Початок роботи з PyTeal

Лекція 3

План

1. Типи даних в PyTeal
2. Арифметичні операції
3. Поля транзакцій і глобальні параметри
4. Scratch space
5. Вирази потоку управління

AVM типи даних в PyTeal

Є два основні типи данних:

- Integers
- Bytes

- Цілі числа насправді є uint64, що означає, що ми можемо використовувати лише позитивні цілі значення до 2^64. TealType.uint64, 64-бітове ціле число без знаку.

  Int(n) створює константу TealType.uint64, де n >= 0 і n < 2 ^ 64.

- Байти або байтовий зріз - це двійковий рядок. TealType.bytes, фрагмент байтів

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

Bytes

Байтовий фрагмент - це двійковий рядок. У PyTeal існує кілька способів кодування байтового фрагмента:

- **UTF-8:** байтові фрагменти можна створювати з рядків у кодуванні UTF-8. Приклад: `Bytes("hello world")`
- **Base16:** зрізи байтів можна створити з двійкового рядка в кодуванні base16 RFC 4648#section-8, наприклад. "0xA21212EF" або "A21212EF". Приклад: `Bytes("base16", "0xA21212EF")` або `Bytes("base16", "A21212EF") # "0x" is optional`
- **Base32:** Байтові фрагменти можуть бути створені з RFC 4648#section-6 двійкового рядка в кодуванні base32 із заповненням або без нього, наприклад. "7Z5PWO2C6LFNQFGHWKSK5H47IQP5OJW2M3HA2QPXTY3WTNP5NU2MHBW27M". Приклад: `Bytes("base64", "Zm9vYmE=")`

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

4

Перетворення

Перетворення значення на відповідне значення в іншому типі даних підтримується такими двома операторами:

- `Itob(n)`: генерує значення TealType.bytes із значення n типу TealType.uint64
- `Btoi(b)`: генерує значення TealType.uint64 зі значення b типу TealType.bytes

Ці операції не призначені для перетворення між зрозумілими людиною рядками та числами. Itob створює 8-байтове кодування цілого числа без знаку, а не зрозумілий людині рядка. Наприклад, Itob(Int(1)) створить рядок "x00x00x00x00x00x00x00x01", а не рядок "1".

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

Арифметичні операції

Арифметичний вираз — це вираз, результатом якого є значення TealType.uint64. У PyTeal арифметичні вирази включають цілі чи логічні оператори (булеві значення — це цілі числа 0 або 1).

Перевірка типів

У PyTeal реалізовано підсистему перевірки типів, яка відхиляє неправильно типізовані програми PyTeal під час створення програми PyTeal.

Наприклад,

```
cond = Txn.fee() < Txn.receiver()
```

Запуск цієї програми PyTeal у Python дає повідомлення про помилку:

```
TealTypeError: Type error: TealType.bytes while expected TealType.uint64
```

Це пояснюється тим, що Txn.receiver() має тип TealType.bytes, тоді як перевантажений < очікує операнди типу uint64 з обох сторін.

