL Version	PyTeal Version
1	$\leq 0.5.4$
2	$\geq 0.6.0$
3	$\geq 0.7.0$

In order to support TEAL v2, PyTeal v0.6.0 breaks backward compatibility with v0.5.4. PyTeal programs written for PyTeal version 0.5.4 and below will not compile properly and most likely will display an error of the form `AttributeError: * object has no attribute 'teal'`.

WARNING: before updating PyTeal to a version which generates TEAL v2 contracts and fixing the programs to use the global function `compileTeal` rather than the class method `.teal()`, make sure your program abides by the TEAL safety guidelines <https://developer.algorand.org/docs/reference/teal/guidelines/>. Changing a v1 TEAL program to a v2 TEAL program without any code changes is insecure because v2 TEAL programs allow rekeying. Specifically, you must add a check that the `RekeyTo` property of any transaction is set to the zero address when updating an older PyTeal program from v0.5.4 and below.

CHAPTER 14

PyTeal Package

`pyteal.Txn = <pyteal.TxnObject object>`
The current transaction being evaluated.

`pyteal.Gtxn = <pyteal.TxnGroup object>`
The current group of transactions being evaluated.

class `pyteal.Expr`
Bases: `abc.ABC`

Abstract base class for PyTeal expressions.

And (*other: pyteal.Expr*) $\rightarrow$ `pyteal.Expr`
Take the logical And of this expression and another one.

This expression must evaluate to uint64.

This is the same as using `And()` with two arguments.

Or (*other: pyteal.Expr*) $\rightarrow$ `pyteal.Expr`
Take the logical Or of this expression and another one.

This expression must evaluate to uint64.

This is the same as using `Or()` with two arguments.

getDefinitionTrace () $\rightarrow$ `List[str]`

type_of () $\rightarrow$ `pyteal.TealType`
Get the return type of this expression.

class `pyteal.LeafExpr`
Bases: `pyteal.Expr`

Leaf expression base class.

class `pyteal.Addr` (*address: str*)
Bases: `pyteal.LeafExpr`

An expression that represents an Algorand address.

`__init__ (address: str) → None`

Create a new Addr expression.

Parameters **address** – A string containing a valid base32 Algorand address

`type_of ()`

Get the return type of this expression.

class `pyteal.Bytes (*args)`

Bases: `pyteal.LeafExpr`

An expression that represents a byte string.

`__init__ (*args) → None`

Create a new byte string.

Depending on the encoding, there are different arguments to pass:

For UTF-8 strings: Pass the string as the only argument. For example, `Bytes ("content")`.

For base16, base32, or base64 strings: Pass the base as the first argument and the string as the second argument. For example, `Bytes ("base16", "636F6E746556E74")`, `Bytes ("base32", "ORFDPQ6ARJK")`, `Bytes ("base64", "Y29udGVudA==")`.

Special case for base16: The prefix “0x” may be present in a base16 byte string. For example, `Bytes ("base16", "0x636F6E746556E74")`.

`type_of ()`

Get the return type of this expression.

class `pyteal.Err`

Bases: `pyteal.LeafExpr`

Expression that causes the program to immediately fail when executed.

`type_of ()`

Get the return type of this expression.

class `pyteal.Int (value: int)`

Bases: `pyteal.LeafExpr`

An expression that represents a uint64.

`__init__ (value: int) → None`

Create a new uint64.

Parameters **value** – The integer value this uint64 will represent. Must be a positive value less than 2^{64} .

`type_of ()`

Get the return type of this expression.

class `pyteal.EnumInt (name: str)`

Bases: `pyteal.LeafExpr`

An expression that represents uint64 enum values.

`__init__ (name: str) → None`

Create an expression to reference a uint64 enum value.

Parameters **name** – The name of the enum value.

`type_of ()`

Get the return type of this expression.

```

class pyteal.Arg(index: int)
    Bases: pyteal.LeafExpr

    An expression to get an argument when running in signature verification mode.

    __init__(index: int) → None
        Get an argument for this program.

        Should only be used in signature verification mode. For application mode arguments, see TxnObject.application_args.

        Parameters index – The integer index of the argument to get. Must be between 0 and 255 inclusive.

    type_of()
        Get the return type of this expression.

class pyteal.TxnType
    Bases: object

    Enum of all possible transaction types.

    ApplicationCall = <pyteal.EnumInt object>
    AssetConfig = <pyteal.EnumInt object>
    AssetFreeze = <pyteal.EnumInt object>
    AssetTransfer = <pyteal.EnumInt object>
    KeyRegistration = <pyteal.EnumInt object>
    Payment = <pyteal.EnumInt object>
    Unknown = <pyteal.EnumInt object>

class pyteal.TxnField(id: int, name: str, type: pyteal.TealType, min_version: int)
    Bases: enum.Enum

    An enumeration.

    accounts = (28, 'Accounts', <TealType.bytes: 1>, 2)
    amount = (8, 'Amount', <TealType.uint64: 0>, 2)
    application_args = (26, 'ApplicationArgs', <TealType.bytes: 1>, 2)
    application_id = (24, 'ApplicationID', <TealType.uint64: 0>, 2)
    applications = (50, 'Applications', <TealType.uint64: 0>, 3)
    approval_program = (30, 'ApprovalProgram', <TealType.bytes: 1>, 2)
    asset_amount = (18, 'AssetAmount', <TealType.uint64: 0>, 2)
    asset_close_to = (21, 'AssetCloseTo', <TealType.bytes: 1>, 2)
    asset_receiver = (20, 'AssetReceiver', <TealType.bytes: 1>, 2)
    asset_sender = (19, 'AssetSender', <TealType.bytes: 1>, 2)
    assets = (48, 'Assets', <TealType.uint64: 0>, 3)
    clear_state_program = (31, 'ClearStateProgram', <TealType.bytes: 1>, 2)
    close_remainder_to = (9, 'CloseRemainderTo', <TealType.bytes: 1>, 2)
    config_asset = (33, 'ConfigAsset', <TealType.uint64: 0>, 2)

```

```
config_asset_clawback = (44, 'ConfigAssetClawback', <TealType.bytes: 1>, 2)
config_asset_decimals = (35, 'ConfigAssetDecimals', <TealType.uint64: 0>, 2)
config_asset_default_frozen = (36, 'ConfigAssetDefaultFrozen', <TealType.uint64: 0>, 2)
config_asset_freeze = (43, 'ConfigAssetFreeze', <TealType.bytes: 1>, 2)
config_asset_manager = (41, 'ConfigAssetManager', <TealType.bytes: 1>, 2)
config_asset_metadata_hash = (40, 'ConfigAssetMetadataHash', <TealType.bytes: 1>, 2)
config_asset_name = (38, 'ConfigAssetName', <TealType.bytes: 1>, 2)
config_asset_reserve = (42, 'ConfigAssetReserve', <TealType.bytes: 1>, 2)
config_asset_total = (34, 'ConfigAssetTotal', <TealType.uint64: 0>, 2)
config_asset_unit_name = (37, 'ConfigAssetUnitName', <TealType.bytes: 1>, 2)
config_asset_url = (39, 'ConfigAssetURL', <TealType.bytes: 1>, 2)
fee = (1, 'Fee', <TealType.uint64: 0>, 2)
first_valid = (2, 'FirstValid', <TealType.uint64: 0>, 2)
first_valid_time = (3, 'FirstValidTime', <TealType.uint64: 0>, 2)
freeze_asset = (45, 'FreezeAsset', <TealType.uint64: 0>, 2)
freeze_asset_account = (46, 'FreezeAssetAccount', <TealType.bytes: 1>, 2)
freeze_asset_frozen = (47, 'FreezeAssetFrozen', <TealType.uint64: 0>, 2)
global_num_byte_slices = (53, 'GlobalNumByteSlice', <TealType.uint64: 0>, 3)
global_num_uints = (52, 'GlobalNumUint', <TealType.uint64: 0>, 3)
group_index = (22, 'GroupIndex', <TealType.uint64: 0>, 2)
last_valid = (4, 'LastValid', <TealType.uint64: 0>, 2)
lease = (6, 'Lease', <TealType.bytes: 1>, 2)
local_num_byte_slices = (55, 'LocalNumByteSlice', <TealType.uint64: 0>, 3)
local_num_uints = (54, 'LocalNumUint', <TealType.uint64: 0>, 3)
note = (5, 'Note', <TealType.bytes: 1>, 2)
num_accounts = (2, 'NumAccounts', <TealType.uint64: 0>, 2)
num_app_args = (27, 'NumAppArgs', <TealType.uint64: 0>, 2)
num_applications = (51, 'NumApplications', <TealType.uint64: 0>, 3)
num_assets = (49, 'NumAssets', <TealType.uint64: 0>, 3)
on_completion = (25, 'OnCompletion', <TealType.uint64: 0>, 2)
receiver = (7, 'Receiver', <TealType.bytes: 1>, 2)
rekey_to = (32, 'RekeyTo', <TealType.bytes: 1>, 2)
selection_pk = (11, 'SelectionPK', <TealType.bytes: 1>, 2)
sender = (0, 'Sender', <TealType.bytes: 1>, 2)
tx_id = (23, 'TxID', <TealType.bytes: 1>, 2)
type = (15, 'Type', <TealType.bytes: 1>, 2)
```

```

type_enum = (16, 'TypeEnum', <TealType.uint64: 0>, 2)
type_of() → pyteal.TealType
vote_first = (12, 'VoteFirst', <TealType.uint64: 0>, 2)
vote_key_dilution = (14, 'VoteKeyDilution', <TealType.uint64: 0>, 2)
vote_last = (13, 'VoteLast', <TealType.uint64: 0>, 2)
vote_pk = (10, 'VotePK', <TealType.bytes: 1>, 2)
xfer_asset = (17, 'XferAsset', <TealType.uint64: 0>, 2)

```

class `pyteal.TxnExpr` (*field: pyteal.TxnField*)
 Bases: `pyteal.LeafExpr`

An expression that accesses a transaction field from the current transaction.

type_of()
 Get the return type of this expression.

class `pyteal.TxnaExpr` (*field: pyteal.TxnField, index: int*)
 Bases: `pyteal.LeafExpr`

An expression that accesses a transaction array field from the current transaction.

type_of()
 Get the return type of this expression.

class `pyteal.TxnArray` (*txnObject: pyteal.TxnObject, accessField: pyteal.TxnField, lengthField: pyteal.TxnField*)
 Bases: `pyteal.Array`

Represents a transaction array field.

