# Основи Ethereum

Запорізький національний університет

2024

# Основи Ethereum

## Відмінності між Ethereum та Bitcoin

- **Мета:**
  - **Bitcoin:** Цифрова валюта для передачі вартості.
  - **Ethereum:** Платформа для створення децентралізованих додатків та смарт-контрактів.
- **Функціональність:**
  - **Bitcoin:** Обмежена функціональність скриптів.
  - **Ethereum:** Повноцінна Turing-повна мова для створення складних смарт-контрактів.
- **Час транзакцій:**
  - **Bitcoin:** Більше часу для підтвердження транзакцій.
  - **Ethereum:** Швидший час підтвердження блоків.

# Відмінності між Ethereum та Bitcoin (продовження)

- **Випуск монет:**
  - **Bitcoin:** Фіксована кількість 21 мільйон BTC.
  - **Ethereum:** Без фіксованої максимальної кількості ETH, з можливістю емітентів регулювати випуск.
- **Механізми консенсусу:**
  - **Bitcoin:** Використовує Proof of Work (PoW).
  - **Ethereum:** Переходить до Proof of Stake (PoS) з оновленням Ethereum 2.0.

# Основні компоненти Ethereum

## Блоки

- **Одиниці зберігання даних у блокчейні.**
- **Містять транзакції та інші метадані.**
- **Кожен блок містить посилання на попередній блок, утворюючи ланцюг.**

## Транзакції

- **Переміщення ETH або виклик функцій смарт-контрактів.**
- **Включають інформацію про відправника, отримувача, суму та дані.**
- **Потребують підпису приватним ключем відправника.**

## Адреси

- **Унікальні ідентифікатори для користувачів та контрактів.**
- **Починаються з "0x" і мають 40 шістнадцяткових символів.**
- **Використовуються для відправки та отримання ETH та токенів.**

# Гаманці

- **Засоби зберігання приватних ключів.**
- **Можуть бути гарячими (підключеними до Інтернету) або холодними (офлайн).**
- **Підтримують управління адресами та взаємодію з блокчейном.**

# Приватні та публічні ключі

- **Приватні ключі:**
  - **Забезпечують доступ до гаманців та авторизацію транзакцій.**
  - **Необхідно зберігати їх у безпечному місці.**
- **Публічні ключі:**
  - **Використовуються для створення адрес.**
  - **Можуть бути надіслані іншим користувачам для отримання ETH або взаємодії зі смарт-контрактами.**

# Gas та Витрати

## Що таке Gas?

- **Gas:** Вимірювання обчислювальної роботи, необхідної для виконання операцій в Ethereum.
- **Функція:** Запобігає спамінгу мережі та забезпечує оплату за ресурси, які використовуються під час виконання транзакцій та смарт-контрактів.

## Як працює Gas?

- **Кожна операція має ціну в Gas:**
  - **Наприклад:** Просте переведення ETH коштує менше Gas, ніж складні обчислення в смарт-контракті.
- **Вартість Gas визначається за допомогою параметра "Gas Price":**
  - **Вимірюється в Gwei (1 ETH = 1,000,000,000 Gwei).**
- **Загальна вартість транзакції:** Gas Limit × Gas Price.

# Gas Limit та Gas Price

- **Gas Limit:**
  - **Максимальна кількість Gas, яку користувач готовий витратити на транзакцію.**
  - **Захищає від нескінченного виконання контрактів.**
- **Gas Price:**
  - **Ціна за одиницю Gas, визначена користувачем.**
  - **Впливає на пріоритет транзакції в мережі.**

# Важливість оптимізації Gas

- **Зменшення витрат на транзакції:**
  - **Ефективний код смарт-контрактів споживає менше Gas.**
- **Швидкість підтвердження транзакцій:**
  - **Вища Gas Price може прискорити підтвердження транзакції.**

# Смарт-контракти

## Що таке смарт-контракт?

- **Автоматизовані контракти, які виконуються при дотриманні певних умов.**
- **Децентралізовані, непідробні та автоматично виконувані.**
- **Застосовуються у фінансах, управлінні активами, геймінгу та інших сферах.**

## Ключові характеристики смарт-контрактів

- **Децентралізованість:** Не залежать від центрального контролю.
- **Непідробність:** Код контракту незмінний після розгортання.
- **Автоматичне виконання:** Виконуються автоматично при задоволенні умов.

## Приклади використання смарт-контрактів

- **Фінансові послуги (DeFi):** Позики, кредитування, децентралізовані біржі.
- **Управління активами:** Токенізація нерухомості, цифрового мистецтва.
- **Геймінг:** Створення та торгівля внутрішньоігровими активами.
- **Логістика та відстеження товарів:** Автоматизація процесів та прозорість ланцюжка постачань.

# Мова програмування Solidity

## Основні концепції

- **Контракт:** Основна одиниця коду в Solidity, подібна до класу в інших мовах програмування.
- **Функції:** Блоки коду, які виконують певні дії.
- **Змінні:** Зберігають стан контракту.

