ブロックチェーン応用講座

Vol.2: スマートコントラクト

小林 聖弥 / Seiya Kobayashi

目次

目次

1. 2つのアカウント種別: EOA・スマートコントラクト

ブロックチェーン（Ethereum）:= ワールドコンピューター（地球規模の状態遷移マシーン）

- **ブロック（Block）**という単位で状態遷移（データの変動）を確定させていく
  - 1スロット（12秒）毎にブロック生成（コンセンサスアルゴリズム経由）
  - 2エポック（12秒 × 32スロット × 2エポック = 12.8分）毎に確定

- 各ブロックは複数の**トランザクション（Transaction・Tx）**から構築される
  - 1ブロックに約300トランザクション程度（約25TPS）が含まれる

- 各トランザクションは**アカウント（Account）**における状態遷移を表現する
  - **EOA（Externally Owned Account）**→ 一般的なアカウント
  - **スマートコントラクト（Smart Contract）**→ プログラム・ロジック

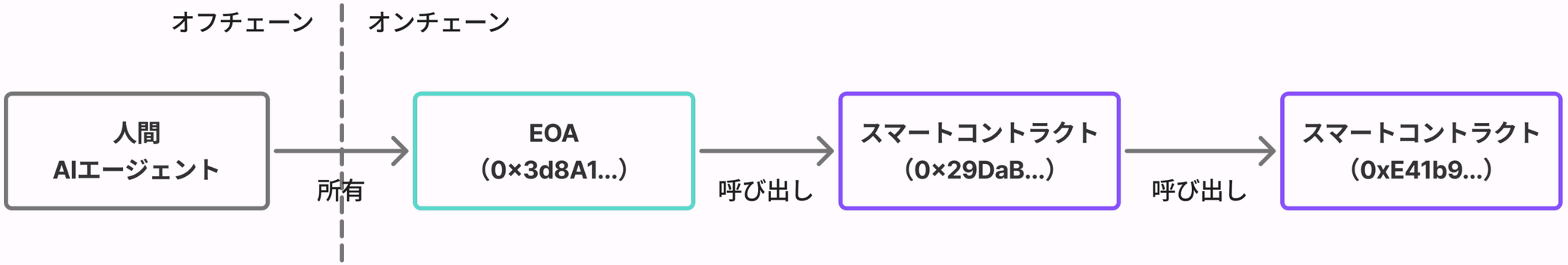
1. 2つのアカウント種別: EOA・スマートコントラクト

EOA

- **ユニークなアドレス**
  - **パブリックキー**から生成
  - 20 bytes（e.g., 0x3d8A1…）
- **オフチェーン**に所有権
  - 人間・AIエージェント等が所有
- トランザクションを**開始できる**

スマートコントラクト

- **ユニークなアドレス**
  - **デプロイヤーアドレス等**から生成
  - 20 bytes（e.g., 0x29DaB…）
- **オンチェーン**に所有権
  - EOAあるいは他のスマコンが所有
- トランザクションを**処理できる**
  - **開始はできない**



目次

2. アカウントの所有権

スマートコントラクトにおける所有権 → プログラムとして柔軟に設定できる

e.g.）デプロイヤーアドレスを所有者（owner）に設定し、所有者のみ特定のロジックを実行できる

EOAにおける所有権 := 有効なトランザクションを生成できる権利

- トランザクションの生成には**電子署名（Digital Signature）**が必要
  - 電子署名の検証が通る → 有効なトランザクションとしてネットワークに伝播される
  - 電子署名の検証が通らない → 無効なトランザクションとしてネットワークに弾かれる
- 電子署名アルゴリズムでは、**鍵ペア（Key pair）**と呼ばれるデータを利用する
  - **秘密鍵（Private Key）** → データに対する署名の**生成**に用いる
  - **公開鍵（Public Key）** → データに対する署名の**検証**に用いる