__getitem__ (*index: int*) → `pyteal.TxnaExpr`
 Get the value at a given index in this array.

length () → `pyteal.TxnExpr`
 Get the length of the array.

class `pyteal.TxnObject` (*txnType: Callable[[pyteal.TxnField], pyteal.TxnExpr], txnaType: Callable[[pyteal.TxnField, int], pyteal.TxnaExpr]*)
 Bases: `object`

Represents a transaction and its fields.

accounts
 The accounts array in an ApplicationCall transaction.
 Type `TxnArray`

amount () → `pyteal.TxnExpr`
 Get the amount of the transaction in micro Algos.
 Only set when `type_enum()` is `TxnType.Payment`.
 For more information, see <https://developer.algorand.org/docs/reference/transactions/#amount>

application_args
 Application call arguments array.
 Type `TxnArray`

application_id() → `pyteal.TxnExpr`

Get the application ID from the `ApplicationCall` portion of the current transaction.

Only set when `type_enum()` is `TxnType.ApplicationCall`.

applications

The applications array in an `ApplicationCall` transaction.

Type `TxnArray`

Requires TEAL version 3 or higher.

approval_program() → `pyteal.TxnExpr`

Get the approval program.

Only set when `type_enum()` is `TxnType.ApplicationCall`.

asset_amount() → `pyteal.TxnExpr`

Get the amount of the asset being transferred, measured in the asset's units.

Only set when `type_enum()` is `TxnType.AssetTransfer`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetamount>

asset_close_to() → `pyteal.TxnExpr`

Get the closeout address of the asset transfer.

Only set when `type_enum()` is `TxnType.AssetTransfer`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetcloseto>

asset_receiver() → `pyteal.TxnExpr`

Get the recipient of the asset transfer.

Only set when `type_enum()` is `TxnType.AssetTransfer`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetreceiver>

asset_sender() → `pyteal.TxnExpr`

Get the 32 byte address of the subject of clawback.

Only set when `type_enum()` is `TxnType.AssetTransfer`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetsender>

assets

The foreign asset array in an `ApplicationCall` transaction.

Type `TxnArray`

Requires TEAL version 3 or higher.

clear_state_program() → `pyteal.TxnExpr`

Get the clear state program.

Only set when `type_enum()` is `TxnType.ApplicationCall`.

close_remainder_to() → `pyteal.TxnExpr`

Get the 32 byte address of the `CloseRemainderTo` field.

Only set when `type_enum()` is `TxnType.Payment`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#closeremainderto>

config_asset() → `pyteal.TxnExpr`

Get the asset ID in asset config transaction.

Only set when `type_enum()` is `TxnType.AssetConfig`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#configasset>

config_asset_clawback () → pyteal.TxnExpr

Get the 32 byte asset clawback address.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#clawbackaddr>

config_asset_decimals () → pyteal.TxnExpr

Get the number of digits to display after the decimal place when displaying the asset.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#decimals>

config_asset_default_frozen () → pyteal.TxnExpr

Check if the asset's slots are frozen by default or not.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#defaultfrozen>

config_asset_freeze () → pyteal.TxnExpr

Get the 32 byte asset freeze address.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#freezeaddr>

config_asset_manager () → pyteal.TxnExpr

Get the 32 byte asset manager address.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#manageraddr>

config_asset_metadata_hash () → pyteal.TxnExpr

Get the 32 byte commitment to some unspecified asset metadata.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#metadathash>

config_asset_name () → pyteal.TxnExpr

Get the asset name.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetname>

config_asset_reserve () → pyteal.TxnExpr

Get the 32 byte asset reserve address.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#reserveaddr>

config_asset_total () → pyteal.TxnExpr

Get the total number of units of this asset created.

Only set when *type_enum*() is *TxnType.AssetConfig*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#total>

config_asset_unit_name () → pyteal.TxnExpr

Get the unit name of the asset.

Only set when `type_enum()` is `TxnType.AssetConfig`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#unitname>

config_asset_url() → `pyteal.TxnExpr`

Get the asset URL.

Only set when `type_enum()` is `TxnType.AssetConfig`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#url>

fee() → `pyteal.TxnExpr`

Get the transaction fee in micro Algos.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#fee>

first_valid() → `pyteal.TxnExpr`

Get the first valid round number.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#firstvalid>

freeze_asset() → `pyteal.TxnExpr`

Get the asset ID being frozen or un-frozen.

Only set when `type_enum()` is `TxnType.AssetFreeze`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#freezeasset>

freeze_asset_account() → `pyteal.TxnExpr`

Get the 32 byte address of the account whose asset slot is being frozen or un-frozen.

Only set when `type_enum()` is `TxnType.AssetFreeze`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#freezeaccount>

freeze_asset_frozen() → `pyteal.TxnExpr`

Get the new frozen value for the asset.

Only set when `type_enum()` is `TxnType.AssetFreeze`.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#assetfrozen>

global_num_byte_slices() → `pyteal.TxnExpr`

Get the schema count of global state byte slices in an application creation call.

Only set when `type_enum()` is `TxnType.ApplicationCall` and this is an app creation call.

Requires TEAL version 3 or higher.

global_num_uints() → `pyteal.TxnExpr`

Get the schema count of global state integers in an application creation call.

Only set when `type_enum()` is `TxnType.ApplicationCall` and this is an app creation call.

Requires TEAL version 3 or higher.

group_index() → `pyteal.TxnExpr`

Get the position of the transaction within the atomic transaction group.

A stand-alone transaction is implicitly element 0 in a group of 1.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#group>

last_valid() → `pyteal.TxnExpr`

Get the last valid round number.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#lastvalid>

lease () → pyteal.TxnExpr
Get the transaction lease.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#lease>

local_num_byte_slices () → pyteal.TxnExpr
Get the schema count of local state byte slices in an application creation call.
Only set when *type_enum()* is *TxnType.ApplicationCall* and this is an app creation call.
Requires TEAL version 3 or higher.

local_num_uints () → pyteal.TxnExpr
Get the schema count of local state integers in an application creation call.
Only set when *type_enum()* is *TxnType.ApplicationCall* and this is an app creation call.
Requires TEAL version 3 or higher.

note () → pyteal.TxnExpr
Get the transaction note.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#note>

on_completion () → pyteal.TxnExpr
Get the on completion action from the ApplicationCall portion of the transaction.
Only set when *type_enum()* is *TxnType.ApplicationCall*.

receiver () → pyteal.TxnExpr
Get the 32 byte address of the receiver.
Only set when *type_enum()* is *TxnType.Payment*.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#receiver>

rekey_to () → pyteal.TxnExpr
Get the sender's new 32 byte AuthAddr.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#rekeyto>

selection_pk () → pyteal.TxnExpr
Get the VRF public key.
Only set when *type_enum()* is *TxnType.KeyRegistration*.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#selectionpk>

sender () → pyteal.TxnExpr
Get the 32 byte address of the sender.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#sender>

tx_id () → pyteal.TxnExpr
Get the 32 byte computed ID for the transaction.

type () → pyteal.TxnExpr
Get the type of this transaction as a byte string.
In most cases it is preferable to use *type_enum()* instead.
For more information, see <https://developer.algorand.org/docs/reference/transactions/#type>

type_enum () → pyteal.TxnExpr
Get the type of this transaction.
See *TxnType* for possible values.

vote_first () → pyteal.TxnExpr

Get the first round that the participation key is valid.

Only set when *type_enum*() is *TxnType.KeyRegistration*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#votefirst>

vote_key_dilution () → pyteal.TxnExpr

Get the dilution for the 2-level participation key.

Only set when *type_enum*() is *TxnType.KeyRegistration*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#votekeydilution>

vote_last () → pyteal.TxnExpr

Get the last round that the participation key is valid.

Only set when *type_enum*() is *TxnType.KeyRegistration*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#votelast>

vote_pk () → pyteal.TxnExpr

Get the root participation public key.

Only set when *type_enum*() is *TxnType.KeyRegistration*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#votePk>

xfer_asset () → pyteal.TxnExpr

Get the ID of the asset being transferred.

Only set when *type_enum*() is *TxnType.AssetTransfer*.

For more information, see <https://developer.algorand.org/docs/reference/transactions/#xferasset>

class pyteal.GtxnExpr (txnIndex: Union[int, pyteal.Expr], field: pyteal.TxnField)

Bases: *pyteal.TxnExpr*

An expression that accesses a transaction field from a transaction in the current group.

class pyteal.GtxnaExpr (txnIndex: Union[int, pyteal.Expr], field: pyteal.TxnField, index: int)

Bases: *pyteal.TxnaExpr*

An expression that accesses a transaction array field from a transaction in the current group.

class pyteal.TxnGroup

Bases: object

Represents a group of transactions.

__getitem__ (txnIndex: Union[int, pyteal.Expr]) → pyteal.TxnObject

class pyteal.Global (field: pyteal.GlobalField)

Bases: *pyteal.LeafExpr*

An expression that accesses a global property.

classmethod creator_address () → pyteal.Global

Address of the creator of the current application.

Fails if no such application is executing. Requires TEAL version 3 or higher.

classmethod current_application_id () → pyteal.Global

Get the ID of the current application executing.

Fails if no application is executing.

```

classmethod group_size() → pyteal.Global
    Get the number of transactions in this atomic transaction group.

    This will be at least 1.

classmethod latest_timestamp() → pyteal.Global
    Get the latest confirmed block UNIX timestamp.

    Fails if negative.

classmethod logic_sig_version() → pyteal.Global
    Get the maximum supported TEAL version.

classmethod max_txn_life() → pyteal.Global
    Get the maximum number of rounds a transaction can have.

classmethod min_balance() → pyteal.Global
    Get the minumum balance in micro Algos.

classmethod min_txn_fee() → pyteal.Global
    Get the minumum transaction fee in micro Algos.

classmethod round() → pyteal.Global
    Get the current round number.

type_of()
    Get the return type of this expression.

classmethod zero_address() → pyteal.Global
    Get the 32 byte zero address.

class pyteal.GlobalField(id: int, name: str, type: pyteal.TealType, min_version: int)
    Bases: enum.Enum

    An enumeration.

    creator_address = (9, 'CreatorAddress', <TealType.bytes: 1>, 3)
    current_app_id = (8, 'CurrentApplicationID', <TealType.uint64: 0>, 2)
    group_size = (4, 'GroupSize', <TealType.uint64: 0>, 2)
    latest_timestamp = (7, 'LatestTimestamp', <TealType.uint64: 0>, 2)
    logic_sig_version = (5, 'LogicSigVersion', <TealType.uint64: 0>, 2)
    max_txn_life = (2, 'MaxTxnLife', <TealType.uint64: 0>, 2)
    min_balance = (1, 'MinBalance', <TealType.uint64: 0>, 2)
    min_txn_fee = (0, 'MinTxnFee', <TealType.uint64: 0>, 2)
    round = (6, 'Round', <TealType.uint64: 0>, 2)
    type_of() → pyteal.TealType
    zero_address = (3, 'ZeroAddress', <TealType.bytes: 1>, 2)

class pyteal.App(field: pyteal.AppField, args)
    Bases: pyteal.LeafExpr

    An expression related to applications.

    classmethod globalDel(key: pyteal.Expr) → pyteal.App
    Delete a key from the global state of the current application.

        Parameters key – The key to delete from the global application state. Must evaluate to bytes.