## Інструменти для розробки

- **Remix IDE:** Онлайн середовище для написання, компіляції та тестування контрактів.
- **Truffle:** Фреймворк для розробки, тестування та розгортання смарт-контрактів.
- **Hardhat:** Інструмент для локальної розробки та тестування контрактів.

# Solidity: Основи та Приклади

## Оголошення Контракту

```
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 public storedData;

    function set(uint256 x) public {
        storedData = x;
    }

    function get() public view returns (uint256) {
        return storedData;
    }
}
```

- **Пояснення:**
  - **pragma solidity ^0.8.0;** — Визначає версію компілятора Solidity.
  - **contract SimpleStorage** — Оголошення контракту з іменем `SimpleStorage`.
  - **uint256 public storedData;** — Публічна змінна для зберігання числа.
  - **function set(uint256 x) public** — Функція для встановлення значення.
  - **function get() public view returns (uint256)** — Функція для отримання значення.

# Синтаксис та структури контролю

- **Умовні оператори та цикли:**

```
pragma solidity ^0.8.0;

contract ControlStructures {
    uint256 public count;

    function incrementIf(uint256 x) public {
        if(x > 10) {
            count += x;
        } else {
            count += 1;
        }
    }

    function loopExample(uint256 times) public {
        for(uint256 i = 0; i < times; i++) {
            count += i;
        }
    }
}
```

- **Пояснення:**
- **if-else:** Виконує різні дії залежно від умови.
- **for цикл:** Повторює блок коду певну кількість разів.

# Спадкування та Модифікатори

```
pragma solidity ^0.8.0;

contract Owned {
    address public owner;
    constructor() {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner, "Not owner");
        _;
    }
}

contract MyContract is Owned {
    uint256 public data;
    function setData(uint256 _data) public onlyOwner {
        data = _data;
    }
}
```

- **Пояснення:**
  - **contract Owned** — Базовий контракт для управління власником.
  - **modifier onlyOwner** — Модифікатор, який дозволяє виконання функції тільки власнику.
  - **contract MyContract is Owned** — Контракт, що наслідує від `Owned`.
  - **function setData(uint256 _data) public onlyOwner** — Функція, доступна тільки власнику.

# Подія (Event) у Solidity

```
pragma solidity ^0.8.0;

contract EventExample {
    event DataStored(uint256 indexed data);

    uint256 public storedData;

    function set(uint256 x) public {
        storedData = x;
        emit DataStored(x);
    }
}
```

- **Пояснення:**
  - **event DataStored(uint256 indexed data);** — Оголошення події `DataStored`.
  - **emit DataStored(x);** — Виклик події після збереження даних.
  - **indexed:** Дозволяє фільтрацію подій за цим параметром.

# Маппінги та Структури

```
pragma solidity ^0.8.0;

contract MappingExample {
    struct Person {
        string name;
        uint256 age;
    }

    mapping(address => Person) public people;

    function addPerson(string memory _name, uint256 _age) public {
        people[msg.sender] = Person(_name, _age);
    }

    function getPerson(address _addr) public view returns (string memory, uint256) {
        Person memory p = people[_addr];
        return (p.name, p.age);
    }
}
```

- **Пояснення:**
  - **struct Person** — Визначення структури `Person` з полями `name` та `age`.
  - **mapping(address => Person) public people;** — Маппінг адрес до осіб.
  - **function addPerson** — Додавання особи до маппінгу.
  - **function getPerson** — Отримання інформації про особу за адресою.

