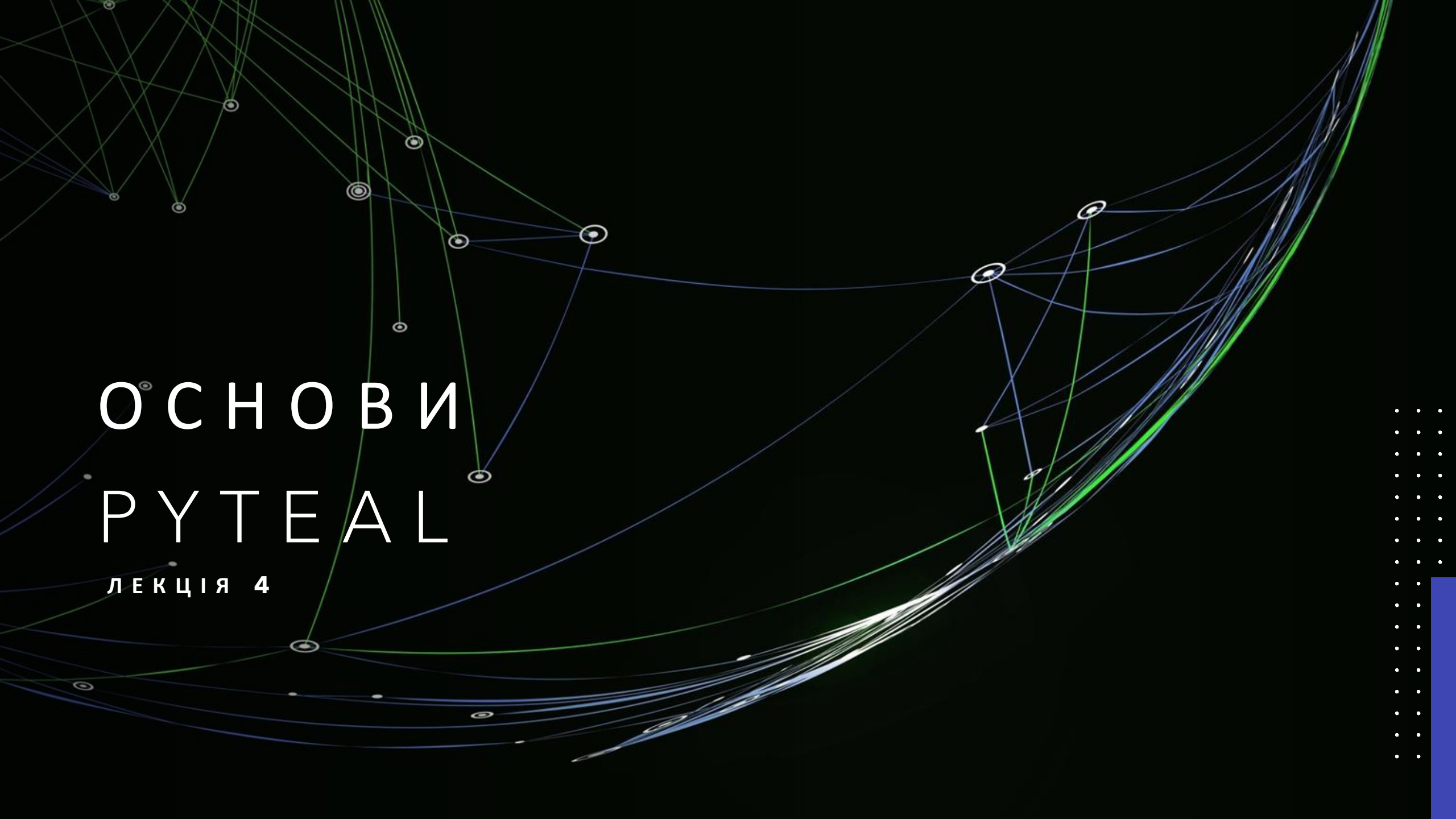


ОСНОВИ PYTEAL

ЛЕКЦІЯ 4

П Л А Н

1. Доступ до стану додатку та його зміна
2. ABI
3. Router
4. Підпрограми

ДОСТУП ДО СТАНУ ДОДАТКУ ТА ЙОГО ЗМІНА

ТИПИ СТАНІВ ДОДАТКУ

Існує кілька типів стану, які може використовувати програма. Стан складається з пар ключ-значення, де ключі є фрагментами байтів, а значення змінюються залежно від типу стану.