```

classmethod **globalGet** (*key: pyteal.Expr*) → *pyteal.App*

Read from the global state of the current application.

Parameters **key** – The key to read from the global application state. Must evaluate to bytes.

classmethod **globalGetEx** (*app: pyteal.Expr, key: pyteal.Expr*) → *pyteal.MaybeValue*

Read from the global state of an application.

Parameters

- **app** – An index into *Txn.ForeignApps* that corresponds to the application to read from. Must evaluate to *uint64*.
- **key** – The key to read from the global application state. Must evaluate to bytes.

classmethod **globalPut** (*key: pyteal.Expr, value: pyteal.Expr*) → *pyteal.App*

Write to the global state of the current application.

Parameters

- **key** – The key to write in the global application state. Must evaluate to bytes.
- **value** – The value to write in the global application state. Can evaluate to any type.

classmethod **id** () → *pyteal.Global*

Get the ID of the current running application.

This is the same as *Global.current_application_id()*.

classmethod **localDel** (*account: pyteal.Expr, key: pyteal.Expr*) → *pyteal.App*

Delete a key from an account's local state for the current application.

Parameters

- **account** – An index into *Txn.Accounts* that corresponds to the account from which the key should be deleted. Must evaluate to *uint64*.
- **key** – The key to delete from the account's local state. Must evaluate to bytes.

classmethod **localGet** (*account: pyteal.Expr, key: pyteal.Expr*) → *pyteal.App*

Read from an account's local state for the current application.

Parameters

- **account** – An index into *Txn.Accounts* that corresponds to the account to read from. Must evaluate to *uint64*.
- **key** – The key to read from the account's local state. Must evaluate to bytes.

classmethod **localGetEx** (*account: pyteal.Expr, app: pyteal.Expr, key: pyteal.Expr*) → *pyteal.MaybeValue*

Read from an account's local state for an application.

Parameters

- **account** – An index into *Txn.Accounts* that corresponds to the account to read from. Must evaluate to *uint64*.
- **app** – The ID of the application being checked. Must evaluate to *uint64*.
- **key** – The key to read from the account's local state. Must evaluate to bytes.

classmethod **localPut** (*account: pyteal.Expr, key: pyteal.Expr, value: pyteal.Expr*) → *pyteal.App*

Write to an account's local state for the current application.

Parameters

- **account** – An index into Txn.Accounts that corresponds to the account to write to. Must evaluate to uint64.
- **key** – The key to write in the account's local state. Must evaluate to bytes.
- **value** – The value to write in the account's local state. Can evaluate to any type.

classmethod **optedIn** (*account: pyteal.Expr, app: pyteal.Expr*) → pyteal.App

Check if an account has opted in for an application.

Parameters

- **account** – An index into Txn.Accounts that corresponds to the account to check. Must evaluate to uint64.
- **app** – The ID of the application being checked. Must evaluate to uint64.

type_of ()

Get the return type of this expression.

class **pyteal.AppField** (*op: pyteal.Op, type: pyteal.TealType*)

Bases: `enum.Enum`

Enum of app fields used to create [App](#) objects.

get_op () → pyteal.Op

globalDel = (<Op.app_global_del: OpType(value='app_global_del', mode=<Mode.Application:

globalGet = (<Op.app_global_get: OpType(value='app_global_get', mode=<Mode.Application:

globalGetEx = (<Op.app_global_get_ex: OpType(value='app_global_get_ex', mode=<Mode.Ap

globalPut = (<Op.app_global_put: OpType(value='app_global_put', mode=<Mode.Application:

localDel = (<Op.app_local_del: OpType(value='app_local_del', mode=<Mode.Application:

localGet = (<Op.app_local_get: OpType(value='app_local_get', mode=<Mode.Application:

localGetEx = (<Op.app_local_get_ex: OpType(value='app_local_get_ex', mode=<Mode.Appli

localPut = (<Op.app_local_put: OpType(value='app_local_put', mode=<Mode.Application:

optedIn = (<Op.app_opted_in: OpType(value='app_opted_in', mode=<Mode.Application: 2>

type_of () → pyteal.TealType

class **pyteal.OnComplete**

Bases: `object`

An enum of values that [TxnObject.on_completion\(\)](#) may return.

ClearState = <pyteal.EnumInt object>

CloseOut = <pyteal.EnumInt object>

DeleteApplication = <pyteal.EnumInt object>

NoOp = <pyteal.EnumInt object>

OptIn = <pyteal.EnumInt object>

UpdateApplication = <pyteal.EnumInt object>

class **pyteal.AssetHolding**

Bases: `object`

classmethod **balance** (*account: pyteal.Expr, asset: pyteal.Expr*) → pyteal.MaybeValue

Get the amount of an asset held by an account.

Parameters

- **account** – An index into Txn.Accounts that corresponds to the account to check. Must evaluate to uint64.
- **asset** – The ID of the asset to get. Must evaluate to uint64.

classmethod frozen (*account: pyteal.Expr, asset: pyteal.Expr*) → pyteal.MaybeValue

Check if an asset is frozen for an account.

A value of 1 indicates frozen and 0 indicates not frozen.

Parameters

- **account** – An index into Txn.Accounts that corresponds to the account to check. Must evaluate to uint64.
- **asset** – The ID of the asset to check. Must evaluate to uint64.

class pyteal.AssetParam

Bases: object

classmethod clawback (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the clawback address for an asset.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod decimals (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the number of decimals for an asset.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod defaultFrozen (*asset: pyteal.Expr*) → pyteal.MaybeValue

Check if an asset is frozen by default.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod freeze (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the freeze address for an asset.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod manager (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the manager address for an asset.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod metadataHash (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the arbitrary commitment for an asset.

If set, this will be 32 bytes long.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod name (*asset: pyteal.Expr*) → pyteal.MaybeValue

Get the name of an asset.

Parameters asset – An index into Txn.ForeignAssets that corresponds to the asset to check. Must evaluate to uint64.

classmethod **reserve** (*asset: pyteal.Expr*) → *pyteal.MaybeValue*

Get the reserve address for an asset.

Parameters **asset** – An index into Txn.ForeignAssets that corresponds to the asset to check.
Must evaluate to uint64.

classmethod **total** (*asset: pyteal.Expr*) → *pyteal.MaybeValue*

Get the total number of units of an asset.

Parameters **asset** – An index into Txn.ForeignAssets that corresponds to the asset to check.
Must evaluate to uint64.

classmethod **unitName** (*asset: pyteal.Expr*) → *pyteal.MaybeValue*

Get the unit name of an asset.

Parameters **asset** – An index into Txn.ForeignAssets that corresponds to the asset to check.
Must evaluate to uint64.

classmethod **url** (*asset: pyteal.Expr*) → *pyteal.MaybeValue*

Get the URL of an asset.

Parameters **asset** – An index into Txn.ForeignAssets that corresponds to the asset to check.
Must evaluate to uint64.

class **pyteal.Array**

Bases: *abc.ABC*

Represents a variable length array of objects.

__getitem__ (*index: int*)

Get the value at a given index in this array.

length () → *pyteal.Expr*

Get the length of the array.

class **pyteal.Tmpl** (*op: pyteal.Op, type: pyteal.TealType, name: str*)

Bases: *pyteal.LeafExpr*

Template expression for creating placeholder values.

classmethod **Addr** (*placeholder: str*)

Create a new Addr template.

Parameters **placeholder** – The name to use for this template variable. Must start with *TMPL_* and only consist of uppercase alphanumeric characters and underscores.

classmethod **Bytes** (*placeholder: str*)

Create a new Bytes template.

Parameters **placeholder** – The name to use for this template variable. Must start with *TMPL_* and only consist of uppercase alphanumeric characters and underscores.

classmethod **Int** (*placeholder: str*)

Create a new Int template.

Parameters **placeholder** – The name to use for this template variable. Must start with *TMPL_* and only consist of uppercase alphanumeric characters and underscores.

type_of ()

Get the return type of this expression.

class **pyteal.Nonce** (*base: str, nonce: str, child: pyteal.Expr*)

Bases: *pyteal.Expr*

A meta expression only used to change the hash of a TEAL program.

`__init__` (*base: str, nonce: str, child: pyteal.Expr*) → None
Create a new Nonce.

The Nonce expression behaves exactly like the child expression passed into it, except it uses the provided nonce string to alter its structure in a way that does not affect execution.

Parameters

- **base** – The base of the nonce. Must be one of utf8, base16, base32, or base64.
- **nonce** – An arbitrary nonce string that conforms to base.
- **child** – The expression to wrap.

`type_of` ()
Get the return type of this expression.

class `pyteal.UnaryExpr` (*op: pyteal.Op, inputType: pyteal.TealType, outputType: pyteal.TealType, arg: pyteal.Expr*)

Bases: `pyteal.Expr`

An expression with a single argument.

`type_of` ()
Get the return type of this expression.

`pyteal.Btoi` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Convert a byte string to a uint64.

`pyteal.Itob` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Convert a uint64 string to a byte string.

`pyteal.Len` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the length of a byte string.

`pyteal.Sha256` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the SHA-256 hash of a byte string.

`pyteal.Sha512_256` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the SHA-512/256 hash of a byte string.

`pyteal.Keccak256` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the KECCAK-256 hash of a byte string.

`pyteal.Not` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the logical inverse of a uint64.

If the argument is 0, then this will produce 1. Otherwise this will produce 0.

`pyteal.BitwiseNot` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the bitwise inverse of a uint64.

Produces ~arg.

`pyteal.Pop` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Pop a value from the stack.

`pyteal.Return` (*arg: pyteal.Expr*) → `pyteal.UnaryExpr`
Immediately exit the program with the given success value.

