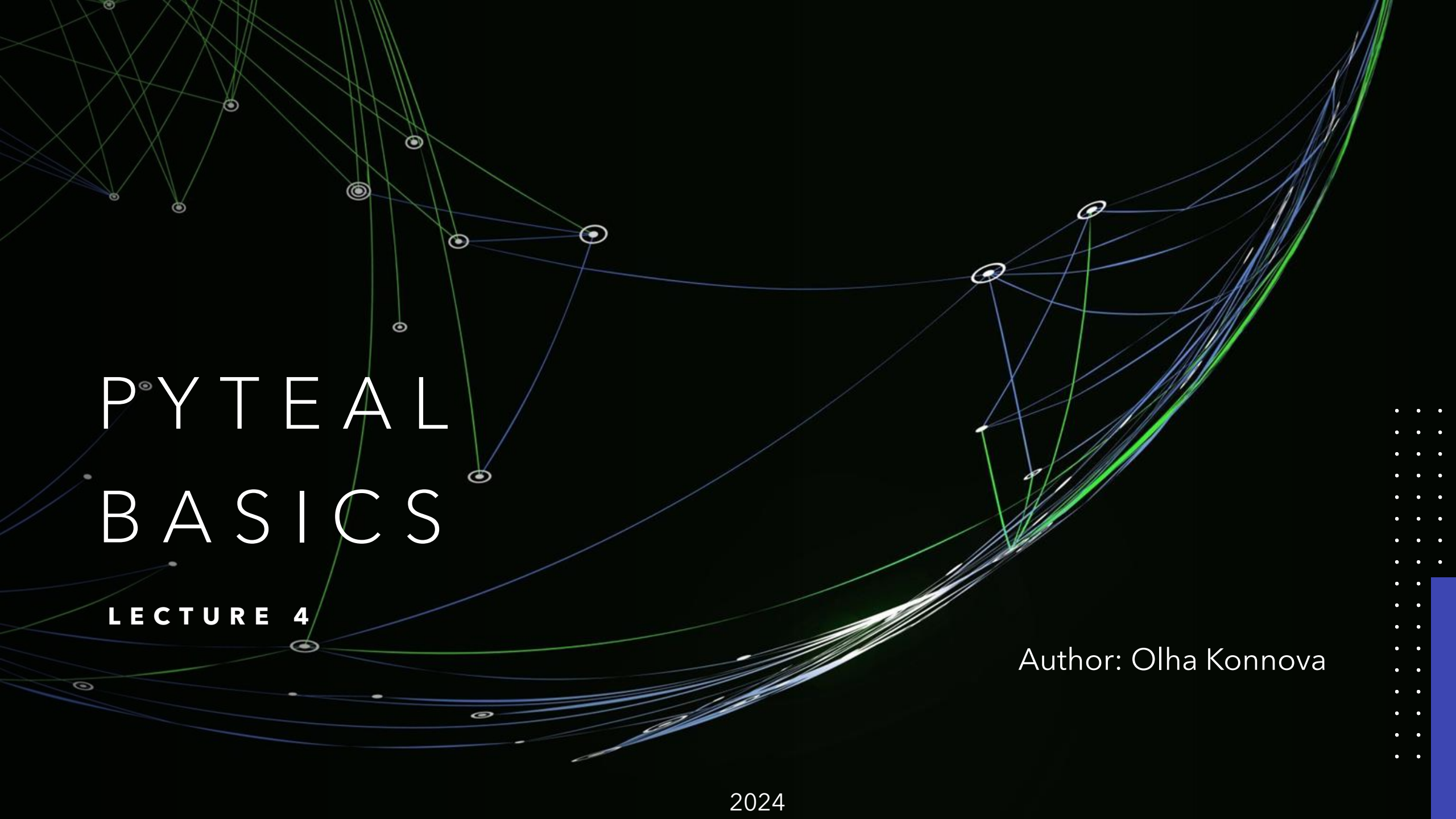


PYTEAL BASICS

LECTURE 4

Author: Olha Konnova

2024

P L A N

1. Accessing and changing the application state
2. ABI
3. Router
4. Subroutines

ACCESS AND CHANGE APPLICATION STATE

APPLICATION STATE TYPES

There are several types of state that an application can use. State consists of key-value pairs, where the keys are byte fragments and the values vary depending on the type of state.

