Solidity 入門

2022/11/17（木）5限 bcali MTG

B1 yum

目的

- 一般に「スマートコントラクト」と呼ばれるものが，どのように作られ、動いているのか実際に手を動かして作ってみることで理解する
  - あくまで流れだけを簡単に触ってみる
- Solidity を使ってクラウドファンディングコントラクトを作ってみる

今回使うツール・ライブラリなど

- **Hardhat と周辺ツール：**

  EVM互換スマートコントラクトの開発フレームワーク.

  Solidity と JavaScript で開発を進める.

- **Metamask**：Chrome 拡張機能版ウォレット

- **Docker & Visual Studio Code：**

  Dev Container でクールに開発環境をセットアップしよう！

事前準備でお願いしたこと

- VSCode と Docker，そして Dev Containers 拡張機能の準備
- Metamask のセットアップ
- SepoliaETH の入手

進め方の方針

- yum は**スライド**を使って前で説明します
- 事前配布の**レジュメ**に**スライド**や口頭で説明しきれない
  であろうことを書いてあります

⇒ 特に手を動かすときは,
スライドと**レジュメ**を**両方参照する**ことをおすすめします！

質問やご指摘があれば，随時ツッコミをお願いします.

1. スマートコントラクトとは？

旧来のスマートコントラクトの概念

- もともとの概念は, 1996年に Nick Szabo が提唱

  The Idea of Smart Contracts :
  https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html

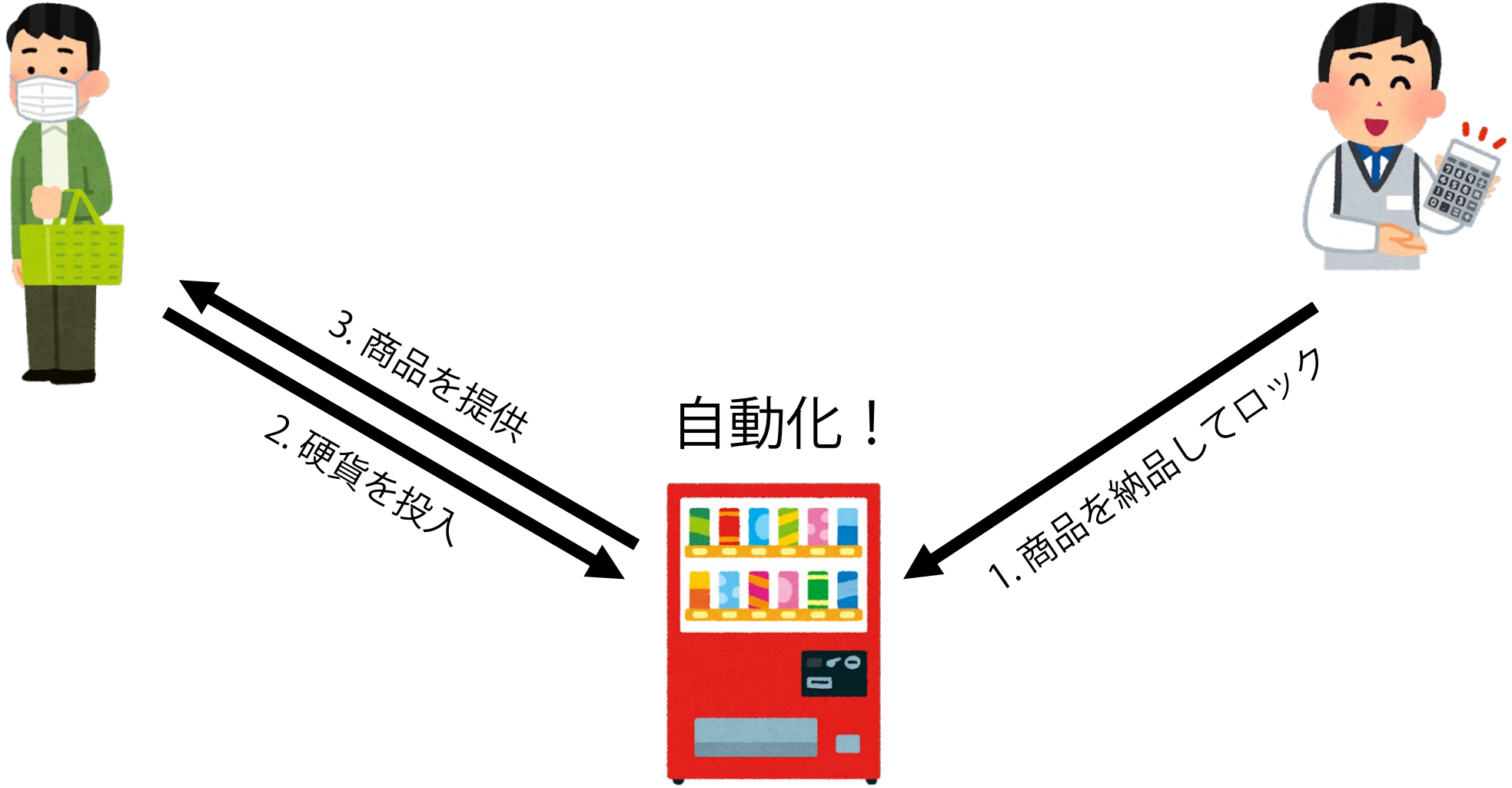
- 当事者が他の約束を実行する手順まで含んだ、デジタル形式で規定された一連の約束
- 第三者へのトラストレスなエスクロー取引を実装できる？