`pyteal.Balance` (*account: pyteal.Expr*) → `pyteal.UnaryExpr`
Get the balance of a user in microAlgos.

Argument must be an index into Txn.Accounts that corresponds to the account to read from. It must evaluate to uint64.

This operation is only permitted in application mode.

`pyteal.MinBalance` (*account*: `pyteal.Expr`) → `pyteal.UnaryExpr`
Get the minimum balance of a user in microAlgos.

For more information about minimum balances, see: <https://developer.algorand.org/docs/features/accounts/#minimum-balance>

Argument must be an index into `Txn.Accounts` that corresponds to the account to read from. It must evaluate to `uint64`.

Requires TEAL version 3 or higher. This operation is only permitted in application mode.

class `pyteal.BinaryExpr` (*op*: `pyteal.Op`, *inputType*: `Union[pyteal.TealType, Tuple[pyteal.TealType, pyteal.TealType]]`, *outputType*: `pyteal.TealType`, *argLeft*: `pyteal.Expr`, *argRight*: `pyteal.Expr`)

Bases: `pyteal.Expr`

An expression with two arguments.

type_of ()
Get the return type of this expression.

`pyteal.Add` (*left*: `pyteal.Expr`, *right*: `pyteal.Expr`) → `pyteal.BinaryExpr`
Add two numbers.

Produces left + right.

Parameters

- **left** – Must evaluate to `uint64`.
- **right** – Must evaluate to `uint64`.

`pyteal.Minus` (*left*: `pyteal.Expr`, *right*: `pyteal.Expr`) → `pyteal.BinaryExpr`
Subtract two numbers.

Produces left - right.

Parameters

- **left** – Must evaluate to `uint64`.
- **right** – Must evaluate to `uint64`.

`pyteal.Mul` (*left*: `pyteal.Expr`, *right*: `pyteal.Expr`) → `pyteal.BinaryExpr`
Multiply two numbers.

Produces left * right.

Parameters

- **left** – Must evaluate to `uint64`.
- **right** – Must evaluate to `uint64`.

`pyteal.Div` (*left*: `pyteal.Expr`, *right*: `pyteal.Expr`) → `pyteal.BinaryExpr`
Divide two numbers.

Produces left / right.

Parameters

- **left** – Must evaluate to `uint64`.
- **right** – Must evaluate to `uint64`.

`pyteal.BitwiseAnd` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Bitwise and expression.

Produces left & right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.BitwiseOr` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Bitwise or expression.

Produces left | right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.BitwiseXor` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Bitwise xor expression.

Produces left ^ right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.Mod` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Modulo expression.

Produces left % right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.Eq` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Equality expression.

Checks if left == right.

Parameters

- **left** – A value to check.
- **right** – The other value to check. Must evaluate to the same type as left.

`pyteal.Neq` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Difference expression.

Checks if left != right.

Parameters

- **left** – A value to check.
- **right** – The other value to check. Must evaluate to the same type as left.

`pyteal.Lt` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Less than expression.

Checks if left < right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.Le` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Less than or equal to expression.

Checks if left <= right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.Gt` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Greater than expression.

Checks if left > right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.Ge` (*left: pyteal.Expr, right: pyteal.Expr*) → `pyteal.BinaryExpr`
Greater than or equal to expression.

Checks if left >= right.

Parameters

- **left** – Must evaluate to uint64.
- **right** – Must evaluate to uint64.

`pyteal.GetBit` (*value: pyteal.Expr, index: pyteal.Expr*) → `pyteal.BinaryExpr`
Get the bit value of an expression at a specific index.

The meaning of index differs if value is an integer or a byte string.

- For integers, bit indexing begins with low-order bits. For example, `GetBit(Int(16), Int(4))` yields 1. Any other valid index would yield a bit value of 0. Any integer less than 64 is a valid index.
- For byte strings, bit indexing begins at the first byte. For example, `GetBit(Bytes("base16", "0xf0"), Int(0))` yields 1. Any index less than 4 would yield 1, and any valid index 4 or greater would yield 0. Any integer less than `8*Len(value)` is a valid index.

Requires TEAL version 3 or higher.

Parameters

- **value** – The value containing bits. Can evaluate to any type.
- **index** – The index of the bit to extract. Must evaluate to uint64.

`pyteal.GetByte` (*value: pyteal.Expr, index: pyteal.Expr*) → `pyteal.BinaryExpr`
Extract a single byte as an integer from a byte string.

Similar to `GetBit`, indexing begins at the first byte. For example, `GetByte(Bytes("base16", "0xff0000"), Int(0))` yields 255. Any other valid index would yield 0.

Requires TEAL version 3 or higher.

Parameters

- **value** – The value containing the bytes. Must evaluate to bytes.
- **index** – The index of the byte to extract. Must evaluate to an integer less than `Len(value)`.

`pyteal.Ed25519Verify` (*data: pyteal.Expr, sig: pyteal.Expr, key: pyteal.Expr*) → `pyteal.TernaryExpr`
Verify the ed25519 signature of the concatenation (“ProgData” + `hash_of_current_program` + *data*).

Parameters

- **data** – The data signed by the public key. Must evaluate to bytes.
- **sig** – The proposed 64-byte signature of the concatenation (“ProgData” + `hash_of_current_program` + *data*). Must evaluate to bytes.
- **key** – The 32 byte public key that produced the signature. Must evaluate to bytes.

`pyteal.Substring` (*string: pyteal.Expr, start: pyteal.Expr, end: pyteal.Expr*) → `pyteal.TernaryExpr`
Take a substring of a byte string.

Produces a new byte string consisting of the bytes starting at *start* up to but not including *end*.

Parameters

- **string** – The byte string.
- **start** – The starting index for the substring. Must be an integer less than or equal to `Len(string)`.
- **end** – The ending index for the substring. Must be an integer greater or equal to *start*, but less than or equal to `Len(string)`.

`pyteal.SetBit` (*value: pyteal.Expr, index: pyteal.Expr, newBitValue: pyteal.Expr*) → `pyteal.TernaryExpr`
Set the bit value of an expression at a specific index.

The meaning of *index* differs if *value* is an integer or a byte string.

- For integers, bit indexing begins with low-order bits. For example, `SetBit(Int(0), Int(4), Int(1))` yields the integer 16 (2^4). Any integer less than 64 is a valid index.
- For byte strings, bit indexing begins at the first byte. For example, `SetBit(Bytes("base16", "0x00"), Int(7), Int(1))` yields the byte string 0x01. Any integer less than $8 * \text{Len}(\text{value})$ is a valid index.

Requires TEAL version 3 or higher.

Parameters

- **value** – The value containing bits. Can evaluate to any type.
- **index** – The index of the bit to set. Must evaluate to `uint64`.
- **newBitValue** – The new bit value to set. Must evaluate to the integer 0 or 1.

`pyteal.SetByte` (*value: pyteal.Expr, index: pyteal.Expr, newByteValue: pyteal.Expr*) → `pyteal.TernaryExpr`
Set a single byte in a byte string from an integer value.

Similar to `SetBit`, indexing begins at the first byte. For example, `SetByte(Bytes("base16", "0x000000"), Int(0), Int(255))` yields the byte string 0xff0000.

Requires TEAL version 3 or higher.

Parameters

- **value** – The value containing the bytes. Must evaluate to bytes.
- **index** – The index of the byte to set. Must evaluate to an integer less than `Len(value)`.
- **newByteValue** – The new byte value to set. Must evaluate to an integer less than 256.

class `pyteal.NaryExpr` (*op: pyteal.Op, inputType: pyteal.TealType, outputType: pyteal.TealType, args: Sequence[pyteal.Expr]*)

Bases: `pyteal.Expr`

N-ary expression base class.

This type of expression takes an arbitrary number of arguments.

type_of()

Get the return type of this expression.

`pyteal.And(*args) → pyteal.NaryExpr`

Logical and expression.

Produces 1 if all arguments are nonzero. Otherwise produces 0.

All arguments must be PyTeal expressions that evaluate to uint64, and there must be at least two arguments.

Example

```
And(Txn.amount() == Int(500), Txn.fee() <= Int(10))
```

`pyteal.Or(*args) → pyteal.NaryExpr`

Logical or expression.

Produces 1 if any argument is nonzero. Otherwise produces 0.

All arguments must be PyTeal expressions that evaluate to uint64, and there must be at least two arguments.

`pyteal.Concat(*args) → pyteal.NaryExpr`

Concatenate byte strings.

Produces a new byte string consisting of the contents of each of the passed in byte strings joined together.

All arguments must be PyTeal expressions that evaluate to bytes, and there must be at least two arguments.

Example

```
Concat(Bytes("hello"), Bytes(" "), Bytes("world"))
```

class `pyteal.If` (*cond: pyteal.Expr, thenBranch: pyteal.Expr, elseBranch: pyteal.Expr = None*)

Bases: `pyteal.Expr`

Simple two-way conditional expression.

__init__ (*cond: pyteal.Expr, thenBranch: pyteal.Expr, elseBranch: pyteal.Expr = None*) → None

Create a new If expression.

When this If expression is executed, the condition will be evaluated, and if it produces a true value, thenBranch will be executed and used as the return value for this expression. Otherwise, elseBranch will be executed and used as the return value, if it is provided.

Parameters

- **cond** – The condition to check. Must evaluate to uint64.

- **thenBranch** – Expression to evaluate if the condition is true.
- **elseBranch** (*optional*) – Expression to evaluate if the condition is false. Must evaluate to the same type as thenBranch, if provided. Defaults to None.

type_of()

Get the return type of this expression.

class `pyteal.Cond(*argv)`

Bases: `pyteal.Expr`

A chainable branching expression that supports an arbitrary number of conditions.

__init__ (**argv*)

Create a new Cond expression.

At least one argument must be provided, and each argument must be a list with two elements. The first element is a condition which evaluates to `uint64`, and the second is the body of the condition, which will execute if that condition is true. All condition bodies must have the same return type. During execution, each condition is tested in order, and the first condition to evaluate to a true value will cause its associated body to execute and become the value for this Cond expression. If no condition evaluates to a true value, the Cond expression produces an error and the TEAL program terminates.

Example

```
Cond([Global.group_size() == Int(5), bid],
     [Global.group_size() == Int(4), redeem],
     [Global.group_size() == Int(1), wrapup])
```

type_of()

Get the return type of this expression.

class `pyteal.Seq(exprs: List[pyteal.Expr])`

Bases: `pyteal.Expr`

A control flow expression to represent a sequence of expressions.

