Disciplina:

SMART CONTRACTS

Professor: Pablo V. Rego

PUC Minas

SMAC_2022Sem02–Mod.04

PUC Minas

Agenda Módulo #MN : Smart Contracts Overview I

- **Smart Contracts : Fundamentos**
  - **Conceito**
  - **Ciclo de Vida**
  - **Linguagens de Programação**

- **Solidity : Conceitos**
  - **Anatomia**
  - **Bibliotecas**
  - **Compilação**
  - **Deployment**
  - Verificação
  - Testes
  - Upgrade
  - Segurança
  - Auditoria
  - Composabilidade

- Redes de Desenvolvimento
- Frameworks de Desenvolvimento
- Ethereum APIs
- Data Mining

PUC Minas

Módulo 04

Smart Contracts Overview

PUC Minas

Tópico :

SMART CONTRACTS : FUNDAMENTOS

PUC Minas

O que são Smart Contracts?

Um "contrato inteligente" é simplesmente um programa executado na blockchain Ethereum.

É uma coleção de código (suas funções) e dados (seu estado) que reside em um endereço específico no blockchain Ethereum.

No caso da Ethereum Blockchain, Contratos inteligentes são um tipo de conta Ethereum.

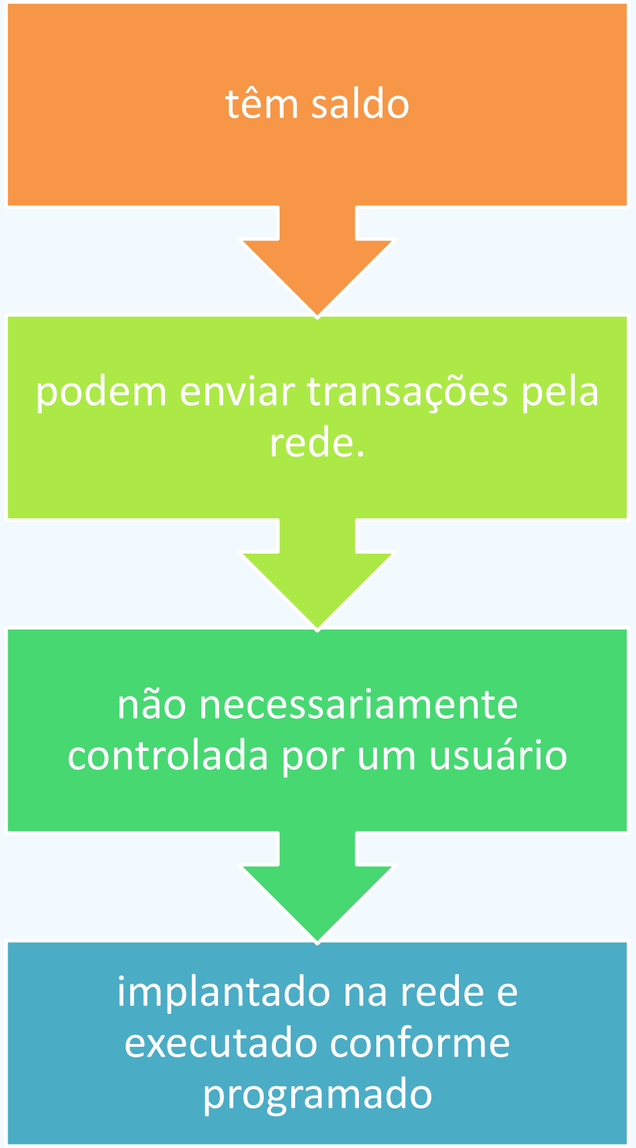
As contas de usuário podem interagir com um contrato inteligente enviando transações

Pode definir regras, como um contrato regular

Execução automática através do código (Enforcement)