Type of State	Type of Key	Type of Value
Global State	`TealType.bytes`	`TealType.uint64` or `TealType.bytes`
Local State	`TealType.bytes`	`TealType.uint64` or `TealType.bytes`
Boxes	`TealType.bytes`	`TealType.bytes`

STATE OPERATIONS TABLE

	Create	Write	Read	Delete	Check If Exists
Global variables of the current application		App.globalPut	App.globalGet	App.globalDel	App.globalGetEx
Local variables of the current application		App.localPut	App.localGet	App.localDel	App.localGetEx
Global variables of another application			App.globalGetEx		App.globalGetEx
Local variables of another application			App.localGetEx		App.localGetEx
Current application boxes	App.box_create App.box_put	App.box_put App.box_replace App.box_splice App.box_resize	App.box_extract App.box_get	App.box_delete	App.box_length App.box_get

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

WRITING VALUES TO THE APPLICATION'S GLOBAL STORAGE

Global state consists of **key-value pairs** that are stored in the global context of the application.

To write values to the application's global storage, you need to use the function

App.globalPut(key: Expr, value: Expr)

The first argument is the key to write (of type bytes), and the second argument is the value to write (can be of any type). For example:

App.globalPut(Bytes("status"), Bytes("active")) # write a byte slice

App.globalPut(Bytes("total supply"), Int(100)) # write a uint64

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

GETTING VALUES FROM THE APPLICATION'S GLOBAL STORAGE

To read from global state, use the function

App.globalGet(key: Expr)

The only required argument is the key to read.

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 application's global state, the integer 0 is returned.

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

DELETING VALUES FROM THE APPLICATION'S GLOBAL STORAGE

To remove a key from the global state, use the function

App.globalDel (key: Expr)

The only argument required is the key to remove.

For example:

App.globalDel(Bytes("status"))

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

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

WRITING VALUES TO THE APPLICATION'S LOCAL STORAGE

Local state consists of **key-value pairs** that are stored in a unique context for **each account** that has connected to your application. As a result, you will need to specify the account when manipulating local state.

To read or manipulate the local state of an account, that account **must be represented** in the Txn.accounts array.

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

WRITING VALUES TO THE APPLICATION'S LOCAL STORAGE

To write a value to the local account state, use the function

App.localPut(account: Expr, key: Expr, value: Expr)

The first argument is the address of the account to write to, the second argument is the key to write to, and the third argument is the value to write.

Example:

App.localPut(Txn.sender(), Bytes("role"), Bytes("admin")) # write a value to the sender account

App.localPut(Txn.sender(), Bytes("balance"), Int(10)) # write a value of type uint64 to the sender account

App.localPut(Txn.accounts[1], Bytes("balance"), Int(10)) # write a value of type uint64 to Txn.account[1]

*Writing to the local account state is only possible if this account is opted-in to your application. If the account is not activated, the program will terminate with an error.

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

GETTING VALUES FROM APPLICATION'S LOCAL STORAGE

To get a value from the local account state, use the function:

App.localGet(account: Expr, key: Expr)

The first argument is the address of the account to read from, and the second argument is the key to read from.

For example:

App.localGet(Txn.sender(), Bytes("role")) #get the value from the sender account

App.localGet(Txn.sender(), Bytes("balance")) #get the value from the sender account

App.localGet(Txn.accounts[1], Bytes("balance")) #get the value from Txn.accounts[1]

*If you try to read a key that does not exist in the local account state, the integer 0 is returned.

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

REMOVING A KEY FROM APPLICATION'S LOCAL STORAGE

To delete a key from the local account state, use the function:

App.localDel (account: Expr, key: Expr).

The first argument is the address of the corresponding account, and the second is the key to delete.