__init__ (*exprs: List[pyteal.Expr]*)

Create a new Seq expression.

The new Seq expression will take on the return value of the final expression in the sequence.

Parameters *exprs* – The expressions to include in this sequence. All expressions that are not the final one in this list must not return any values.

Example

```
Seq([
    App.localPut(Bytes("key"), Bytes("value")),
    Int(1)
])
```

type_of()

Get the return type of this expression.

class `pyteal.Assert(cond: pyteal.Expr)`

Bases: `pyteal.Expr`

A control flow expression to verify that a condition is true.

`__init__` (*cond*: *pyteal.Expr*) → None

Create an assert statement that raises an error if the condition is false.

Parameters *cond* – The condition to check. Must evaluate to a uint64.

`type_of` ()

Get the return type of this expression.

class *pyteal.ScratchSlot*

Bases: *object*

Represents the allocation of a scratch space slot.

`load` (*type*: *pyteal.TealType* = <*TealType.anytype*: 2>) → *pyteal.ScratchLoad*

Get an expression to load a value from this slot.

Parameters *type* (*optional*) – The type being loaded from this slot, if known. Defaults to *TealType.anytype*.

`slotId` = 0

`store` (*value*: *pyteal.Expr* = None) → *pyteal.Expr*

Get an expression to store a value in this slot.

Parameters

- **value** (*optional*) – The value to store in this slot. If not included, the last value on
- **stack will be stored.** **NOTE** (*the*) – storing the last value on the stack breaks the typical
- **of PyTeal, only use if you know what you're doing.** (*semantics*) –

class *pyteal.ScratchLoad* (*slot*: *pyteal.ScratchSlot*, *type*: *pyteal.TealType* = <*TealType.anytype*: 2>)

Bases: *pyteal.Expr*

Expression to load a value from scratch space.

`__init__` (*slot*: *pyteal.ScratchSlot*, *type*: *pyteal.TealType* = <*TealType.anytype*: 2>)

Create a new *ScratchLoad* expression.

Parameters

- **slot** – The slot to load the value from.
- **type** (*optional*) – The type being loaded from this slot, if known. Defaults to *TealType.anytype*.

`type_of` ()

Get the return type of this expression.

class *pyteal.ScratchStore* (*slot*: *pyteal.ScratchSlot*, *value*: *pyteal.Expr*)

Bases: *pyteal.Expr*

Expression to store a value in scratch space.

`__init__` (*slot*: *pyteal.ScratchSlot*, *value*: *pyteal.Expr*)

Create a new *ScratchStore* expression.

Parameters

- **slot** – The slot to store the value in.
- **value** – The value to store.

type_of()

Get the return type of this expression.

class `pyteal.ScratchStackStore` (*slot: pyteal.ScratchSlot*)

Bases: `pyteal.Expr`

Expression to store a value from the stack in scratch space.

NOTE: This expression breaks the typical semantics of PyTeal, only use if you know what you're doing.

__init__ (*slot: pyteal.ScratchSlot*)

Create a new ScratchStackStore expression.

Parameters **slot** – The slot to store the value in.

type_of()

Get the return type of this expression.

class `pyteal.ScratchVar` (*type: pyteal.TealType = <TealType.anytype: 2>*)

Bases: `object`

Interface around Scratch space, similar to get/put local/global state

Example

```
myvar = ScratchVar(TealType.uint64)
Seq([
    myvar.store(Int(5)),
    Assert(myvar.load() == Int(5))
])
```

__init__ (*type: pyteal.TealType = <TealType.anytype: 2>*)

Create a new ScratchVar with an optional type.

Parameters **type** (*optional*) – The type that this variable can hold. An error will be thrown if an expression with an incompatible type is stored in this variable. Defaults to `TealType.anytype`.

load() → `pyteal.ScratchLoad`

Load value from Scratch Space

storage_type() → `pyteal.TealType`

Get the type of expressions that can be stored in this ScratchVar.

store (*value: pyteal.Expr*) → `pyteal.Expr`

Store value in Scratch Space

Parameters **value** – The value to store. Must conform to this ScratchVar's type.

class `pyteal.MaybeValue` (*op: pyteal.Op, type: pyteal.TealType, *, immediate_args: List[Union[int, str]] = None, args: List[pyteal.Expr] = None*)

Bases: `pyteal.LeafExpr`

Represents a get operation returning a value that may not exist.

__init__ (*op: pyteal.Op, type: pyteal.TealType, *, immediate_args: List[Union[int, str]] = None, args: List[pyteal.Expr] = None*)

Create a new MaybeValue.

Parameters

- **op** – The operation that returns values.

- **type** – The type of the returned value.
- **immediate_args** (*optional*) – Immediate arguments for the op. Defaults to None.
- **args** (*optional*) – Stack arguments for the op. Defaults to None.

hasValue () → pyteal.ScratchLoad

Check if the value exists.

This will return 1 if the value exists, otherwise 0.

type_of ()

Get the return type of this expression.

value () → pyteal.ScratchLoad

Get the value.

If the value exists, it will be returned. Otherwise, the zero value for this type will be returned (i.e. either 0 or an empty byte string, depending on the type).

class pyteal.Op

Bases: enum.Enum

Enum of program opcodes.

add = OpType(value='+', mode=<Mode.Application|Signature: 3>, min_version=2)

addr = OpType(value='addr', mode=<Mode.Application|Signature: 3>, min_version=2)

addw = OpType(value='addw', mode=<Mode.Application|Signature: 3>, min_version=2)

app_global_del = OpType(value='app_global_del', mode=<Mode.Application: 2>, min_version=2)

app_global_get = OpType(value='app_global_get', mode=<Mode.Application: 2>, min_version=2)

app_global_get_ex = OpType(value='app_global_get_ex', mode=<Mode.Application: 2>, min_version=2)

app_global_put = OpType(value='app_global_put', mode=<Mode.Application: 2>, min_version=2)

app_local_del = OpType(value='app_local_del', mode=<Mode.Application: 2>, min_version=2)

app_local_get = OpType(value='app_local_get', mode=<Mode.Application: 2>, min_version=2)

app_local_get_ex = OpType(value='app_local_get_ex', mode=<Mode.Application: 2>, min_version=2)

app_local_put = OpType(value='app_local_put', mode=<Mode.Application: 2>, min_version=2)

app_opted_in = OpType(value='app_opted_in', mode=<Mode.Application: 2>, min_version=2)

arg = OpType(value='arg', mode=<Mode.Signature: 1>, min_version=2)

assert_ = OpType(value='assert', mode=<Mode.Application|Signature: 3>, min_version=3)

asset_holding_get = OpType(value='asset_holding_get', mode=<Mode.Application: 2>, min_version=2)

asset_params_get = OpType(value='asset_params_get', mode=<Mode.Application: 2>, min_version=2)

b = OpType(value='b', mode=<Mode.Application|Signature: 3>, min_version=2)

balance = OpType(value='balance', mode=<Mode.Application: 2>, min_version=2)

bitwise_and = OpType(value='&', mode=<Mode.Application|Signature: 3>, min_version=2)

bitwise_not = OpType(value='~', mode=<Mode.Application|Signature: 3>, min_version=2)

bitwise_or = OpType(value='|', mode=<Mode.Application|Signature: 3>, min_version=2)

bitwise_xor = OpType(value='^', mode=<Mode.Application|Signature: 3>, min_version=2)

bnz = OpType(value='bnz', mode=<Mode.Application|Signature: 3>, min_version=2)

```

btoi = OpType(value='btoi', mode=<Mode.Application|Signature: 3>, min_version=2)
byte = OpType(value='byte', mode=<Mode.Application|Signature: 3>, min_version=2)
bz = OpType(value='bz', mode=<Mode.Application|Signature: 3>, min_version=2)
concat = OpType(value='concat', mode=<Mode.Application|Signature: 3>, min_version=2)
dig = OpType(value='dig', mode=<Mode.Application|Signature: 3>, min_version=3)
div = OpType(value='/', mode=<Mode.Application|Signature: 3>, min_version=2)
dup = OpType(value='dup', mode=<Mode.Application|Signature: 3>, min_version=2)
dup2 = OpType(value='dup2', mode=<Mode.Application|Signature: 3>, min_version=2)
ed25519verify = OpType(value='ed25519verify', mode=<Mode.Signature: 1>, min_version=2)
eq = OpType(value='==', mode=<Mode.Application|Signature: 3>, min_version=2)
err = OpType(value='err', mode=<Mode.Application|Signature: 3>, min_version=2)
ge = OpType(value='>=', mode=<Mode.Application|Signature: 3>, min_version=2)
getbit = OpType(value='getbit', mode=<Mode.Application|Signature: 3>, min_version=3)
getbyte = OpType(value='getbyte', mode=<Mode.Application|Signature: 3>, min_version=3)
global_ = OpType(value='global', mode=<Mode.Application|Signature: 3>, min_version=2)
gt = OpType(value='>', mode=<Mode.Application|Signature: 3>, min_version=2)
gtxn = OpType(value='gtxn', mode=<Mode.Application|Signature: 3>, min_version=2)
gtxna = OpType(value='gtxna', mode=<Mode.Application|Signature: 3>, min_version=2)
gtxns = OpType(value='gtxns', mode=<Mode.Application|Signature: 3>, min_version=3)
gtxnsa = OpType(value='gtxnsa', mode=<Mode.Application|Signature: 3>, min_version=3)
int = OpType(value='int', mode=<Mode.Application|Signature: 3>, min_version=2)
itob = OpType(value='itob', mode=<Mode.Application|Signature: 3>, min_version=2)
keccak256 = OpType(value='keccak256', mode=<Mode.Application|Signature: 3>, min_version=2)
le = OpType(value='<=', mode=<Mode.Application|Signature: 3>, min_version=2)
len = OpType(value='len', mode=<Mode.Application|Signature: 3>, min_version=2)
load = OpType(value='load', mode=<Mode.Application|Signature: 3>, min_version=2)
logic_and = OpType(value='&&', mode=<Mode.Application|Signature: 3>, min_version=2)
logic_not = OpType(value='!', mode=<Mode.Application|Signature: 3>, min_version=2)
logic_or = OpType(value='||', mode=<Mode.Application|Signature: 3>, min_version=2)
lt = OpType(value='<', mode=<Mode.Application|Signature: 3>, min_version=2)
min_balance = OpType(value='min_balance', mode=<Mode.Application: 2>, min_version=3)
min_version
    Get the minimum version where this op is available.
minus = OpType(value='-', mode=<Mode.Application|Signature: 3>, min_version=2)
mod = OpType(value='%', mode=<Mode.Application|Signature: 3>, min_version=2)
mode
    Get the modes where this op is available.