7

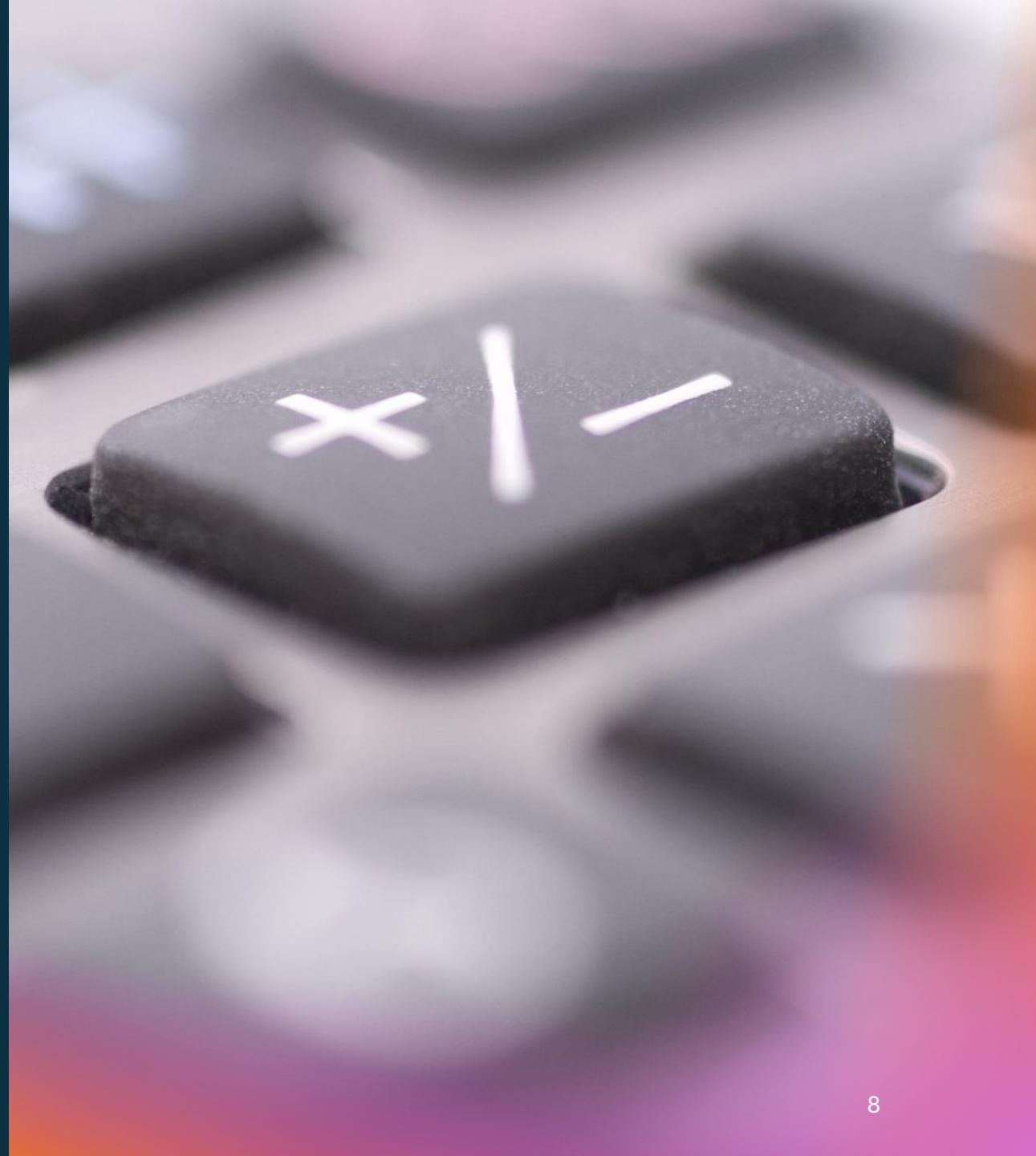
Перевантаження операторів

PyTeal перевантажує оператори арифметики та порівняння Python (+, -, *, /, >, <, >=, >=, ==), щоб користувачі PyTeal могли більш природно виражати логіку смарт-контракту в Python.

Приклади:

`Txn.fee() < Int(4000)` #перевіряє, що комісія поточної транзакції нижча за 4000 microAlgo.

`Gtxn.amount(0) == Gtxn.amount(1) * Int(2)` #перевіряє, що сума Algo, переданих у першій транзакції групової транзакції, вдвічі перевищує суму Algo, переданих у другій транзакції.



Арифметика байтового зрізу

Поля транзакцій і глобальні параметри

Смарт-контракти PyTeal можуть отримати доступ до властивостей поточної транзакції та стану блокчейну, коли вони запущені.

Поля транзакції. Загальні поля

Інформацію про поточну оцінювану транзакцію можна отримати за допомогою **Txn** об'єкта за допомогою виразів PyTeal, показаних нижче.

Оператор	Тип	Замітки
Txn.type()	TealType.bytes	Визначає тип транзакції. Якщо транзакція є платежем, то повернеться "pay".
Txn.type_enum()	TealType.uint64	Тип транзакції у вигляді цілого числа. Значення відповідає певному типу транзакції (наприклад, 1 для платежу).
Txn.sender()	TealType.bytes	Адреса відправника транзакції у вигляді 32-байтової адреси Algorand
Txn.fee()	TealType.uint64	Комісія за транзакцію у microAlgos
Txn.first_valid()	TealType.uint64	Перший раунд, в якому транзакція може бути дійсною
Txn.first_valid_time()	TealType.uint64	UNIX-мітка часу блоку перед Txn.first_valid(). Помилка, якщо від’ємне
Txn.last_valid()	TealType.uint64	
Txn.note()	TealType.bytes	Примітка транзакції в байтах
Txn.group_index()	TealType.uint64	Позиція цієї транзакції в групі транзакцій, починаючи з 0
Txn.tx_id()	TealType.bytes	Обчислений ID для цієї транзакції. Це 32-байтовий ідентифікатор транзакції.