For example:

App.localDel(Txn.sender(), Bytes("role")) #delete "role" from the sender account

App.localDel(Txn.sender(), Bytes("balance")) #delete "balance" from the sender account

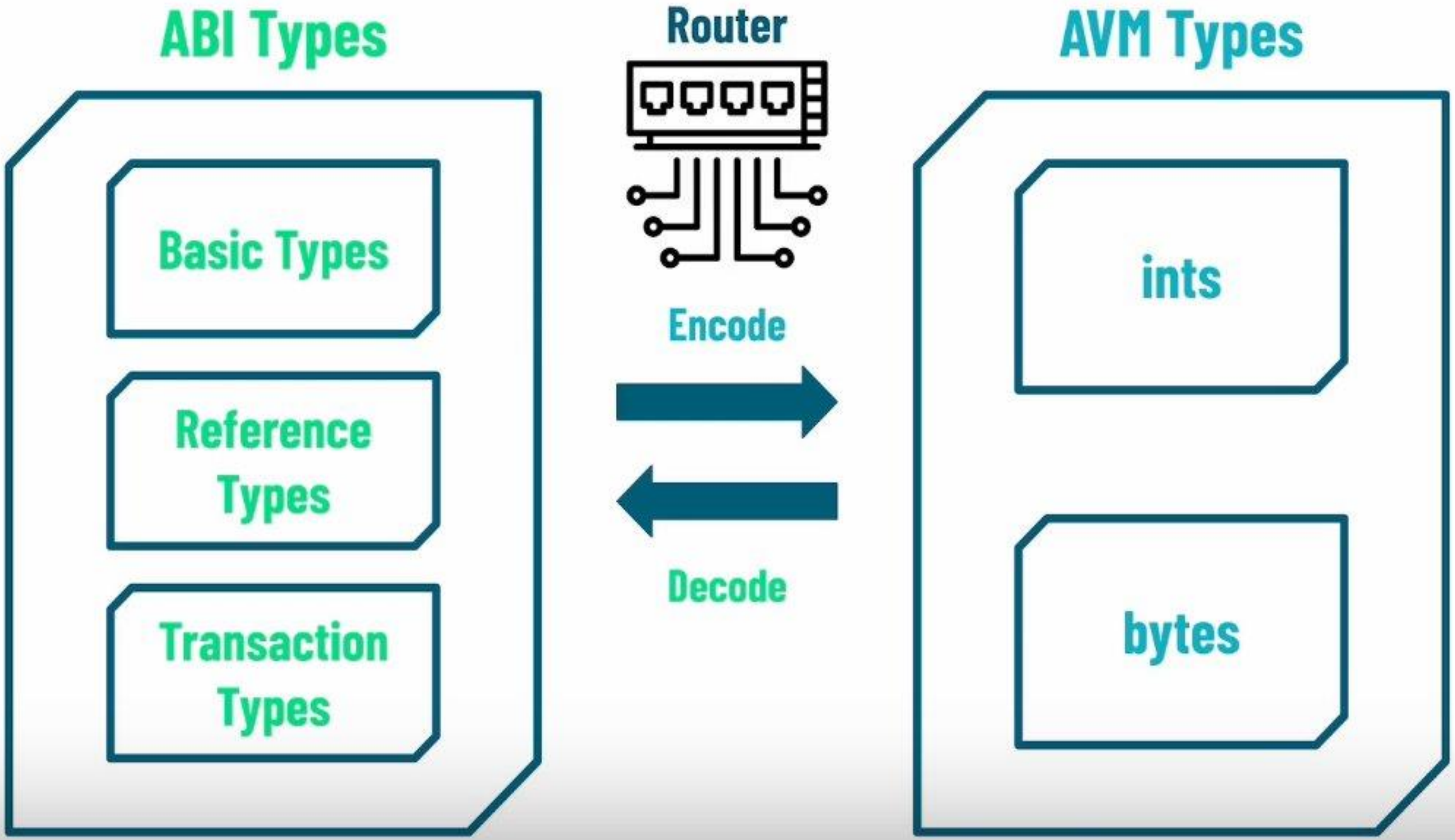
App.localDel(Txn.accounts[1], Bytes("balance")) # delete "balance" from the account Txn.accounts[1]

If you try to delete a key that does not exist in the local account state, nothing will happen.