Não podem ser excluídos por padrão

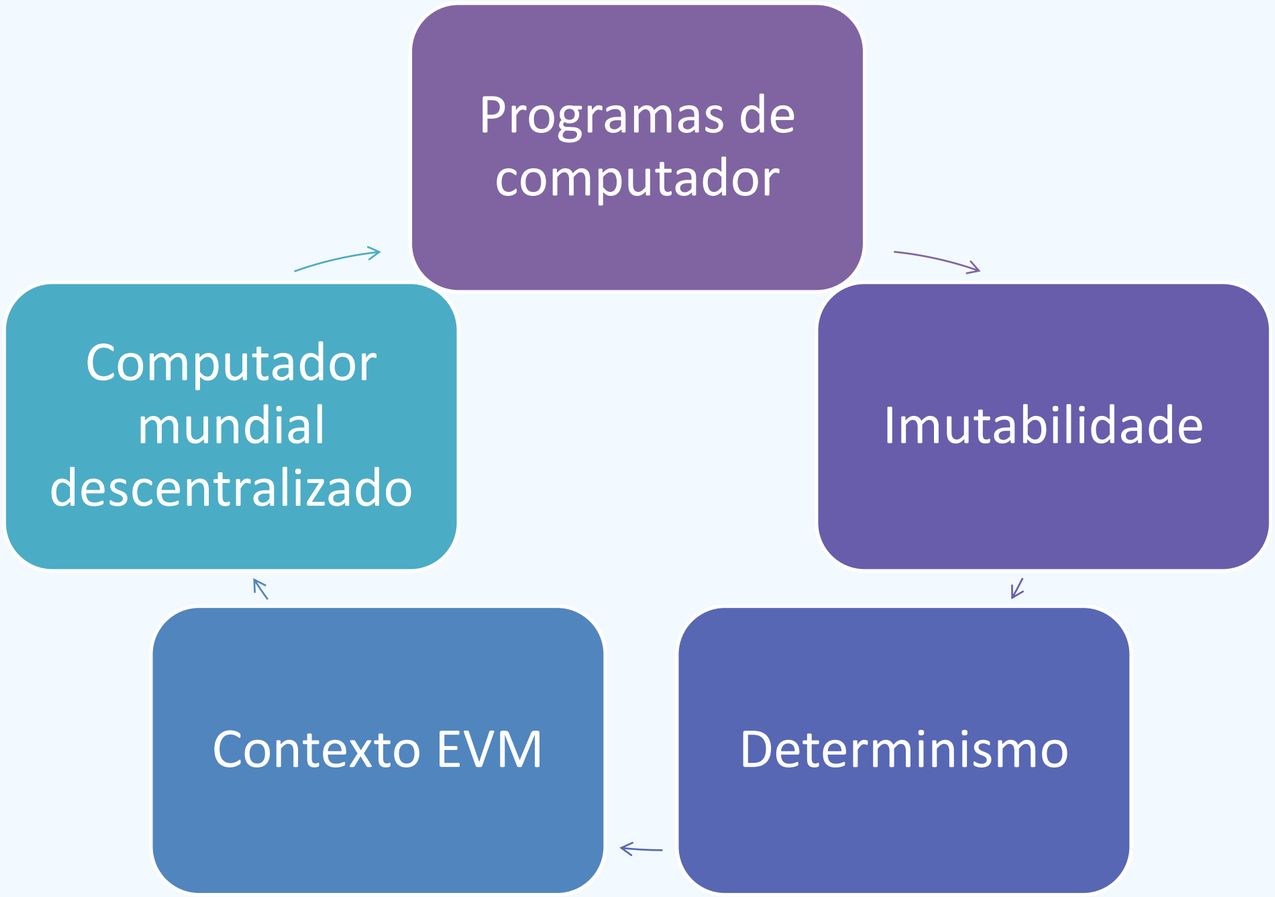
As interações com eles são irreversíveis.



PUC Minas

Definição Aplicável

Programas de computador imutáveis que são executados deterministicamente no contexto de uma máquina virtual Ethereum como parte do protocolo de rede Ethereum - ou seja, no computador mundial Ethereum descentralizado.



PUC Minas

Confiança e Contratos

- Contratos convencionais podem ser sujeitos a revogação não consensual
- Exemplo : João e Maria apostam em uma corrida de bicicleta