Джерело: https://pyteal.readthedocs.io/en/latest/accessing_transaction_field.html

Приклад використання

Txn.sender()

```
from pyteal import *

program = Seq([
    Assert(Txn.sender() == Addr("YOUR_ADDRESS_HERE")),
    Approve()
])
```

Txn.group_index()

```
from pyteal import *

program = Seq([
    Assert(Txn.group_index() == Int(0)),
    Approve()
])
```

Txn.note()

```
from pyteal import *

program = Seq([
    Assert(Txn.note() == Bytes("example note")),
    Approve()
])
```

Txn.fee()

```
from pyteal import *

program = Seq([
    Assert(Txn.fee() < Int(1000)),
    Approve()
])
```

Поля транзакції. Виклик програми

Оператор	Тип	Замітки
Txn.application_id()	TealType.uint64	ID додатку, який викликається
Txn.on_completion()	TealType.uint64	Транзакція програми повинна вказувати дію, яка має бути виконана після виконання її approvalProgram або clearStateProgram. OnComplete
Txn.approval_program()	TealType.bytes	Повертає байтовий рядок з approval TEAL програмою для створення або оновлення додатку
Txn.global_num_uints()	TealType.uint64	Максимальна кількість глобальних змінних типу uint для додатку
Txn.global_num_byte_slices()	TealType.uint64	Максимальна кількість глобальних змінних типу byte slice для додатку
Txn.local_num_uints()	TealType.uint64	Кількість локальних змінних типу uint для додатку
Txn.local_num_byte_slices()	TealType.uint64	Кількість локальних змінних типу byte slice для додатку
Txn.accounts	TealType.bytes[]	Масив акаунтів, доступних додатку
Txn.assets	TealType.uint64[]	Масив активів, доступних додатку
Txn.applications	TealType.uint64[]	
Txn.clear_state_program()	TealType.bytes	Повертає байтовий рядок з TEAL програмою для очищення стану додатку
Txn.application_args	TealType.bytes[]	Масив аргументів виклику програми

Джерело: https://pyteal.readthedocs.io/en/latest/accessing_transaction_field.html

Приклад використання

Перевірка, що кількість глобальних типу Integer змінних дорівнює 5

```
from pyteal import *

program = Seq([
    Assert(Txn.global_num_uints() == Int(5)),
    Approve()
])
```

Логування програми очищення стану

```
from pyteal import *

program = Seq([
    Log(Txn.clear_state_program()),
    Approve()
])
```

Перевіряє, що перший аргумент додатку відповідає "arg1"

```
from pyteal import *

program = Seq([
    Assert(Txn.application_args[0] == Bytes("arg1")),
    Approve()
])
```

Поля транзакції. Передача активів

Оператор	Тип	Замітки
Txn.xfer_asset()	TealType.uint64	Використовується для отримання ID активу, який передається у транзакції типу Asset Transfer. Цей оператор допомагає ідентифікувати актив, що передається.
Txn.asset_amount()	TealType.uint64	Кількість активу, що передається у транзакції типу Asset Transfer. Значення представляє кількість одиниць активу у вигляді цілого числа (TealType.uint64).
Txn.asset_sender()	TealType.bytes	Адреса акаунта, який ініціює передачу активу.
Txn.asset_receiver()	TealType.bytes	Адреса акаунта, який отримує актив у транзакції типу Asset Transfer. Це поле дозволяє визначити, кому саме передається актив.
Txn.asset_close_to()	TealType.bytes	Укажіть це поле, щоб видалити активи з облікового запису відправника та зменшити мінімальний баланс облікового запису (тобто відмовитися від активу).

Інші поля транзакції при роботі з активами:
Конфігурація активів: використовуються для керування створенням і налаштуванням активів у блокчейні Algorand.
Заморожування активів: використовуються для управління статусом заморожування активів на блокчейні Algorand.