2. アカウントの所有権

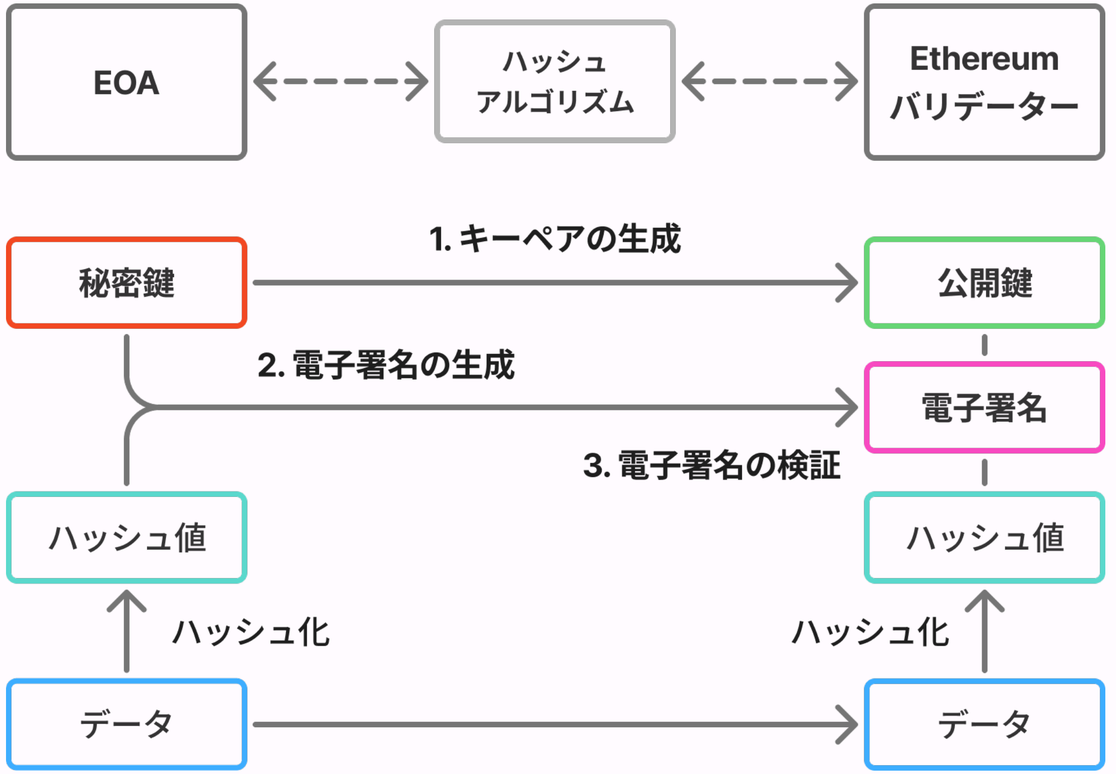
電子署名アルゴリズムの流れ（🤔 電子署名の目的？）

1. 鍵ペアの生成
    - ランダムな秘密鍵の生成 → 公開鍵の生成
    - ✅ **数学的不可逆性**

      → 公開鍵から秘密鍵は割り出せない

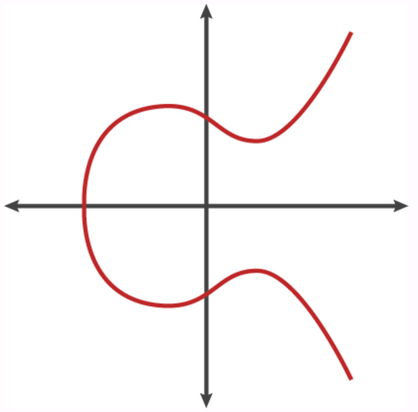
2. 電子署名の生成
    - **対象データと秘密鍵**を用いて電子署名を生成