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`

ТАБЛИЦЯ ОПЕРАЦІЙ СТАНІВ

	Create	Write	Read	Delete	Check If Exists
Глобальні змінні поточного додатку		App.globalPut	App.globalGet	App.globalDel	App.globalGetEx
Локальні змінні поточного додатку		App.localPut	App.localGet	App.localDel	App.localGetEx
Глобальні змінні іншого додатку			App.globalGetEx		App.globalGetEx
Локальні змінні іншого додатку			App.localGetEx		App.localGetEx
Бокси поточного додатку	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

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

ЗАПИС ЗНАЧЕНЬ У ГЛОБАЛЬНЕ СХОВИЩЕ ДОДАТКУ

Глобальний стан складається з пар ключ-значення, які зберігаються в глобальному контексті програми.

Для запису значень в глобальне сховище додатку, необхідно використовувати функцію

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

Перший аргумент – це ключ для запису (типу bytes), а другий аргумент – значення для запису (може бути будь-якого типу). Наприклад:

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

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

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

ОТРИМАННЯ ЗНАЧЕНЬ З ГЛОБАЛЬНОГО СХОВИЩА ДОДАТКУ

Для читання з глобального стану використовуйте функцію

App.globalGet(key: Expr)

Єдиний необхідний аргумент – це ключ для читання.

Наприклад:

App.globalGet(Bytes("status"))

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

Якщо ви спробуєте прочитати з ключа, який не існує в глобальному стані вашої програми, повертається ціле число 0.

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

ВИДАЛЕННЯ ЗНАЧЕНЬ З ГЛОБАЛЬНОГО СТАНУ ДОДАТКУ

Щоб видалити ключ із глобального стану, скористайтеся функцією

```
App.globalDel (key: Expr)
```

Єдиний аргумент, який потрібен, це ключ для видалення.

Наприклад:

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

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

ЗАПИС ЗНАЧЕНЬ У ЛОКАЛЬНЕ СХОВИЩЕ ДОДАТКУ

Локальний стан складається з пар **ключ-значення**, які зберігаються в унікальному контексті для **кожного облікового запису**, який підключився до вашого додатку. У результаті вам потрібно буде вказати обліковий запис під час маніпулювання локальним станом.

Щоб читати або маніпулювати локальним станом облікового запису, цей обліковий запис **має бути представлено в масиві** Txn.accounts.

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

ЗАПИС ЗНАЧЕНЬ У ЛОКАЛЬНЕ СХОВИЩЕ ДОДАТКУ

Щоб записати значення у локальний стан облікового запису, скористайтеся функцією

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

Перший аргумент – це адреса облікового запису, у який потрібно писати, другий аргумент – це ключ, куди потрібно писати, а третій аргумент – це значення, яке потрібно записати.

Приклад:

```
App.localPut(Txn.sender(), Bytes("role"),
Bytes("admin")) # записати значення в обліковий запис
```
відправника

```
App.localPut(Txn.sender(), Bytes("balance"), Int(10)) #
```
записати значення типу uint64 в обліковий запис відправника

```
App.localPut(Txn.accounts[1], Bytes("balance"),
Int(10)) # записати значення типу uint64 до Txn.account[1]
```

*Запис у локальний стан акаунту можливий лише в тому випадку, якщо цей обліковий запис підключився до вашого додатку. Якщо обліковий запис не активовано, програма завершиться з помилкою.

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

ОТРИМАННЯ ЗНАЧЕНЬ З ЛОКАЛЬНОГО СТАНУ АКАУНТ

Щоб отримати значення з локального стану облікового запису, скористайтеся функцією:

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

Перший аргумент – це адреса облікового запису, з якого потрібно читати, а другий аргумент – ключ для читання.

Наприклад:

```
App.localGet(Txn.sender(), Bytes("role"))
```
#отримати значення з акаунту відправника

```
App.localGet(Txn.sender(), Bytes("balance"))
```
#отримати значення з акаунту відправника

```
App.localGet(Txn.accounts[1], Bytes("balance"))
```
#отримати значення з Txn.accounts[1]

*Якщо ви намагаєтеся прочитати ключ, який не існує в локальному стані облікового запису, повертається ціле число 0.