Джерело: https://pyteal.readthedocs.io/en/latest/accessing_transaction_field.html

Типи транзакцій

Значення Txn.type_enum() можна перевірити за допомогою переліку TxnType:

Значення	Числове значення	Тип string	Опис
TxnType.Unknown	0	unknown	невідомий тип, недійсний
TxnType.Payment	1	pay	платіжна транзакція
TxnType.KeyRegistration	2	keyreg	
TxnType.AssetConfig	3	acfg	
TxnType.AssetTransfer	4	axfer	транзакція передачі активів
TxnType.AssetFreeze	5	afrz	транзакція заморожування активів
TxnType.ApplicationCall	6	appl	Транзакція виклику додатку

Приклад:
Txn.type_enum() == Int(1)

Scratch space

Scratch Space у PyTeal є **тимчасовим місцем** для збереження значень для подальшого використання у програмі. Це місце зберігання є тимчасовим, оскільки будь-які зміни в ньому **не зберігаються після завершення** поточної транзакції.

Scratch Space у PyTeal дозволяє зберігати проміжні значення під час виконання смарт-контрактів.

Scratch Space складається з 256 слотів, кожен з яких здатний зберігати одне значення типу Int або Byte slices. Коли використовується клас `ScratchVar` для роботи зі Scratch Space, слот автоматично призначається кожній змінній.

ScratchVar: Запис і Читання з Scratch Space

1. **Створення об'єкта ScratchVar:** створити об'єкт ScratchVar. Передати тип значень, які будуть зберігатися. Це може бути `TealType.uint64` для цілих чисел або `TealType.bytes` для байтових масивів.

   ```
   myVar = ScratchVar(TealType.uint64)
   ```

2. **Зберігання значення в ScratchVar:** для запису значення використовується метод `store`. Він приймає значення, яке потрібно зберегти у `ScratchVar`.

   ```
   myVar.store(Int(5))
   ```

3. **Читання значення з ScratchVar:** для читання значення використовується метод `load`. Він повертає значення, збережене в `ScratchVar`.

Можливо також вручну вказати, якому слоту повинен бути присвоєний ScratchVar у TEAL коді. Якщо ідентифікатор слота не вказаний, компілятор призначить його будь-якому доступному слоту.

Приклад використання ScratchVar

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

У цьому прикладі ми створюємо дві змінні ScratchVar: одну для зберігання цілого числа, іншу для зберігання байтового масиву, при цьому друга змінна призначена конкретному слоту (слот #4).

Програма PyTeal – це вираз PyTeal, що складається з інших виразів PyTeal.

Неможна включати власні вирази Python у дерево виразів PyTeal.

```
def approval_program():
        program = Return(Int(1))
        return compileTeal(program, Mode.Application, version=5)
```

У наведеному прикладі вся програма це Return(Int(1)). Тут Return — це вираз PyTeal, який приймає у якості аргументу інший вираз PyTeal. Аргументом, який ми йому передаємо, є вираз PyTeal Int(1).

Наприклад, якщо замість передачі Int(1) ми створимо програму Return(1), PyTeal видасть помилку, повідомляючи, що 1 не є допустимим виразом PyTeal.

*Якщо ви коли-небудь побачите будь-яку помилку Python, наприклад `...has no attribute 'type_of'`, PyTeal, ймовірно, намагається повідомити вам, що ви включили щось, що не є допустимим виразом PyTeal.

Джерело: https://developer.algorand.org/docs/get-details/dapps/writing-contracts/pyteal/

Вирази потоку управління

Вихід із програми: Approve та Reject

Вирази *Approve* і *Reject* призводять до негайного виходу з програми. Якщо використовується Approve, то виконання позначається як успішне, а якщо використовується Reject, то виконання позначається як неуспішне.

TEAL Version 4+	Еквівалентний вираз
Approve()	Return(Int(1))
Reject()	Return(Int(0))

Ці вирази також працюють усередині підпрограм. Коли вони використовуються всередині підпрограм, вони також призводять до негайного виходу з програми, на відміну від Return(...), який просто повертається з підпрограми.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Об'єднання виразів у ланцюжок: Seq

Вираз Seq можна використовувати для створення послідовності з кількох виразів. Аргументи - це вирази, які потрібно включити в послідовність, або у вигляді змінної кількості аргументів, або у вигляді єдиного списку.