```

```

mul = OpType(value='*', mode=<Mode.Application|Signature: 3>, min_version=2)
mulw = OpType(value='mulw', mode=<Mode.Application|Signature: 3>, min_version=2)
neq = OpType(value='!=', mode=<Mode.Application|Signature: 3>, min_version=2)
pop = OpType(value='pop', mode=<Mode.Application|Signature: 3>, min_version=2)
pushbytes = OpType(value='pushbytes', mode=<Mode.Application|Signature: 3>, min_version=2)
pushint = OpType(value='pushint', mode=<Mode.Application|Signature: 3>, min_version=3)
return_ = OpType(value='return', mode=<Mode.Application|Signature: 3>, min_version=2)
select = OpType(value='select', mode=<Mode.Application|Signature: 3>, min_version=3)
setbit = OpType(value='setbit', mode=<Mode.Application|Signature: 3>, min_version=3)
setbyte = OpType(value='setbyte', mode=<Mode.Application|Signature: 3>, min_version=3)
sha256 = OpType(value='sha256', mode=<Mode.Application|Signature: 3>, min_version=2)
sha512_256 = OpType(value='sha512_256', mode=<Mode.Application|Signature: 3>, min_version=2)
store = OpType(value='store', mode=<Mode.Application|Signature: 3>, min_version=2)
substring = OpType(value='substring', mode=<Mode.Application|Signature: 3>, min_version=3)
substring3 = OpType(value='substring3', mode=<Mode.Application|Signature: 3>, min_version=3)
swap = OpType(value='swap', mode=<Mode.Application|Signature: 3>, min_version=3)
txn = OpType(value='txn', mode=<Mode.Application|Signature: 3>, min_version=2)
txna = OpType(value='txna', mode=<Mode.Application|Signature: 3>, min_version=2)

class pyteal.Mode
    Bases: enum.Flag

    Enum of program running modes.

    Application = 2
    Signature = 1

class pyteal.TealComponent (expr: Optional[Expr])
    Bases: abc.ABC

    class Context
        Bases: object

        class EqualityContext
            Bases: contextlib.AbstractContextManager

            checkExpr = True

            classmethod ignoreExprEquality()

        assemble() → str

        assignSlot(slot: ScratchSlot, location: int)

        getSlots() → List[ScratchSlot]

class pyteal.TealOp (expr: Optional[Expr], op: pyteal.Op, *args)
    Bases: pyteal.TealComponent

    assemble() → str

    assignSlot(slot: ScratchSlot, location: int)

```

getOp () → `pyteal.Op`

getSlots () → `List[ScratchSlot]`

class `pyteal.TealLabel` (*expr: Optional[Expr], label: str*)

Bases: `pyteal.TealComponent`

assemble () → `str`

class `pyteal.TealBlock` (*ops: List[pyteal.TealOp]*)

Bases: `abc.ABC`

Represents a basic block of TealComponents in a graph.

classmethod **FromOp** (*options: CompileOptions, op: pyteal.TealOp, *args*) → `Tuple[TealBlock, TealSimpleBlock]`

Create a path of blocks from a TealOp and its arguments.

Returns The starting and ending block of the path that encodes the given TealOp and arguments.

classmethod **Iterate** (*start: pyteal.TealBlock*) → `Iterator[pyteal.TealBlock]`

Perform a depth-first search of the graph of blocks starting with start.

classmethod **NormalizeBlocks** (*start: pyteal.TealBlock*) → `pyteal.TealBlock`

Minimize the number of blocks in the graph of blocks starting with start by combining sequential blocks. This operation does not alter the operations of the graph or the functionality of its underlying program, however it does mutate the input graph.

Returns The new starting point of the altered graph. May be the same or different than start.

addIncoming (*parent: Optional[pyteal.TealBlock] = None, visited: List[TealBlock] = None*) → `None`

Calculate the parent blocks for this block and its children.

Parameters

- **parent** (*optional*) – The parent block to this one, if it has one. Defaults to None.
- **visited** (*optional*) – Used internally to remember blocks that have been visited. Set to None.

getOutgoing () → `List[pyteal.TealBlock]`

Get this block's children blocks, if any.

isTerminal () → `bool`

Check if this block ends the program.

replaceOutgoing (*oldBlock: pyteal.TealBlock, newBlock: pyteal.TealBlock*) → `None`

Replace one of this block's child blocks.

validateSlots (*slotsInUse: Set[ScratchSlot] = None, visited: Set[Tuple[int, ...]] = None*) → `List[pyteal.TealCompileError]`

validateTree (*parent: Optional[pyteal.TealBlock] = None, visited: List[TealBlock] = None*) → `None`

Check that this block and its children have valid parent pointers.

Parameters

- **parent** (*optional*) – The parent block to this one, if it has one. Defaults to None.
- **visited** (*optional*) – Used internally to remember blocks that have been visited. Set to None.

class `pyteal.TealSimpleBlock` (*ops: List[pyteal.TealOp]*)

Bases: `pyteal.TealBlock`

Represents a basic block of TealComponents in a graph that does not contain a branch condition.

getOutgoing () → List[pyteal.TealBlock]

Get this block's children blocks, if any.

replaceOutgoing (oldBlock: pyteal.TealBlock, newBlock: pyteal.TealBlock) → None

Replace one of this block's child blocks.

setNextBlock (block: pyteal.TealBlock) → None

Set the block that follows this one.

class pyteal.TealConditionalBlock (ops: List[pyteal.TealOp])

Bases: pyteal.TealBlock

Represents a basic block of TealComponents in a graph ending with a branch condition.

getOutgoing () → List[pyteal.TealBlock]

Get this block's children blocks, if any.

replaceOutgoing (oldBlock: pyteal.TealBlock, newBlock: pyteal.TealBlock) → None

Replace one of this block's child blocks.

setFalseBlock (block: pyteal.TealBlock) → None

Set the block that this one should branch to if its condition is false.

setTrueBlock (block: pyteal.TealBlock) → None

Set the block that this one should branch to if its condition is true.

class pyteal.CompileOptions (*, mode: pyteal.Mode = <Mode.Signature: 1>, version: int = 2)

Bases: object

pyteal.compileTeal (ast: pyteal.Expr, mode: pyteal.Mode, *, version: int = 2) → str

Compile a PyTeal expression into TEAL assembly.

Parameters

- **ast** – The PyTeal expression to assemble.
- **mode** – The mode of the program to assemble. Must be Signature or Application.
- **version** (*optional*) – The TEAL version used to assemble the program. This will determine which expressions and fields are able to be used in the program and how expressions compile to TEAL opcodes. Defaults to 2 if not included.

Returns A TEAL assembly program compiled from the input expression.

Raises *TealInputError* – if an operation in ast is not supported by the supplied mode and version.

class pyteal.TealType

Bases: enum.Enum

Teal type enum.

anytype = 2

bytes = 1

none = 3

uint64 = 0

exception pyteal.TealInternalError (message: str)

Bases: Exception

exception pyteal.TealTypeError (actual, expected)

Bases: Exception

```
exception pyteal.TealInputError (msg: str)  
    Bases: Exception
```

```
exception pyteal.TealCompileError (msg: str, sourceExpr: Optional[Expr])  
    Bases: Exception
```

CHAPTER 15

Indices and tables

- `genindex`

p

pyteal, [61](#)

Symbols

__getitem__() (pyteal.Array method), 75
 __getitem__() (pyteal.TxnArray method), 65
 __getitem__() (pyteal.TxnGroup method), 70
 __init__() (pyteal.Addr method), 61
 __init__() (pyteal.Arg method), 63
 __init__() (pyteal.Assert method), 82
 __init__() (pyteal.Bytes method), 62
 __init__() (pyteal.Cond method), 82
 __init__() (pyteal.EnumInt method), 62
 __init__() (pyteal.If method), 81
 __init__() (pyteal.Int method), 62
 __init__() (pyteal.MaybeValue method), 84
 __init__() (pyteal.Nonce method), 75
 __init__() (pyteal.ScratchLoad method), 83
 __init__() (pyteal.ScratchStackStore method), 84
 __init__() (pyteal.ScratchStore method), 83
 __init__() (pyteal.ScratchVar method), 84
 __init__() (pyteal.Seq method), 82

A

accounts (pyteal.TxnField attribute), 63
 accounts (pyteal.TxnObject attribute), 65
 add (pyteal.Op attribute), 85
 Add() (in module pyteal), 77
 addIncoming() (pyteal.TealBlock method), 88
 Addr (class in pyteal), 61
 addr (pyteal.Op attribute), 85
 Addr() (pyteal.Tmpl class method), 75
 addw (pyteal.Op attribute), 85
 amount (pyteal.TxnField attribute), 63
 amount() (pyteal.TxnObject method), 65
 And() (in module pyteal), 81
 And() (pyteal.Expr method), 61
 anytype (pyteal.TealType attribute), 89
 App (class in pyteal), 71
 app_global_del (pyteal.Op attribute), 85
 app_global_get (pyteal.Op attribute), 85
 app_global_get_ex (pyteal.Op attribute), 85

app_global_put (pyteal.Op attribute), 85
 app_local_del (pyteal.Op attribute), 85
 app_local_get (pyteal.Op attribute), 85
 app_local_get_ex (pyteal.Op attribute), 85
 app_local_put (pyteal.Op attribute), 85
 app_opted_in (pyteal.Op attribute), 85
 AppField (class in pyteal), 73
 Application (pyteal.Mode attribute), 87
 application_args (pyteal.TxnField attribute), 63
 application_args (pyteal.TxnObject attribute), 65
 application_id (pyteal.TxnField attribute), 63
 application_id() (pyteal.TxnObject method), 65
 ApplicationCall (pyteal.TxnType attribute), 63
 applications (pyteal.TxnField attribute), 63
 applications (pyteal.TxnObject attribute), 66
 approval_program (pyteal.TxnField attribute), 63
 approval_program() (pyteal.TxnObject method), 66
 Arg (class in pyteal), 62
 arg (pyteal.Op attribute), 85
 Array (class in pyteal), 75
 assemble() (pyteal.TealComponent method), 87
 assemble() (pyteal.TealLabel method), 88
 assemble() (pyteal.TealOp method), 87
 Assert (class in pyteal), 82
 assert_ (pyteal.Op attribute), 85
 asset_amount (pyteal.TxnField attribute), 63
 asset_amount() (pyteal.TxnObject method), 66
 asset_close_to (pyteal.TxnField attribute), 63
 asset_close_to() (pyteal.TxnObject method), 66
 asset_holding_get (pyteal.Op attribute), 85
 asset_params_get (pyteal.Op attribute), 85
 asset_receiver (pyteal.TxnField attribute), 63
 asset_receiver() (pyteal.TxnObject method), 66
 asset_sender (pyteal.TxnField attribute), 63
 asset_sender() (pyteal.TxnObject method), 66
 AssetConfig (pyteal.TxnType attribute), 63
 AssetFreeze (pyteal.TxnType attribute), 63
 AssetHolding (class in pyteal), 73
 AssetParam (class in pyteal), 74

assets (*pyteal.TxnField attribute*), 63
 assets (*pyteal.TxnObject attribute*), 66
 AssetTransfer (*pyteal.TxnType attribute*), 63
 assignSlot () (*pyteal.TealComponent method*), 87
 assignSlot () (*pyteal.TealOp method*), 87