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

ABI
(APPLICATION BINARY INTERFACE)

ABI (Application Binary Interface) is a specification that defines the encoding/decoding of data types and a standard for calling methods in a smart contract.


ABI Types
Basic Types
Reference Types
Transaction Types
Router
Encode
Decode
AVM Types
ints
bytes

ROUTER

- Router is a mechanism in PyTeal for organizing method calls in Algorand smart contracts.
- It simplifies the routing of transactions to the appropriate subroutines.
- Provides compatibility with Application Binary Interface (ABI).

BASIC ROUTER FUNCTIONS

- **Method registration:** binds methods to actions (OnCompleteAction).
- **ABI call handling:** decodes transaction input.
- **Action support:**
  - createOnly: for creating an application.
  - callOnly: for usual calls.
  - always: for all transactions.
  - never: the method is not called.

EXAMPLE "FIRST ROUTER" – COUNTER

A simple smart contract to increment the value of Count.

Router routes ABI calls to methods.

1. Router receives call: transaction arrives at Router.
2. @router.method decorator: automatically registers increment in Router for routing.
3. Update Count: increments Count by 1.

```
count_key = Bytes("Count")
handle_creation = Seq(App.globalPut(count_key, Int(0)), Approve())

# Router initialization
router = Router(  # <--- 1. Router receives the call
    "my-first-router",
    BareCallActions(
        no_op=OnCompleteAction.create_only(handle_creation),
        update_application=OnCompleteAction.always(Reject()),
        delete_application=OnCompleteAction.always(Reject()),
        close_out=OnCompleteAction.never(),
        opt_in=OnCompleteAction.never(),
    ),
)

# Method to increment Count, added to Router
@router.method  # <--- 2. Decorator automatically registers the method in Router
def increment():
    scratchCount = ScratchVar(TealType.uint64)
    return Seq(
        Assert(Global.group_size() == Int(1)),
        scratchCount.store(App.globalGet(count_key)),
        App.globalPut(count_key, scratchCount.load() + Int(1)),  # <--- 3.
Updating Count
    )
```

Transaction --> Router --> increment() --> Update Count

SUBROUTINES IN PYTEAL

01

Subroutines allow you to create modular code in smart contracts.

02

Used with the @Subroutine decorator to define functions.

03

Help avoid code duplication and simplify logic.

1. **@Subroutine decorator:** defines a subroutine that returns TealType.none (no value).
2. **State update:** subroutine changes global variable Count.
3. **Router integration:** increment method is called via Router.
4. **Subroutine call:** increment_count() is called inside the method.

```
# Subroutine definition
@Subroutine(TealType.none)  # <--- 1. Decorator for the subroutine
def increment_count():
    scratchCount = ScratchVar(TealType.uint64)
    return Seq(
        scratchCount.store(App.globalGet(Bytes("Count"))),
        App.globalPut(Bytes("Count"), scratchCount.load() + Int(1)),  # <--- 2.
Updating state
    )

# Usage in Router
router = Router("my-first-router", BareCallActions(...))
@router.method  # <--- 3. Calling the subroutine via Router
def increment():
    return Seq(
        Assert(Global.group_size() == Int(1)),
        increment_count(),  # <--- 4. Subroutine call
    )
```

Transaction --> Router --> increment() --> increment count() --> Update Count

ADDITIONAL MATERIALS

- Pyteal for beginners (Video Course). In this course, you will learn how to get started building Algorand smart contracts with PyTeal and dive into each of the PyTeal operations you will need to write complex and powerful smart contracts.
- Algorand PyTeal Github repository. This repository contains examples of smart contracts and smart signatures written in PyTeal.
- Fundamentals of PyTeal (Course). This PyTeal course provides a comprehensive introduction to writing smart contracts for the Algorand blockchain using PyTeal. Students will learn key concepts such as routers, expressions, AVM and ABI data types, transaction fields, atomic transfers, and subroutines. An NFT certificate is available.