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

ВИДАЛЕННЯ КЛЮЧА З ЛОКАЛЬНОГО СТАНУ АКАУНТА

Щоб видалити ключ із локального стану облікового запису, скористайтеся функцією:

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

Перший аргумент – це адреса відповідного облікового запису, а другий – ключ для видалення.

Наприклад:

`App.localDel(Txn.sender(), Bytes("role"))` #видалити «role» з облікового запису відправника

`App.localDel(Txn.sender(), Bytes("balance"))` #видалити «balance» з облікового запису відправника

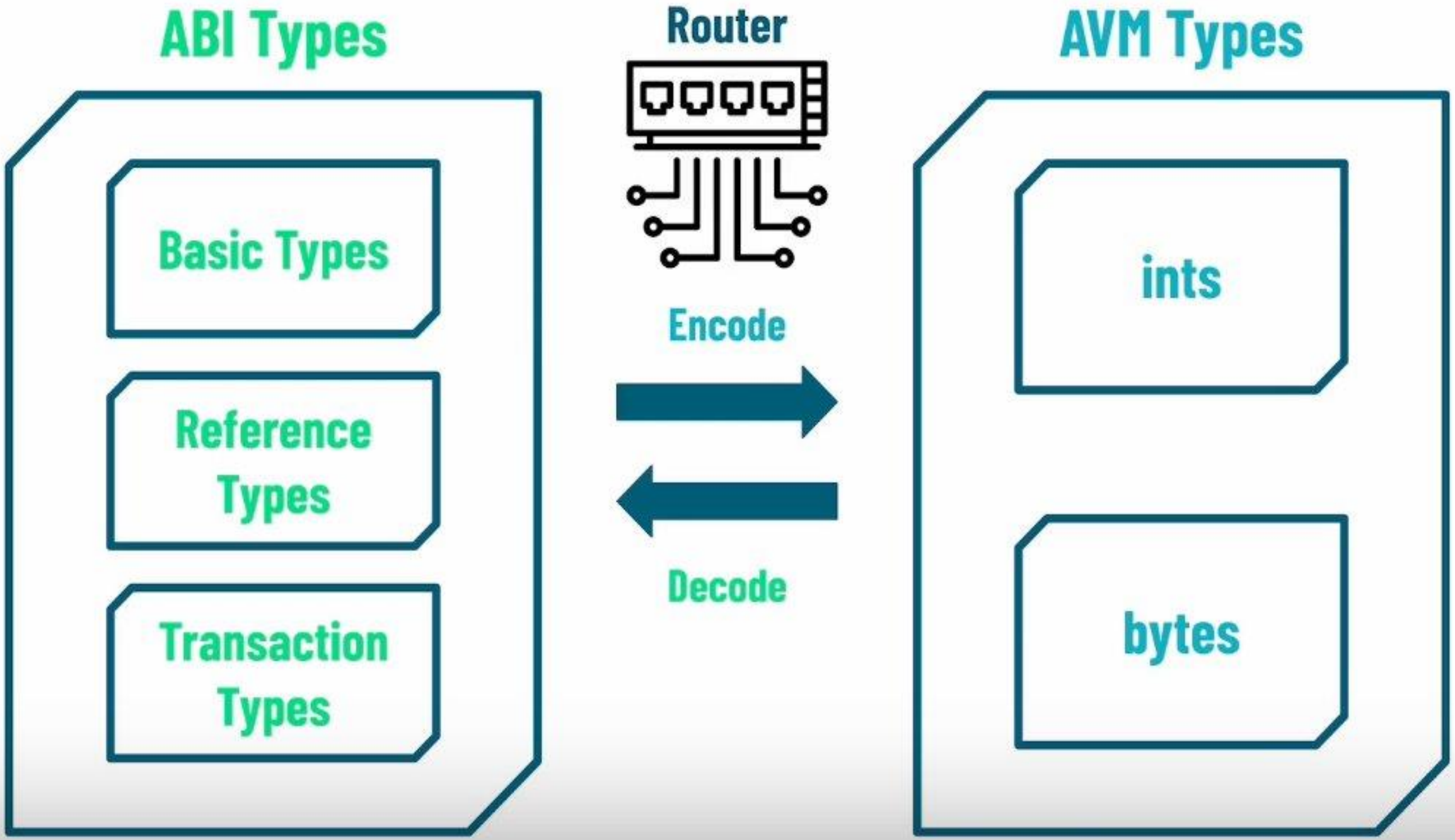
`App.localDel(Txn.accounts[1], Bytes("balance"))` # видалити «balance» з облікового запису Txn.accounts[1]

Якщо ви спробуєте видалити ключ, який не існує в локальному стані облікового запису, нічого не відбудеться.