3. 電子署名の検証
    - **対象データと公開鍵**を用いて電子署名を検証



2. アカウントの所有権

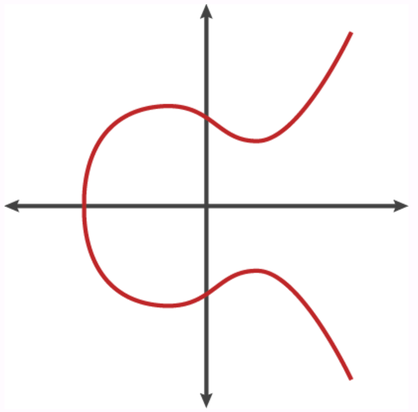
ECDSA（EVMチェーンで用いられている電子署名アルゴリズム）の仕組み



A Primer on Elliptic Curve Cryptography

0. 楕円曲線（elliptic curve）の選定
   - 楕円曲線 secp256k1 := $y^2 \equiv x^3 + 7 \mod p$
     - 楕円曲線の位数 $n := 2^{256} - 1.4 \times 10^{38}$
       - 楕円曲線上に存在する点 (x, y) の数 → 十分に大きな素数
     - 楕円曲線上に存在する全ての点から構成される群の生成元 G（固定値）
       - x 座標: 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
       - y 座標: 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
     - 有限体 $\mathbb{F}_p$ の位数 $p := 2^{256} - 2^{32} - 977$
       - 座標 (x, y) それぞれの取りうる最大値 → 十分に大きな素数（ただし $p > n$）

2. アカウントの所有権



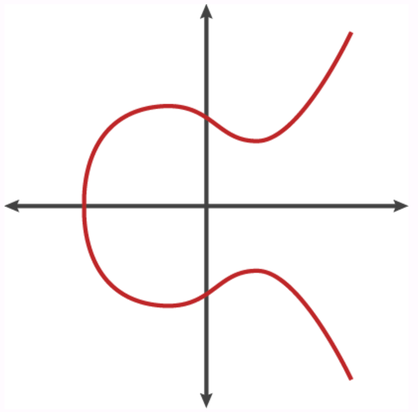
A Primer on Elliptic Curve Cryptography

ECDSA（EVMチェーンで用いられている電子署名アルゴリズム）の仕組み

1. 鍵ペア $\{sk, pk\}$ の生成
    - 秘密鍵 sk を楕円曲線の位数 n からサンプリング: $sk \xleftarrow{\$} [1, n-1]$
    - 秘密鍵 sk と楕円曲線上の基点 G から公開鍵 pk を算出: $pk = sk \cdot G$
        - G := 楕円曲線上に存在する全ての点から構成される群の生成元
            - 💡 G を足し合わせることで楕円曲線上の全ての点 $\{G, 2G, \ldots, nG\}$ が生成できる
            - 位数 n は素数であるため、部分群は存在しない（**ラグランジュの定理**）
        - pk := 楕円曲線上のいずれかの点（無限遠点を除く）
            - EOAアドレス := pk のハッシュ値（keccak-256）の末尾 20 bytes

2. アカウントの所有権

ECDSA（EVMチェーンで用いられている電子署名アルゴリズム）の仕組み



A Primer on Elliptic Curve Cryptography

2. 電子署名 $\{r, s\}$ の生成

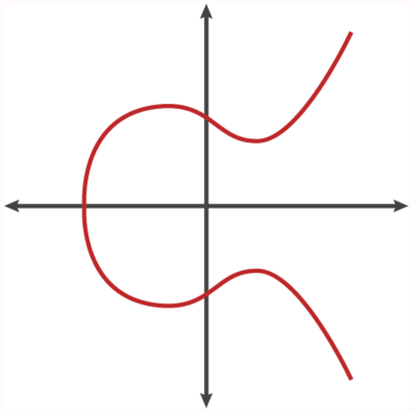
- トランザクション m のハッシュ値 h を算出: $h = keccak256(m)$

  → 🤔 なぜ元データそのものではなく、ハッシュ値に対して電子署名を行うのか？

- ナンス k を楕円曲線の位数 n からサンプリング: $k \xleftarrow{\$} [1, n-1]$
- r を算出: 楕円曲線上の点 $R = k \cdot G$ の x 座標
- s を算出: $s = k^{-1} \cdot (h + r \cdot sk) \mod n$
  - $k \cdot k^{-1} \equiv 1 \mod n$ となる k^{-1} は、**拡張ユークリッドの互除法**で効率的に算出できる
  - 💡 トランザクションハッシュ h と秘密鍵 sk を**数学的に紐付ける**