Приклад:

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

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Об'єднання виразів у ланцюжок: Seq

Особливості Seq:

- Вираз **Seq** приймає значення останнього виразу в послідовності.
- Усі вирази в **Seq**, окрім останнього, не повинні повертати жодних значень (наприклад, повинні оцінюватися як TealType.none). Це обмеження існує тому, що проміжні значення не повинні додавати нічого до TEAL стека.

Невірний вираз Seq:

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

Цей вираз є неправильним, тому що Txn.sender() повертає адресу відправника транзакції і додає це значення до стека TEAL, але це значення не використовується перед тим, як виконується Return(Int(1)). У результаті в стеці TEAL залишається значення, яке не використовується, що спричиняє помилку.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Об'єднання виразів у ланцюжок: Seq

Якщо вам необхідно включити операцію, що повертає значення, до більш ранньої частини послідовності, ви можете обернути це значення у вираз Pop, щоб відкинути його.

Правильна послідовність виглядає так:

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

*Pop використовується для вилучення значення зі стека

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Просте розгалуження: If

У PyTeal умовні оператори реалізовані через вирази **If**. Ці вирази дозволяють виконувати різні частини коду в залежності від результату логічного тесту.

Вираз `If` має наступний формат:

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

Тут:

- `test-expr` завжди оцінюється і повинен мати тип `TealType.uint64`.
- Якщо результат `test-expr` більший за `0`, то виконується `then-expr`.
- Якщо результат `test-expr` дорівнює `0`, то виконується `else-expr`.

Зверніть увагу, що `then-expr` та `else-expr` повинні мати однаковий тип (наприклад, обидва повинні бути `TealType.uint64`).

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Просте розгалуження: If

Існує альтернативний спосіб запису виразу **If**, що полегшує читання складних умовних операторів:

```
If(test-expr)
    .Then(then-expr)
    .ElseIf(test-expr)
    .Then(then-expr)
    .Else(else-expr)
```

Цей формат дозволяє легко читати та розуміти складні умовні логіки, розбиваючи їх на кілька рядків. Він також надає можливість додати кілька умов **ElseIf**, що робить код більш організованим і читабельним.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Перевірка умов у PyTeal: Assert

Assert використовується для забезпечення виконання певних умов перед продовженням програми.

Синтаксис:

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

Як працює Assert?

- test-expr завжди оцінюється.
- test-expr повинен мати тип TealType.uint64.
- Якщо test-expr результат більше 0, програма продовжує роботу.
- Якщо test-expr результат дорівнює 0, програма завершується з помилкою.
- У випадку більш ніж однієї умови слід використовувати And().

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Перевірка умов у PyTeal: Assert

Приклад:

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

У цьому прикладі програма негайно завершиться з помилкою, якщо тип транзакції не є платежем.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Ланцюжок Тестів у PyTeal: Cond

Cond — це вираз у PyTeal, який створює ланцюжок тестів для вибору виразу-результату.

Синтаксис:

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

Як працює Cond?

- Кожен `test-expr` оцінюється по порядку, і відповідне тіло виконується для тесту, який повертає дійсне значення.
- Якщо `test-expr` повертає `0`, відповідний `body` ігнорується, і оцінюється наступний `test-expr`.
- Як тільки `test-expr` повертає істинне значення (> 0), відповідний `body` оцінюється для отримання значення цього виразу **Cond**.
- Якщо жоден `test-expr` не повертає істинне значення, вираз **Cond** оцінюється як `err` (TEAL opcode), що викликає аварійне завершення програми.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Ланцюжок Тестів у PyTeal: Cond

Типи даних у Cond

- Кожен `test-expr` повинен мати тип `TealType.uint64`.
- `body` може мати тип `TealType.uint64` або `TealType.bytes`.
- Усі `body` повинні мати однаковий тип даних. Інакше виникає помилка `TealTypeError`.