B

b (*pyteal.Op attribute*), 85
 balance (*pyteal.Op attribute*), 85
 Balance () (*in module pyteal*), 76
 balance () (*pyteal.AssetHolding class method*), 73
 BinaryExpr (*class in pyteal*), 77
 bitwise_and (*pyteal.Op attribute*), 85
 bitwise_not (*pyteal.Op attribute*), 85
 bitwise_or (*pyteal.Op attribute*), 85
 bitwise_xor (*pyteal.Op attribute*), 85
 BitwiseAnd () (*in module pyteal*), 77
 BitwiseNot () (*in module pyteal*), 76
 BitwiseOr () (*in module pyteal*), 78
 BitwiseXor () (*in module pyteal*), 78
 bnz (*pyteal.Op attribute*), 85
 btoi (*pyteal.Op attribute*), 85
 Btoi () (*in module pyteal*), 76
 byte (*pyteal.Op attribute*), 86
 Bytes (*class in pyteal*), 62
 bytes (*pyteal.TealType attribute*), 89
 Bytes () (*pyteal.Tmpl class method*), 75
 bz (*pyteal.Op attribute*), 86

C

checkExpr (*pyteal.TealComponent.Context attribute*), 87
 clawback () (*pyteal.AssetParam class method*), 74
 clear_state_program (*pyteal.TxnField attribute*), 63
 clear_state_program () (*pyteal.TxnObject method*), 66
 ClearState (*pyteal.OnComplete attribute*), 73
 close_remainder_to (*pyteal.TxnField attribute*), 63
 close_remainder_to () (*pyteal.TxnObject method*), 66
 CloseOut (*pyteal.OnComplete attribute*), 73
 CompileOptions (*class in pyteal*), 89
 compileTeal () (*in module pyteal*), 89
 concat (*pyteal.Op attribute*), 86
 Concat () (*in module pyteal*), 81
 Cond (*class in pyteal*), 82
 config_asset (*pyteal.TxnField attribute*), 63
 config_asset () (*pyteal.TxnObject method*), 66
 config_asset_clawback (*pyteal.TxnField attribute*), 63
 config_asset_clawback () (*pyteal.TxnObject method*), 67

config_asset_decimals (*pyteal.TxnField attribute*), 64
 config_asset_decimals () (*pyteal.TxnObject method*), 67
 config_asset_default_frozen (*pyteal.TxnField attribute*), 64
 config_asset_default_frozen () (*pyteal.TxnObject method*), 67
 config_asset_freeze (*pyteal.TxnField attribute*), 64
 config_asset_freeze () (*pyteal.TxnObject method*), 67
 config_asset_manager (*pyteal.TxnField attribute*), 64
 config_asset_manager () (*pyteal.TxnObject method*), 67
 config_asset_metadata_hash (*pyteal.TxnField attribute*), 64
 config_asset_metadata_hash () (*pyteal.TxnObject method*), 67
 config_asset_name (*pyteal.TxnField attribute*), 64
 config_asset_name () (*pyteal.TxnObject method*), 67
 config_asset_reserve (*pyteal.TxnField attribute*), 64
 config_asset_reserve () (*pyteal.TxnObject method*), 67
 config_asset_total (*pyteal.TxnField attribute*), 64
 config_asset_total () (*pyteal.TxnObject method*), 67
 config_asset_unit_name (*pyteal.TxnField attribute*), 64
 config_asset_unit_name () (*pyteal.TxnObject method*), 67
 config_asset_url (*pyteal.TxnField attribute*), 64
 config_asset_url () (*pyteal.TxnObject method*), 68
 creator_address (*pyteal.GlobalField attribute*), 71
 creator_address () (*pyteal.Global class method*), 70
 current_app_id (*pyteal.GlobalField attribute*), 71
 current_application_id () (*pyteal.Global class method*), 70

D

decimals () (*pyteal.AssetParam class method*), 74
 defaultFrozen () (*pyteal.AssetParam class method*), 74
 DeleteApplication (*pyteal.OnComplete attribute*), 73
 dig (*pyteal.Op attribute*), 86
 div (*pyteal.Op attribute*), 86
 Div () (*in module pyteal*), 77
 dup (*pyteal.Op attribute*), 86

dup2 (*pyteal.Op* attribute), 86

E

ed25519verify (*pyteal.Op* attribute), 86
 Ed25519Verify () (in module *pyteal*), 80
 EnumInt (class in *pyteal*), 62
 eq (*pyteal.Op* attribute), 86
 Eq () (in module *pyteal*), 78
 Err (class in *pyteal*), 62
 err (*pyteal.Op* attribute), 86
 Expr (class in *pyteal*), 61

F

fee (*pyteal.TxnField* attribute), 64
 fee () (*pyteal.TxnObject* method), 68
 first_valid (*pyteal.TxnField* attribute), 64
 first_valid () (*pyteal.TxnObject* method), 68
 first_valid_time (*pyteal.TxnField* attribute), 64
 freeze () (*pyteal.AssetParam* class method), 74
 freeze_asset (*pyteal.TxnField* attribute), 64
 freeze_asset () (*pyteal.TxnObject* method), 68
 freeze_asset_account (*pyteal.TxnField* attribute), 64
 freeze_asset_account () (*pyteal.TxnObject* method), 68
 freeze_asset_frozen (*pyteal.TxnField* attribute), 64
 freeze_asset_frozen () (*pyteal.TxnObject* method), 68
 FromOp () (*pyteal.TealBlock* class method), 88
 frozen () (*pyteal.AssetHolding* class method), 74

G

ge (*pyteal.Op* attribute), 86
 Ge () (in module *pyteal*), 79
 get_op () (*pyteal.AppField* method), 73
 getbit (*pyteal.Op* attribute), 86
 GetBit () (in module *pyteal*), 79
 getbyte (*pyteal.Op* attribute), 86
 GetByte () (in module *pyteal*), 79
 getDefinitionTrace () (*pyteal.Expr* method), 61
 getOp () (*pyteal.TealOp* method), 87
 getOutgoing () (*pyteal.TealBlock* method), 88
 getOutgoing () (*pyteal.TealConditionalBlock* method), 89
 getOutgoing () (*pyteal.TealSimpleBlock* method), 88
 getSlots () (*pyteal.TealComponent* method), 87
 getSlots () (*pyteal.TealOp* method), 88
 Global (class in *pyteal*), 70
 global_ (*pyteal.Op* attribute), 86
 global_num_byte_slices (*pyteal.TxnField* attribute), 64
 global_num_byte_slices () (*pyteal.TxnObject* method), 68

global_num_uints (*pyteal.TxnField* attribute), 64
 global_num_uints () (*pyteal.TxnObject* method), 68
 globalDel (*pyteal.AppField* attribute), 73
 globalDel () (*pyteal.App* class method), 71
 GlobalField (class in *pyteal*), 71
 globalGet (*pyteal.AppField* attribute), 73
 globalGet () (*pyteal.App* class method), 71
 globalGetEx (*pyteal.AppField* attribute), 73
 globalGetEx () (*pyteal.App* class method), 72
 globalPut (*pyteal.AppField* attribute), 73
 globalPut () (*pyteal.App* class method), 72
 group_index (*pyteal.TxnField* attribute), 64
 group_index () (*pyteal.TxnObject* method), 68
 group_size (*pyteal.GlobalField* attribute), 71
 group_size () (*pyteal.Global* class method), 70
 gt (*pyteal.Op* attribute), 86
 Gt () (in module *pyteal*), 79
 Gtxn (in module *pyteal*), 61
 gtxn (*pyteal.Op* attribute), 86
 gtxna (*pyteal.Op* attribute), 86
 GtxnaExpr (class in *pyteal*), 70
 GtxnaExpr (class in *pyteal*), 70
 gtxns (*pyteal.Op* attribute), 86
 gtxnsa (*pyteal.Op* attribute), 86

H

hasValue () (*pyteal.MaybeValue* method), 85

I

id () (*pyteal.App* class method), 72
 If (class in *pyteal*), 81
 ignoreExprEquality () (*pyteal.TealComponent.Context* class method), 87
 Int (class in *pyteal*), 62
 int (*pyteal.Op* attribute), 86
 Int () (*pyteal.Tmpl* class method), 75
 isTerminal () (*pyteal.TealBlock* method), 88
 Iterate () (*pyteal.TealBlock* class method), 88
 itob (*pyteal.Op* attribute), 86
 Itob () (in module *pyteal*), 76

K

keccak256 (*pyteal.Op* attribute), 86
 Keccak256 () (in module *pyteal*), 76
 KeyRegistration (*pyteal.TxnType* attribute), 63