スマートコントラクト（オフライン）

料金を支払い

商品を引き渡し

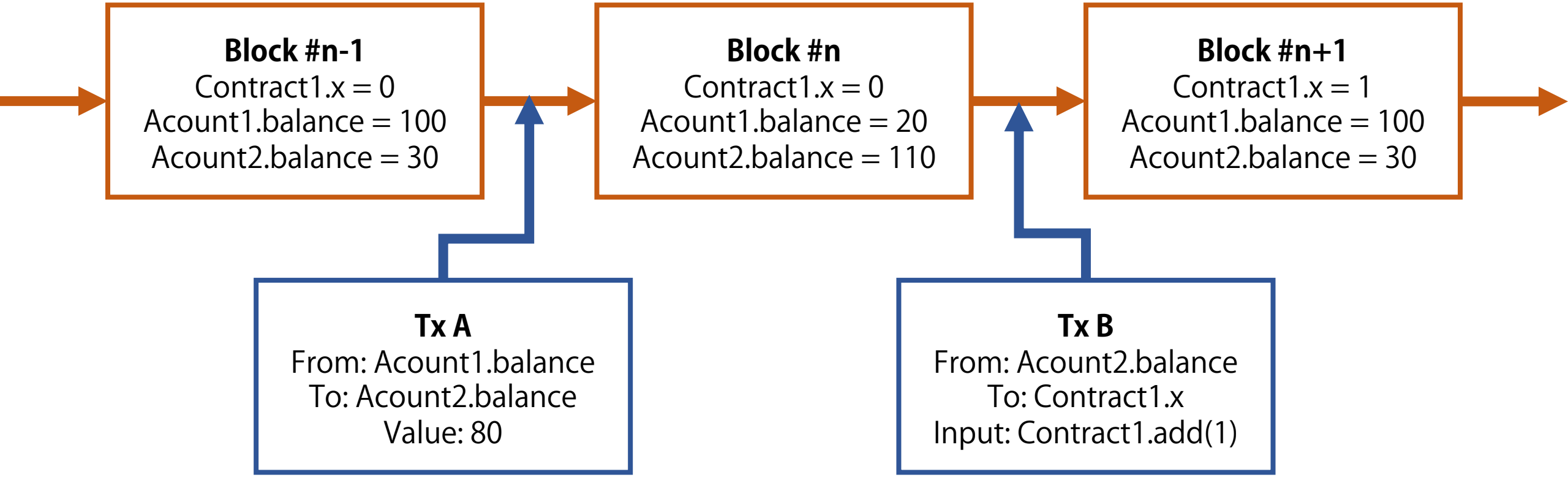
スマートコントラクト（オフライン）



Ethereum のスマートコントラクト

- Ethereum によって導入されたスマートコントラクトは
  ブロックチェーン上で動くプログラム
- Ethereum をステートマシンとして捉える（概念としてのEVM）
  1. あるステート（状態）がある
  2. 外部からあるイベントがエミットされる
  3. 該当ステートが更新され、それによって処理が切り替わる
- Tx によって関数呼び出しを行い、状態遷移をする
  - 処理はそれぞれのノード上で実行される