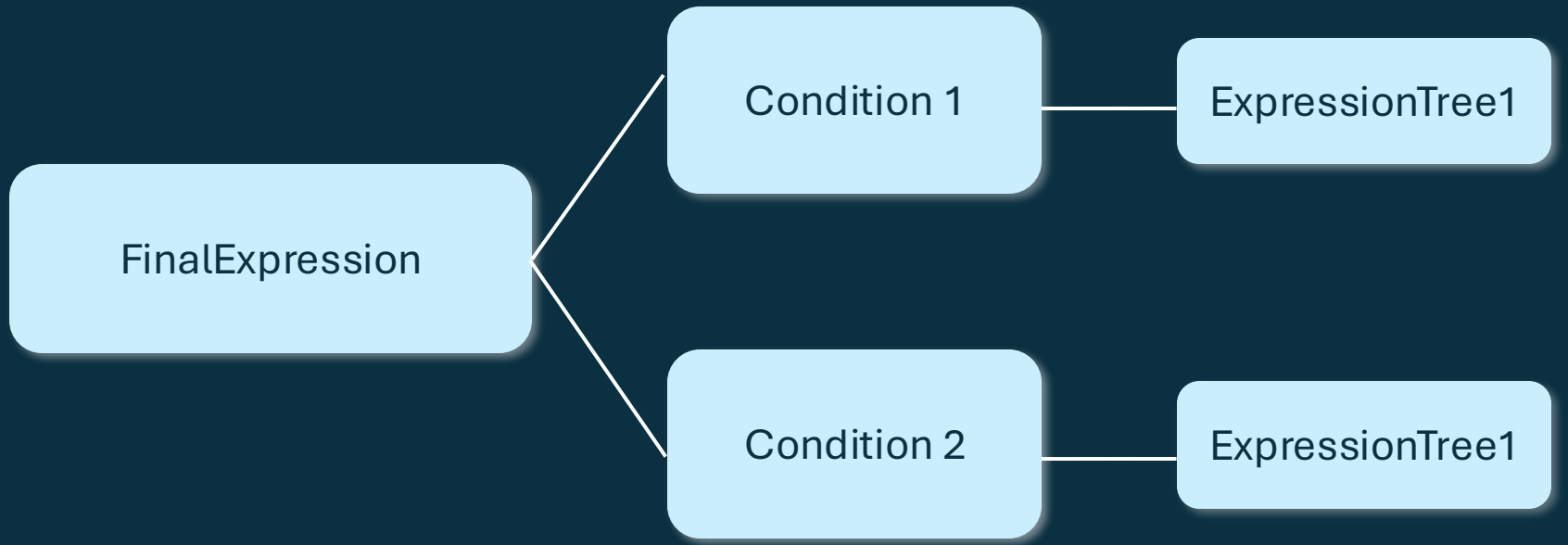
Приклад використання Cond:

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

Цей код PyTeal розгалужується в залежності від розміру групи атомарних транзакцій.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Потік управління деревом виразів PyTeal




```
return Cond(
    [Int(1) == Int(0), tree1],
    [Int(1) == Int(1), tree2],
)
```

Цикли у PyTeal: While

While дозволяє створювати прості цикли у PyTeal.

Синтаксис:

```
While(loop-condition).Do(loop-body)
```

- loop-condition — вираз, який повинен оцінюватися до TealType.uint64.
- loop-body — вираз, який повинен оцінюватися до TealType.none.
- loop-body виконується доти, доки loop-condition повертає істинне значення (> 0).

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Цикли у PyTeal: While

Приклад використання While

Розглянемо код, який використовує ScratchVar для ітерації чере з кожну транзакцію в поточній групі та підсумовування всіх їх ко місій:

```
totalFees = ScratchVar(TealType.uint64)  #змінна
  для зберігання загальної суми комісій
i = ScratchVar(TealType.uint64)  #змінна-лічильник для ітерації


Seq([
    i.store(Int(0)),
    totalFees.store(Int(0)),
    While(i.load() < Global.group_size()).Do(
        totalFees.store(totalFees.load() + Gtxn[i.load()].fee()),
        i.store(i.load() + Int(1))
    )
])
```

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Цикли у PyTeal: For

Цикли типу For дозволяють виконувати серію дій повторювано, базуючись на початковому значенні, умові циклу та кроку ітерації.

Синтаксис:

```
For(loop-start, loop-condition, loop-step).Do(loop-body)
```

- **loop-start**: Початкова дія, яка виконується один раз перед початком циклу.
- **loop-condition**: Умова, яка перевіряється перед кожною ітерацією циклу. Якщо умова істинна (> 0), виконується тіло циклу.
- **loop-step**: Дія, яка виконується після кожної ітерації циклу.
- **loop-body**: Тіло циклу, яке виконується в кожній ітерації, якщо умова істинна.