L

last_valid (*pyteal.TxnField* attribute), 64
 last_valid () (*pyteal.TxnObject* method), 68
 latest_timestamp (*pyteal.GlobalField* attribute), 71

latest_timestamp() (pyteal.Global class method), 71
 le (pyteal.Op attribute), 86
 Le() (in module pyteal), 79
 LeafExpr (class in pyteal), 61
 lease (pyteal.TxnField attribute), 64
 lease() (pyteal.TxnObject method), 68
 len (pyteal.Op attribute), 86
 Len() (in module pyteal), 76
 length() (pyteal.Array method), 75
 length() (pyteal.TxnArray method), 65
 load (pyteal.Op attribute), 86
 load() (pyteal.ScratchSlot method), 83
 load() (pyteal.ScratchVar method), 84
 local_num_byte_slices (pyteal.TxnField attribute), 64
 local_num_byte_slices() (pyteal.TxnObject method), 69
 local_num_uints (pyteal.TxnField attribute), 64
 local_num_uints() (pyteal.TxnObject method), 69
 localDel (pyteal.AppField attribute), 73
 localDel() (pyteal.App class method), 72
 localGet (pyteal.AppField attribute), 73
 localGet() (pyteal.App class method), 72
 localGetEx (pyteal.AppField attribute), 73
 localGetEx() (pyteal.App class method), 72
 localPut (pyteal.AppField attribute), 73
 localPut() (pyteal.App class method), 72
 logic_and (pyteal.Op attribute), 86
 logic_not (pyteal.Op attribute), 86
 logic_or (pyteal.Op attribute), 86
 logic_sig_version (pyteal.GlobalField attribute), 71
 logic_sig_version() (pyteal.Global class method), 71
 lt (pyteal.Op attribute), 86
 Lt() (in module pyteal), 78

M

manager() (pyteal.AssetParam class method), 74
 max_txn_life (pyteal.GlobalField attribute), 71
 max_txn_life() (pyteal.Global class method), 71
 MaybeValue (class in pyteal), 84
 metadataHash() (pyteal.AssetParam class method), 74
 min_balance (pyteal.GlobalField attribute), 71
 min_balance (pyteal.Op attribute), 86
 min_balance() (pyteal.Global class method), 71
 min_txn_fee (pyteal.GlobalField attribute), 71
 min_txn_fee() (pyteal.Global class method), 71
 min_version (pyteal.Op attribute), 86
 MinBalance() (in module pyteal), 77
 minus (pyteal.Op attribute), 86
 Minus() (in module pyteal), 77

mod (pyteal.Op attribute), 86
 Mod() (in module pyteal), 78
 Mode (class in pyteal), 87
 mode (pyteal.Op attribute), 86
 mul (pyteal.Op attribute), 86
 Mul() (in module pyteal), 77
 mulw (pyteal.Op attribute), 87

N

name() (pyteal.AssetParam class method), 74
 NaryExpr (class in pyteal), 81
 neq (pyteal.Op attribute), 87
 Neq() (in module pyteal), 78
 Nonce (class in pyteal), 75
 none (pyteal.TealType attribute), 89
 NoOp (pyteal.OnComplete attribute), 73
 NormalizeBlocks() (pyteal.TealBlock class method), 88
 Not() (in module pyteal), 76
 note (pyteal.TxnField attribute), 64
 note() (pyteal.TxnObject method), 69
 num_accounts (pyteal.TxnField attribute), 64
 num_app_args (pyteal.TxnField attribute), 64
 num_applications (pyteal.TxnField attribute), 64
 num_assets (pyteal.TxnField attribute), 64

O

on_completion (pyteal.TxnField attribute), 64
 on_completion() (pyteal.TxnObject method), 69
 OnComplete (class in pyteal), 73
 Op (class in pyteal), 85
 optedIn (pyteal.AppField attribute), 73
 optedIn() (pyteal.App class method), 73
 OptIn (pyteal.OnComplete attribute), 73
 Or() (in module pyteal), 81
 Or() (pyteal.Expr method), 61

P

Payment (pyteal.TxnType attribute), 63
 pop (pyteal.Op attribute), 87
 Pop() (in module pyteal), 76
 pushbytes (pyteal.Op attribute), 87
 pushint (pyteal.Op attribute), 87
 pyteal (module), 61

R

receiver (pyteal.TxnField attribute), 64
 receiver() (pyteal.TxnObject method), 69
 rekey_to (pyteal.TxnField attribute), 64
 rekey_to() (pyteal.TxnObject method), 69
 replaceOutgoing() (pyteal.TealBlock method), 88
 replaceOutgoing() (pyteal.TealConditionalBlock method), 89

`replaceOutgoing()` (*pyteal.TealSimpleBlock* method), 89
`reserve()` (*pyteal.AssetParam* class method), 74
`Return()` (in module *pyteal*), 76
`return_` (*pyteal.Op* attribute), 87
RFC
 RFC 4648#section-4, 30
 RFC 4648#section-6, 30
 RFC 4648#section-8, 29
`round` (*pyteal.GlobalField* attribute), 71
`round()` (*pyteal.Global* class method), 71

S

`ScratchLoad` (class in *pyteal*), 83
`ScratchSlot` (class in *pyteal*), 83
`ScratchStackStore` (class in *pyteal*), 84
`ScratchStore` (class in *pyteal*), 83
`ScratchVar` (class in *pyteal*), 84
`select` (*pyteal.Op* attribute), 87
`selection_pk` (*pyteal.TxnField* attribute), 64
`selection_pk()` (*pyteal.TxnObject* method), 69
`sender` (*pyteal.TxnField* attribute), 64
`sender()` (*pyteal.TxnObject* method), 69
`Seq` (class in *pyteal*), 82
`setbit` (*pyteal.Op* attribute), 87
`SetBit()` (in module *pyteal*), 80
`setbyte` (*pyteal.Op* attribute), 87
`SetByte()` (in module *pyteal*), 80
`setFalseBlock()` (*pyteal.TealConditionalBlock* method), 89
`setNextBlock()` (*pyteal.TealSimpleBlock* method), 89
`setTrueBlock()` (*pyteal.TealConditionalBlock* method), 89
`sha256` (*pyteal.Op* attribute), 87
`Sha256()` (in module *pyteal*), 76
`sha512_256` (*pyteal.Op* attribute), 87
`Sha512_256()` (in module *pyteal*), 76
`Signature` (*pyteal.Mode* attribute), 87
`slotId` (*pyteal.ScratchSlot* attribute), 83
`storage_type()` (*pyteal.ScratchVar* method), 84
`store` (*pyteal.Op* attribute), 87
`store()` (*pyteal.ScratchSlot* method), 83
`store()` (*pyteal.ScratchVar* method), 84
`substring` (*pyteal.Op* attribute), 87
`Substring()` (in module *pyteal*), 80
`substring3` (*pyteal.Op* attribute), 87
`swap` (*pyteal.Op* attribute), 87

T

`TealBlock` (class in *pyteal*), 88
`TealCompileError`, 90
`TealComponent` (class in *pyteal*), 87
`TealComponent.Context` (class in *pyteal*), 87
`TealComponent.Context.EqualityContext` (class in *pyteal*), 87
`TealConditionalBlock` (class in *pyteal*), 89
`TealInputError`, 89
`TealInternalError`, 89
`TealLabel` (class in *pyteal*), 88
`TealOp` (class in *pyteal*), 87
`TealSimpleBlock` (class in *pyteal*), 88
`TealType` (class in *pyteal*), 89
`TealTypeError`, 89
`Tmpl` (class in *pyteal*), 75
`total()` (*pyteal.AssetParam* class method), 75
`tx_id` (*pyteal.TxnField* attribute), 64
`tx_id()` (*pyteal.TxnObject* method), 69
`Txn` (in module *pyteal*), 61
`txn` (*pyteal.Op* attribute), 87
`txna` (*pyteal.Op* attribute), 87
`TxnaExpr` (class in *pyteal*), 65
`TxnArray` (class in *pyteal*), 65
`TxnExpr` (class in *pyteal*), 65
`TxnField` (class in *pyteal*), 63
`TxnGroup` (class in *pyteal*), 70
`TxnObject` (class in *pyteal*), 65
`TxnType` (class in *pyteal*), 63
`type` (*pyteal.TxnField* attribute), 64
`type()` (*pyteal.TxnObject* method), 69
`type_enum` (*pyteal.TxnField* attribute), 64
`type_enum()` (*pyteal.TxnObject* method), 69
`type_of()` (*pyteal.Addr* method), 62
`type_of()` (*pyteal.App* method), 73
`type_of()` (*pyteal.AppField* method), 73
`type_of()` (*pyteal.Arg* method), 63
`type_of()` (*pyteal.Assert* method), 83
`type_of()` (*pyteal.BinaryExpr* method), 77
`type_of()` (*pyteal.Bytes* method), 62
`type_of()` (*pyteal.Cond* method), 82
`type_of()` (*pyteal.EnumInt* method), 62
`type_of()` (*pyteal.Err* method), 62
`type_of()` (*pyteal.Expr* method), 61
`type_of()` (*pyteal.Global* method), 71
`type_of()` (*pyteal.GlobalField* method), 71
`type_of()` (*pyteal.If* method), 82
`type_of()` (*pyteal.Int* method), 62
`type_of()` (*pyteal.MaybeValue* method), 85
`type_of()` (*pyteal.NaryExpr* method), 81
`type_of()` (*pyteal.Nonce* method), 76
`type_of()` (*pyteal.ScratchLoad* method), 83
`type_of()` (*pyteal.ScratchStackStore* method), 84
`type_of()` (*pyteal.ScratchStore* method), 83
`type_of()` (*pyteal.Seq* method), 82
`type_of()` (*pyteal.Tmpl* method), 75
`type_of()` (*pyteal.TxnaExpr* method), 65
`type_of()` (*pyteal.TxnExpr* method), 65
`type_of()` (*pyteal.TxnField* method), 65

`type_of()` (*pyteal.UnaryExpr method*), 76

U

`uint64` (*pyteal.TealType attribute*), 89

`UnaryExpr` (*class in pyteal*), 76

`unitName()` (*pyteal.AssetParam class method*), 75

`Unknown` (*pyteal.TxnType attribute*), 63

`UpdateApplication` (*pyteal.OnComplete attribute*), 73

`url()` (*pyteal.AssetParam class method*), 75

V

`validateSlots()` (*pyteal.TealBlock method*), 88

`validateTree()` (*pyteal.TealBlock method*), 88

`value()` (*pyteal.MaybeValue method*), 85

`vote_first` (*pyteal.TxnField attribute*), 65

`vote_first()` (*pyteal.TxnObject method*), 69

`vote_key_dilution` (*pyteal.TxnField attribute*), 65

`vote_key_dilution()` (*pyteal.TxnObject method*), 70

`vote_last` (*pyteal.TxnField attribute*), 65

`vote_last()` (*pyteal.TxnObject method*), 70

`vote_pk` (*pyteal.TxnField attribute*), 65

`vote_pk()` (*pyteal.TxnObject method*), 70

X

`xfer_asset` (*pyteal.TxnField attribute*), 65

`xfer_asset()` (*pyteal.TxnObject method*), 70

Z

`zero_address` (*pyteal.GlobalField attribute*), 71

`zero_address()` (*pyteal.Global class method*), 71