Ethereum のスマートコントラクト



※ Ethereum は UTXO 型ではなく Account 型を採用している

※ Gas については考慮していない　※ 以後，スマートコントラクトは Ethereum 上のものとする．

スマートコントラクトのライフサイクル

1. **実装**：Solidity 等のスマートコントラクト用言語で実装
2. **コンパイル**：EVM用のバイトコードにコンパイル
3. **ネットワークに展開**：0x0 宛にトランザクションを送信
4. **状態遷移**：EOA からのトランザクションによって関数実行
   - Ethereum の Tx は**アトミック性**を持つ

     i.e. 実行に失敗すると Tx 開始前にロールバックされる
   - コントラクトからはトランザクションを開始できない

2. Solidity 入門

Solidity とは？

- Ethereum・EVM互換チェーンで稼働する
  スマートコントラクトを記述するためのDSL
- Gavin Woodらを中心に開発
- オブジェクト指向な高級言語に似た形でコントラクトを記述できる
- 言語の仕様的にはチューリング完全である
  - 実際はEVM上で gas による制約を受ける

環境構築

せっかくなので，前回学んだ Docker と git を用いる

- **Dev Containers**：

  Docker コンテナ内の開発環境で快適に開発を行える

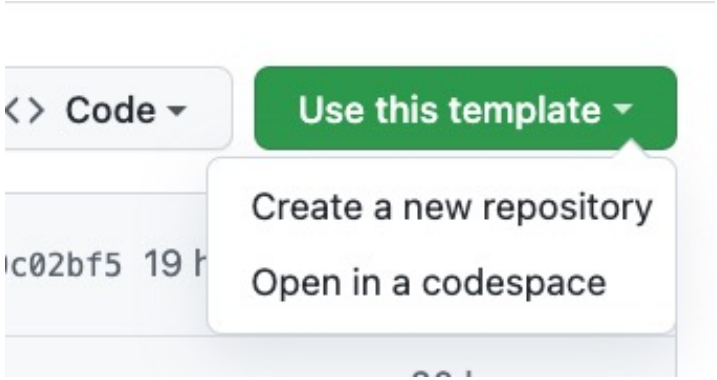
  ＝ 開発環境の**可搬性 UP**

手順1.　GitHub のテンプレートレリポジトリをクローンする

手順2.　クローンしたリポジトリを開発用コンテナ内で開く

環境構築 – リポジトリをクローン

1. テンプレートリポジトリにアクセス：
   https://github.com/ymsg19/solidity-handson
2. 「Use this template」から「Create a new repository」を選択
3. リポジトリ作成画面が出てくるので
   リポジトリを作成する
4. クローンしたいディレクトリに移動し
   git clone <url> を実行



環境構築 – 開発用コンテナを起動

1. クローンしたフォルダを VSCode で開く
2. Mac: ⌘ + Shift + P，Win: Ctrl + Shift + P でコマンドパレット
3. 入力欄に dev と入力し，Dev Containers: Open Folder in Container… を選択
4. エラーなくコンテナに接続できれば完了
5. ターミナルを起動（Ctrl + Shift + @）して yarn を実行

3. はじめてのコントラクト作成

はじめてのコントラクト作成

Hello, World コントラクトを作成する.

contracts/HelloWorld.sol を作成し以下のように編集

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

contract HelloWorld {
    function sayHello() external pure returns (string memory) {
        return "Hello, World!";
    }
}
```

はじめてのコントラクト作成

- コンパイル