2. アカウントの所有権

ECDSA（EVMチェーンで用いられている電子署名アルゴリズム）の仕組み



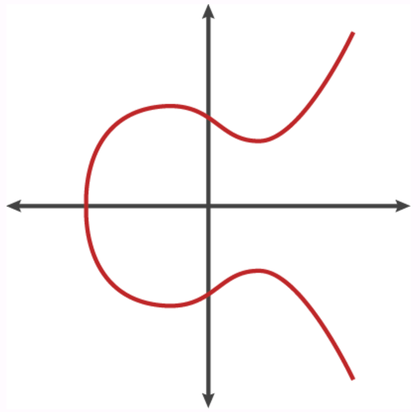
A Primer on Elliptic Curve Cryptography

3. 電子署名 $\{r, s\}$ の検証
   - r が楕円曲線の位数 n の範囲に収まっていることを検証
   - $\{u_1, u_2\}$ の算出: $u_1 = h \cdot s^{-1} \mod n$、$u_2 = r \cdot s^{-1} \mod n$
   - 楕円曲線上の点 P の算出: $P = u_1 \cdot G + u_2 \cdot pk$
   - 点 P の x 座標が r（点 R の x 座標）と等しければ検証成功、等しくなければ検証失敗
     - $P = h \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot sk \cdot G = (h + r \cdot sk) \cdot s^{-1} \cdot G = k \cdot G = R$
     - 💡 公開鍵 pk を用いて、電子署名生成時にランダムに選んだ点 R が復元されるかを検証する

       → 🤔 なぜ x 座標のみを確認すれば良いのか？

2. アカウントの所有権

ECDSA（EVMチェーンで用いられている電子署名アルゴリズム）の仕組み



A Primer on Elliptic Curve Cryptography

💡 公開鍵 pk の導出（EVMチェーンでは、**EOAは公開鍵 pk を直接送付しない**）

- 電子署名 $\{r, s, v\}$ の生成
  - v の追加: 点 R の y 座標が**偶数か奇数か**（🤔 なぜ？）を判別するための小さな自然数
- 電子署名 $\{r, s, v\}$ の検証 → EVMレベルで以下を行う **ecrecover** と呼ばれる関数が存在
  - 点 R の特定 → 座標 (x, y) の特定
    - 楕円曲線式 $y^2 \equiv r^3 + 7 \mod p$ から y を算出し、v を用いて y 座標を特定
      - ごく小さい確率で $n < r < p$ となるため、その場合は $x = r + n$ を代入する
  - 公開鍵 pk の復元
    - $pk = r^{-1} \cdot (s \cdot R - h \cdot G) = r^{-1} \cdot (s \cdot k - h) \cdot G = r^{-1} \cdot (h + r \cdot sk - h) \cdot G = sk \cdot G$

目次

3. アカウントの実装

EOA

- **アカウントの管理 = 秘密鍵の管理**
  → ✅ 秘密鍵を「安全に管理」すれば良い
  → 🤔 「安全な管理」とは？
- 秘密鍵の管理・秘密鍵による電子署名
  → **ウォレット**（e.g., MetaMask）
  → 自前コード（e.g., TypeScript）