# Використання Бібліотек (OpenZeppelin)

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "MTK") {
        _mint(msg.sender, 1000000 * 10 ** decimals());
    }
}
```

- **Пояснення:**
  - **import "@openzeppelin/contracts/token/ERC20/ERC20.sol";** — Імпорт стандартного ERC20 контракту з OpenZeppelin.
  - **contract MyToken is ERC20** — Контракт токена, що наслідує від ERC20.
  - **constructor() ERC20("MyToken", "MTK")** — Конструктор, що задає назву та символ токена.
  - **_mint(msg.sender, 1000000 * 10** decimals());** — Створення початкової кількості токенів та їх присвоєння власнику.

# Ethereum Virtual Machine (EVM)

**Що таке EVM?**

- **Ethereum Virtual Machine (EVM):** Віртуальна машина для виконання смарт-контрактів у мережі Ethereum.
- **Функція:** Забезпечує виконання байт-коду контрактів у децентралізованому середовищі.

**Архітектура та функціонування EVM**

- Сумісність з різними мовами програмування через компілятори.
- Абстракція апаратного забезпечення для забезпечення консистентності виконання.
- Кожен вузол мережі має свою копію EVM для синхронізації стану блокчейну.

**Виконання смарт-контрактів в EVM**

- Контракти завантажуються у пам'ять EVM.
- Виконуються крок за кроком, змінюючи стан блокчейну відповідно до логіки контракту.
- Ізольоване середовище забезпечує безпеку та запобігає несанкціонованому доступу до системи хоста.

# Безпека та оптимізація

## Основні загрози

- **Reentrancy атаки:**
  - Використання рекурсивних викликів для витоку коштів зі смарт-контракту.
- **Переповнення чисел:**
  - Некоректне обчислення великих чисел, що може призвести до помилок у виконанні контракту.
- **Недостатнє тестування:**
  - Вразливості через відсутність належного тестування та аудиту коду.

# Методи захисту

- **Використання перевірених бібліотек:**
  - Наприклад, OpenZeppelin для реалізації стандартних функцій безпеки.
- **Аудит контрактів:**
  - Проведення незалежних перевірок коду сторонніми експертами.
- **Тестування та верифікація коду:**
  - Написання юніт-тестів та використання інструментів для статичного аналізу коду.

# Оптимізація газових витрат

- **Ефективний код:**
  - Використання оптимізованих алгоритмів та структур даних.
- **Мінімізація складних операцій:**
  - Зменшення кількості обчислень та зберігання даних у контракті.
- **Перегляд логіки контракту:**
  - Оптимізація логіки для зменшення потреби в газі.

# Токени та стандарти

## Огляд стандартів токенів

- **ERC-20:**
  - **Опис:** Стандарт для взаємозамінних токенів.
  - **Властивості:** Взаємодія з гаманцями та біржами, сумісність з інфраструктурою Ethereum.
- **ERC-721:**
  - **Опис:** Стандарт для незамінних токенів (NFT).
  - **Властивості:** Унікальність кожного токена, ідеально підходить для цифрового мистецтва та колекцій.
- **ERC-1155:**
  - **Опис:** Гібридний стандарт для взаємозамінних та незамінних токенів.
  - **Властивості:** Ефективність у витратах газу при масовій обробці токенів.

# Бібліотеки та інструменти

- **OpenZeppelin:**
  - **Опис:** Бібліотека для безпечної розробки смарт-контрактів.
  - **Функціональність:**
  - Реалізації стандартних контрактів (ERC-20, ERC-721).
  - Бібліотеки для управління ролями та доступом.
  - Безпечні математичні операції.
- **Використання бібліотек:**
  - **Шаблони контрактів:** Швидке створення контрактів з передбаченою функціональністю.
  - **Безпечні реалізації стандартів:** Мінімізація ризику помилок у власній реалізації стандартів.
  - **Розширення функціональності:** Використання модульної архітектури для додавання нових функцій.

# Використання токенів

- **Токенізація активів:**
  - **Опис:** Перетворення реальних активів (нерухомість, цінні папери) у токени на блокчейні.
  - **Переваги:** Ліквідність, доступність, прозорість.
- **ICO (Initial Coin Offering):**
  - **Опис:** Залучення капіталу через продаж токенів на ранніх етапах проекту.
  - **Переваги:** Швидкий доступ до фінансування, глобальний ринок інвесторів.
- **DeFi (Decentralized Finance):**
  - **Опис:** Децентралізовані фінансові послуги на базі Ethereum.
  - **Приклади:** Децентралізовані біржі, позики, страхування.

# Сучасна Sepolia Testnet

## Що таке Sepolia Testnet?

- **Sepolia:** Одна з офіційних тестових мереж Ethereum.
- **Функція:** Дозволяє розробникам тестувати смарт-контракти та додатки без ризику втрати реальних коштів.
- **Особливості:** Використовує сучасні оновлення та функціональності Ethereum, схожа на основну мережу.

## Переваги використання Sepolia

- **Сумісність з основною мережею:** Подібна структура та правила до mainnet, що забезпечує точніші тести.
- **Безкоштовні ресурси:** Отримання тестового ETH для розробки без витрат реальних коштів.
- **Підтримка оновлень:** Включає найновіші зміни та покращення, що дозволяє тестувати сучасні функції Ethereum.

# Як використовувати Sepolia Testnet

- **Підключення до гаманця:**
  - Вибір Sepolia як мережі у вашому гаманці (наприклад, MetaMask).
- **Отримання тестового ETH:**
  - Використання крана (faucet) для отримання безкоштовного тестового ETH.
  - Наприклад: Sepolia Faucet.
- **Розгортання смарт-контрактів:**
  - Використання інструментів розробки (Remix, Truffle, Hardhat) для деплою контрактів на Sepolia.
  - Тестування функціональності та безпеки контрактів у реалістичному середовищі.

# Інструменти для навчання

- **Платформи для розробки:**
  - **Remix IDE:** Онлайн середовище для написання, компіляції та тестування смарт-контрактів
  - **Truffle Suite:** Фреймворк для розробки, тестування та розгортання смарт-контрактів
  - **Hardhat:** Інструмент для локальної розробки та тестування контрактів
- **Тестові мережі:**
  - **Sepolia:** Сучасна тестова мережа з підтримкою нових функцій.
  - **Ropsten:** Тестова мережа з механізмом PoW.
  - **Rinkeby:** Тестова мережа з механізмом PoS.
  - **Kovan:** Тестова мережа з підтримкою Proof of Authority.