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Цикли у PyTeal: For

Приклад використання For

```
totalFees = ScratchVar(TealType.uint64)
i = ScratchVar(TealType.uint64)

Seq([
    totalFees.store(Int(0)),
    For(i.store(Int(0)), i.load()
< Global.group_size(), i.store(i.load() + Int(1))).Do(
        totalFees.store(totalFees.load() + Gtxn[i.load()].fee())
    )
])
```

Наведений код використовує ScratchVar для повторення кожної транзакції в поточній групі та підсумовування всіх їхніх комісій. Код тут функціонально еквівалентний прикладу циклу While, що був наведений раніше .

Джерело: https://pyteal.readthedocs.io/en/latest/control_structures.html?highlight=Cond#chaining-tests-cond

Вихід з циклів у PyTeal: **Continue і Break**

Вирази Continue і Break можна використовувати для виходу з циклів While і For.

- Оператор `Continue` вказує програмі пропустити залишок тіла циклу та перейти до наступної ітерації. Цикл продовжується до тих пір, поки умова циклу залишається істинною.
- Оператор Break вказує програмі повністю вийти з поточного циклу. Цикл припиняється незалежно від того, чи умова циклу залишається істинною.

38

Continue. Приклад використання

Розглянемо приклад, де ми ітеруємо через кожну транзакцію в поточній групі та рахуємо кількість платіжних транзакцій, використовуючи оператор Continue.

```
numPayments = ScratchVar(TealType.uint64)
i = ScratchVar(TealType.uint64)

Seq([
    numPayments.store(Int(0)),
    For(i.store(Int(0)), i.load() < Global.group_size(), i.store(i.load() + Int(1))).Do(
        If(Gtxn[i.load()].type_enum() != TxnType.Payment)
        .Then(Continue()),
        numPayments.store(numPayments.load() + Int(1))
    )
])
```

У цьому прикладі, якщо тип транзакції не є платіжним (TxnType.Payment), оператор Continue пропускає залишок тіла циклу та переходить до наступної ітерації. Інакше, лічильник платіжних транзакцій збільшується на одиницю.

Break. Приклад використання

Розглянемо приклад, де ми шукаємо індекс першої платіжної транзакції в поточній групі, використовуючи оператор Break.

```
firstPaymentIndex = ScratchVar(TealType.uint64)
i = ScratchVar(TealType.uint64)

Seq([
    # store a default value in case no payment transactions are found
    firstPaymentIndex.store(Global.group_size()),
    For(i.store(Int(0)), i.load() < Global.group_size(), i.store(i.load() + Int(1))).Do(
        If(Gtxn[i.load()].type_enum() == TxnType.Payment)
        .Then(
            firstPaymentIndex.store(i.load()),
            Break()
        )
    ),
    # assert that a payment was found
    Assert(firstPaymentIndex.load() < Global.group_size())
])
```

У цьому прикладі, якщо тип транзакції є платіжним, ми зберігаємо індекс цієї транзакції та використовуємо оператор Break для виходу з циклу. Це дозволяє нам знайти першу платіжну транзакцію та уникнути непотрібних ітерацій.

Компіляція

Функція `compileTeal` дозволяє скомпілювати смарт-контракт на PyTeal у TEAL.

Приклад:

```
# Компілируємо програми у TEAL
compiled_approval = compileTeal(approval, mode=Mode.Application, version=3)

compiled_clear = compileTeal(clear, mode=Mode.Application, version=3)
```

- Перший аргумент представляє собою програму на PyTeal, яка виконує певну логіку смарт-контракту. Зазвичай, це approval_program().
- Другий аргумент вказує режим компіляції. Це може бути Mode.Application — смарт-контракт буде використовуватися як додаток на блокчейні Algorand, або Mode.Signature, який використовується для написання логіки підписання транзакцій.
- Третій аргумент вказує версію TEAL, яка буде використовуватися для компіляції.

Результатом виклику методу compileTeal є рядок, що представляє скомпільований код TEAL.

Компіляція коду та запис у файл

```python
if __name__ == "__main__":
    approval_program = approval()
    clear_program = clear()
    # This is where the PyTeal code is compiled to TEAL
    approval_teal = compileTeal(approval_program, mode=Mode.Application, version=6)
    clear_state_teal = compileTeal(clear_program, mode=Mode.Application, version=6)

    with open("approval.teal", "w") as f:
        f.write(approval_teal)
        f.close()

    with open("clear.teal", "w") as f:
        f.write(clear_state_teal)
        f.close()
```

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.