https://metamask.io/

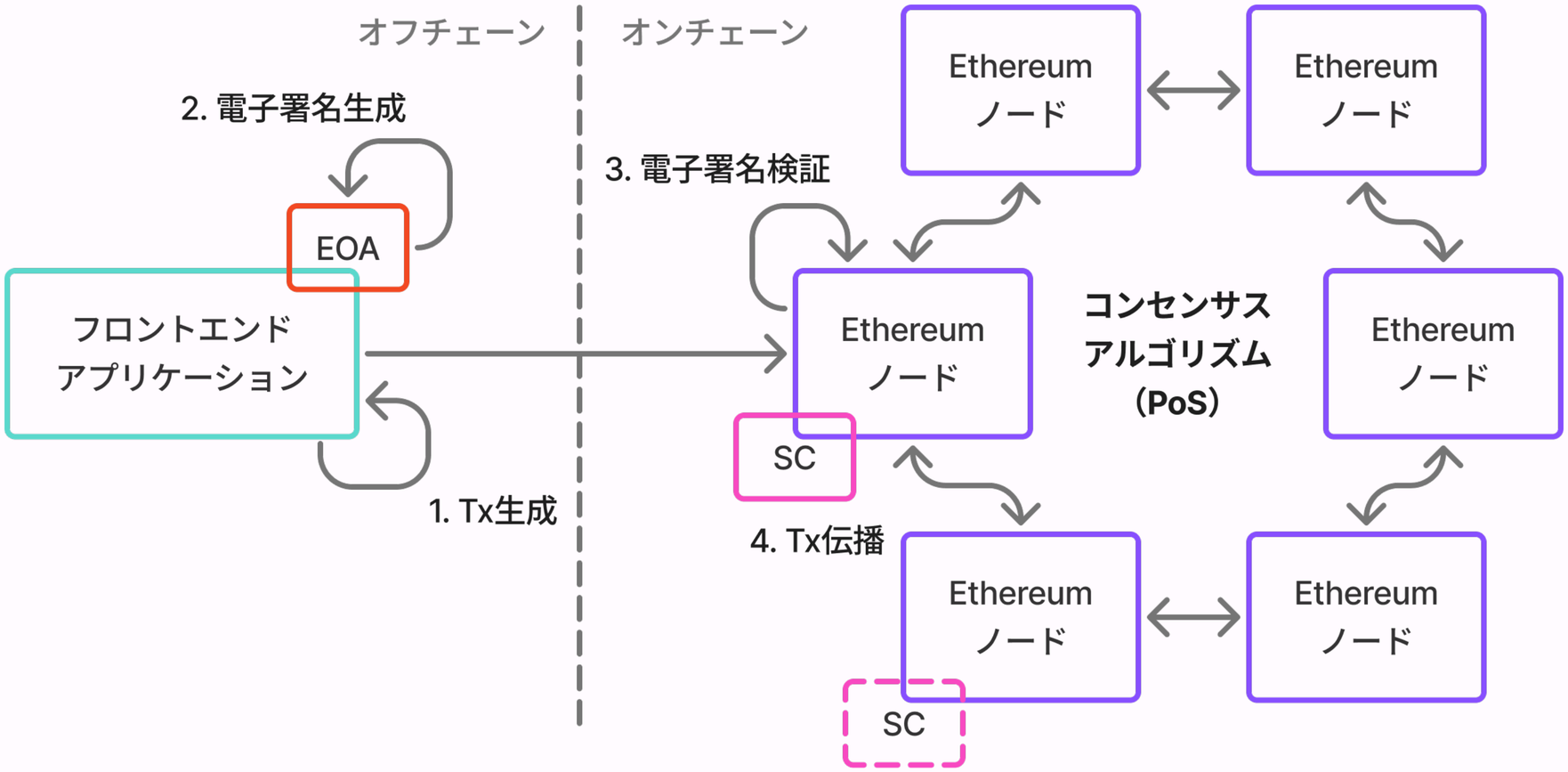
スマートコントラクト

- **EVM互換性のある言語**で実装
  → e.g.）**Solidity**・Vyper
- EOAがネットワークにデプロイする
  → **スマコンのデプロイも1つのTx**
  → EOAによる電子署名が必要