```
$ yarn hardhat compile
Generating typings for: x artifacts in dir: typechain-types for target: ethers-v5
Successfully generated 1 typings!
Compiled 1 Solidity file successfully
Done in xx.
```

はじめてのコントラクト作成

- Hardhat のローカルテストネットにデプロイ & 実行

```
$ yarn hardhat console
Welcome to Node.js v18.12.1.
Type ".help" for more information.
> const Contract = await ethers.getContractFactory("HelloWorld")
> const contract = await Contract.deploy()
> contract.address
'0x5FbDB2315678afecb367f032d93F642f64180aa3'
> await contract.sayHello()
'Hello, World!'
```

ここまでのまとめ

- キーワード Contract でコントラクトを書くことができる
  - 一つのファイルに複数個のコントラクトを書くことも可能
  - コントラクトはオブジェクト指向プログラミングのようなカプセル化や再利用をすることもできる
- キーワード function でコントラクトを書くことができる
  - キーワード return で値を返すことができる

4. 文法の簡単な説明

Pragma

- コンパイラのバージョンを指定する
- 基本的にファイルの先頭行に記述する
  - 指定しないとコンパイルエラーが発生する

```
pragma solidity 0.8.17;
// バージョン0.8.17のコンパイラでコンパイルしなさい
pragma solidity ^0.8.0;
// バージョン0.8.0か、0.8.0よりも新しく、1.0.0 よりも小さいバージョンのコンパイラでコンパイルしなさい
```

コメント

- 行の // 以降は解釈されないのでコメントが書ける

```
uint256 count = 0 // これはコメント
```

状態変数 と メモリ変数

Solidity には状態変数とメモリ変数の２つの変数が存在する.

状態変数（State variables）

- チェーン上（Storage）に記録される
- コントラクト内で宣言した変数は状態変数になる
- Visibility（public，external，internal，private）を設定
- 宣言方法：型 visibility 変数名 (= 初期値)

状態変数とメモリ変数

Solidity には状態変数とメモリ変数の２つの変数が存在する.

メモリ変数（Memory variables）

- 一時的な変数で，メソッドの実行終了時に破棄される
- メソッド内で宣言した変数はメモリ変数になる
- 一時的な変数に状態変数を使うのは無駄になるため，なるべくメモリ変数を使う

状態変数 と メモリ変数

```
contract Sample {
    uint256 public a = 10;
    uint256 internal b = 10;
    uint256 c = 10; // デフォルトは internal
    uint256 private d = 10;
    uint256 constant e = 10;
    function hoge() public {
        uint256 memoryVal = 20; // メモリ変数
    }
}
```

変数の型

値型（Value Types）

- bool：真偽値
- uint：符号なし整数
- int：符号付き整数
- byte：ビット列
- address：アドレス (20byte)
- enum：列挙型

参照型（Reference Types）

- array：配列
- string：”文字列”
- mapping：辞書・連想配列
- struct：構造体

参照型のデータロケーション

全ての参照型は，データロケーションと呼ばれるアノテーションを持つ．これらは参照型のデータがどこに保存されているのかを示す．

- memory：一時的な保存領域，ミュータブル
- calldata：一時的な保存領域，イミュータブル
- storage：永続的な保存領域 = 状態変数

関数（メソッド）

function 関数名 (引数) visibility modifier [returns (<types>)]

- Visibility：変数に同じ.
- Modifier：修飾子
  - pure：純粋関数，状態変数へのアクセスをしない，
  - view：定数関数，状態変数を書き換えない
  - payable：eth を一緒に送信できるようにする
  - pure，view な関数は外部からの呼び出しに gas を消費しない
  - 自分で修飾子を実装することもできる

Contract

- Ethereumにデプロイするプログラムの基本単位
- 変数とメソッドを束ねたもの
  - オブジェクト指向言語におけるクラス
  - カプセル化や再利用をすることも可能
  - 変数・関数の他にも、event、modifier なども含められる

Contract