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

ABI
(APPLICATION BINARY INTERFACE)

ABI (Application Binary Interface) – **це специфікація, яка визначає кодування/декодування типів даних і стандарт для надання та виклику методів у смарт-контракті.**


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

ROUTER

- Router – це механізм у PyTeal для організації викликів методів у смарт-контрактах Algorand.
- Спрощує маршрутизацію транзакцій до відповідних підпрограм.
- Забезпечує сумісність із Application Binary Interface (ABI).

ОСНОВНІ ФУНКЦІЇ ROUTER

- **Реєстрація методів**: пов’язує методи з діями (OnCompleteAction).
- **Обробка** ABI-**викликів**: декодує вхідні дані транзакцій.
- **Підтримка дій**:
  - createOnly: для створення додатку.
  - callOnly: для звичайних викликів.
  - always: для всіх транзакцій.
  - never: метод не викликається.

ПРИКЛАД "FIRST ROUTER" – ЛІЧИЛЬНИК

Простий смарт-контракт для інкременту значення Count.

Router маршрутизує ABI-виклики до методів.

1. Router отримує виклик: транзакція надходить до Router.
2. Маршрутизація до методу: @router.method спрямовує виклик до increment.
3. Оновлення Count: збільшує значення Count на 1.

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

# Ініціалізація Router
router = Router(  # <--- 1. Router отримує виклик
    "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(),
    ),
)

# Метод для збільшення Count
@router.method  # <--- 2. Маршрутизація до методу
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.
Оновлення Count
    )
```

Транзакція --> Router --> increment() --> Оновлення Count

ПІДПРОГРАМИ У PYTEAL

01

Підпрограми дозволяють створювати модульний код у смарт-контрактах.

02

Використовуються з декоратором @Subroutine для визначення функцій.

03

Допомагають уникнути дублювання коду та спрощують логіку.

1. Декоратор @Subroutine: визначає підпрограму, яка повертає TealType.none (без значення).
2. Оновлення стану: підпрограма змінює глобальну змінну Count.
3. Інтеграція з Router: метод increment викликається через Router.
4. Виклик підпрограми: increment_count() викликається всередині методу.

```python
# Визначення підпрограми
@Subroutine(TealType.none)  # <--- 1. Декоратор для підпрограми
def increment_count():
    scratchCount = ScratchVar(TealType.uint64)
    return Seq(
        scratchCount.store(App.globalGet(Bytes("Count"))),
        App.globalPut(Bytes("Count"), scratchCount.load() + Int(1)),  # <--- 2.
Оновлення стану
    )

# Використання у Router
router = Router("my-first-router", BareCallActions(...))
@router.method  # <--- 3. Виклик підпрограми через Router
def increment():
    return Seq(
        Assert(Global.group_size() == Int(1)),
        increment_count(),  # <--- 4. Виклик підпрограми
    )
```

Транзакція --> Router --> increment() --> increment_count() --> Оновлення Count

ДОДАТКОВІ МАТЕРІАЛИ

- Pyteal for beginners (Відео курс). У цьому курсі ви дізнаєтесь, як розпочати створення смарт-контрактів Algorand із PyTeal, і детально зануритеся в кожну з операцій PyTeal, які знадобляться вам для написання складних і потужних смарт-контрактів.
- Algorand PyTeal Github репозиторій. У цьому репозиторії наявні приклади смарт-контрактів та смарт-підписів, написаних мовою PyTeal.
- Fundamentals of PyTeal (Курс). Цей курс на PyTeal містить вичерпний вступ до написання смарт-контрактів для блокчейну Algorand за допомогою PyTeal. Студенти вивчать такі ключові поняття, як маршрутизатори, вирази, типи даних AVM і ABI, поля транзакцій, атомарні передачі та підпрограми. Можливо отримати NFT сертифікат.