https://www.soliditylang.org/

3. アカウントの実装

ブロックチェーンアプリケーション（dApp）の概観



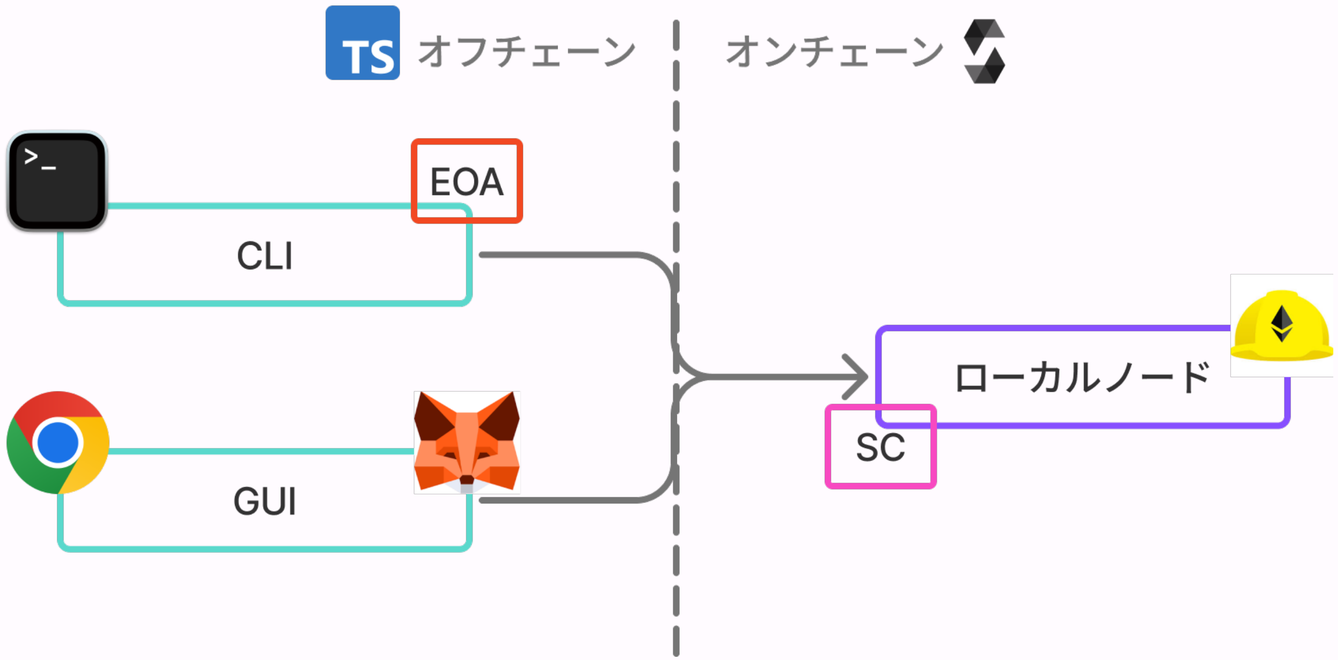
目次

4. ワークショップにおける開発環境・ツール

ワークショップで実装するdAppの概観

- スマートコントラクト開発
  → 環境: Hardhat
  → 実装: Solidity

- フロントエンドアプリケーション開発
  - CLI（Command Line Interface）
    → 環境: Terminal
    → 実装: TypeScript（viem）
  - GUI（Graphical User Interface）
    → 環境: Chrome + MetaMask
    → 実装: TypeScript（React・wagmi）



4. ワークショップにおける開発環境・ツール

開発環境を整える

- 各種ツールの設定

  - **git**: バージョン管理ソフトウェア
    - インストール: https://gitforwindows.org/

      → 参考記事: Qiita
    - GitHub連携: 上記の参考記事の最後のセクション「Gitの設定」を参考に設定

  - **GitHub**: gitをチームや複数人で扱うためのソフトウェア
    - リポジトリのクローン: Git Bashで以下（$は除く）を実行

      $ git clone https://github.com/Institution-for-a-Global-Society/blockchain-lecture-series.git/

4. ワークショップにおける開発環境・ツール

開発環境を整える

- 各種ツールの設定
    - **VS Code**: コードエディター（実際にコードを書くエディター）
        - git経由でクローンしたディレクトリ（ダウンロード場所は各自で設定）を開く
          → 参考記事: Zenn
        - Solidity言語用の拡張機能をダウンロード・有効化
          → https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity

目次

💻 ワークショップ#1: Solidityの基礎

./lectures/2/README.md へ移動

目次

💻 ワークショップ#2: スマートコントラクトを実装してみよう

🗣️ **スマートコントラクトの所有可能性（Ownability）と停止可能性（Pausability）について考える**