```
contract SampleContract {
    int256 internal stateVar = 10;

    function currentState() external view returns (int256) {
        return stateVar;
    }
    function add(int256 a, int256 b) external pure returns (int256) {
        return a + b;
    }
}
```

5. 演習

カウンターコントラクト

- メソッドを呼び出すたびにカウントが1増えるカウンターを実装してみる
- 本番環境でインクリメント処理する場合，致命的なバグが発生しやすいため十分に検証されたライブラリを利用する
  - cf. https://docs.openzeppelin.com/contracts/4.x/api/utils#Counters

カウンターコントラクト

- contracts/counter.sol を作成して編集

```
pragma solidity ^0.8.0;

contract Counter {
    uint256 public count = 0;
    function increment() external {
        count += 1;
    }
}
```

カウンターコントラクト

- コンパイル＆実行

```
$ yarn hardhat console
Welcome to Node.js v18.12.1.
Type ".help" for more information.
> const Contract = await ethers.getContractFactory("HelloWorld")
> const contract = await Contract.deploy()
> contract.address
'0x5FbDB2315678afecb367f032d93F642f64180aa3’
> await contract.count()
BigNumber { value: "0" }
```

Crowdfunding コントラクト

実装する機能は以下の通り。

- デプロイ時に目標金額（やメタ情報）を設定機能
- コントラクトでのETHの受け取り機能
- 目標金額に達成したらコントラクトから ETH を取り出せる機能

Crowdfunding コントラクト

コンストラクタを実装

contracts/Crowdfunding.sol

```
contract Crowdfunding {
    constructor(…略…) payable {
        title = _title;
        description = _description;
        target = _target;
        toAddr = _toAddr;
        accomplished = false;
    }
}
```

Crowdfunding コントラクト

コントラクトに送金可能にする　contracts/Crowdfunding.sol

```
contract Crowdfunding {
    …略… // コンストラクタの下に追記
    receive() external payable {}
}
```

Crowdfunding コントラクト

コントラクトに預けられた資金を引き出す

contracts/Crowdfunding.sol

```
contract Crowdfunding {
    // ...中略...
    receive() external payable {}
    function withdraw() public {
        require(target <= address(this).balance,
            "insufficient balance to withdraw");
        toAddr.transfer(address(this).balance);
    }
}
```

Crowdfunding コントラクト

- 関数内に require(bool condition, string reason) を記述すると，condition が false であった場合，tx は revert される
  - アトミック性：revert された場合，tx 開始以降の状態変数の変更は取り消される
  - revert された場合、gas は消費された分以外返却される
- コントラクト自身の残高は address(this).balance で参照
- address.transfer(uint amoumt) でコントラクトから送金

ローカルネットワークにデプロイ

1. scripts/deploy.ts と scripts/send.ts を開き指示の通りに修正
2. yarn hardhat node を実行

   このシェルのプロセスはテストする間はずっと起動させ続けること!!
3. 別のシェルを立ち上げて以下を実行
   - yarn hardhat run scripts/deploy.ts --network localhost
   - yarn hardhat run scripts/sendETH.ts --network localhost
   - yarn dev

Sepolia テストネットワークにデプロイ

1. sample.env をコピーして、.env にファイル名を改名
2. .env にSepoliaETH を保有するアカウントの秘密鍵を記入
   - ***秘密鍵の扱いは厳重に注意！！***
   - .env は git の差分に含まない。（.gitignore に追加済）
3. yarn hardhat run scripts/deploy.ts --network sepolia を実行
4. フロントエンドであれこれテストしてみる

おまけ課題（時間が余ったらやる）

- withdraw は送金先のアドレスの持ち主のみ実行可能にする
  - msg.sender で tx 送信者のアドレスを取得できる
- このままでは完了後にも ETH が送金できてしまうので,
  - withdraw されたタイミングで accomplished フラグを true に設定するように実装
  - accomplished が false である場合のみ送金を受け付けるように実装
- 新たに状態変数を加え，最低・最高送金限度額を実装
- 新たに状態変数を加え，各アカウントの寄付額を記録し，外部から参照できるように実装