1. João aposta 100
2. Maria aposta 100
3. A competição ocorre
4. Maria ganha
5. João decide não pagar 100 a Maria

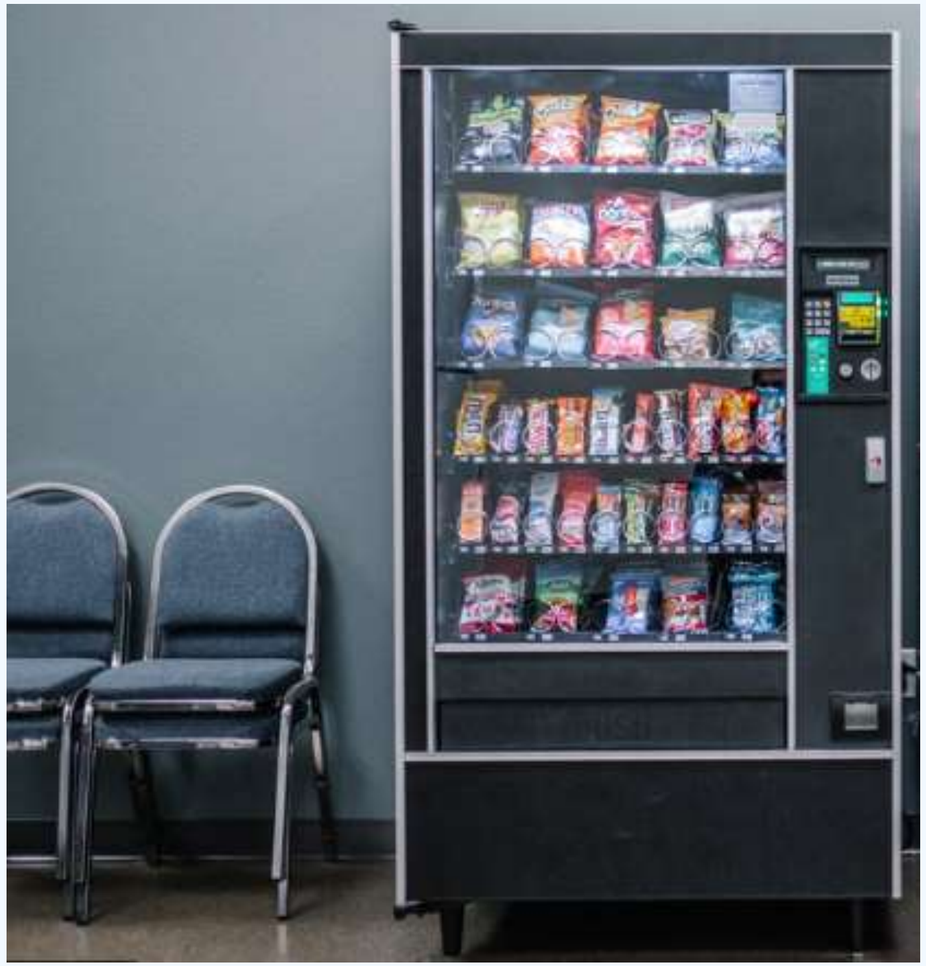
Características Notáveis

- Previsibilidade
- Registro público
- Proteção de privacidade
- Termos visíveis

PUC Minas

Exemplo : Máquina de Venda

- Digital Vending Machine : The Idea of Smart Contracts (Nick Szabo, 1997)



Dinheiro + Seleção = Liberação

Entradas específicas garantem saídas predeterminadas.

1. Você seleciona um produto
2. A máquina de venda automática retorna o valor necessário para adquirir o produto
3. Você insere a quantidade correta
4. A máquina de venda automática verifica se você inseriu a quantidade correta
5. A máquina de venda automática dispensa o produto de sua escolha

`vendingmachine.sol`

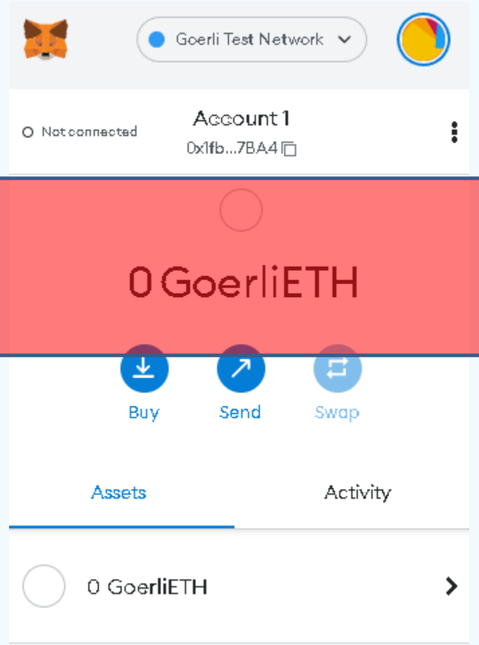
PUC Minas

vendingmachine.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.7 <0.9.0;

contract VendingMachine {

    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the contract
    // 2. set the deployed smart contract's cupcake balance to 100
    constructor() {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 100;
    }

    // Allow the owner to increase the smart contract's cupcake balance
    function refill(uint amount) public {
        require(msg.sender == owner, "Only the owner can refill.");
        cupcakeBalances[address(this)] += amount;
    }

    // Allow anyone to purchase cupcakes
    function purchase(uint amount) public payable {
        require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per cupcake");
        require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock to complete this purchase");
        cupcakeBalances[address(this)] -= amount;
        cupcakeBalances[msg.sender] += amount;
    }
}
```

PUC Minas

MetaMask : Free Ether (Testnets)



Faucets (nem sempre funcionam)

- https://faucet.metamask.io/
- Paradigm :
  - https://faucet.paradigm.xyz
- *Ropsten*:
  - https://faucet.metamask.io
- *Rinkeby*:
  - https://faucet.rinkeby.io
  - https://www.rinkebyfaucet.com
  - https://app.mycrypto.com/faucet
  - https://faucets.chain.link/rinkeby
- *Kovan*:
  - https://gitter.im/kovan-testnet/faucet
- *Goerli*:
  - https://goerli-faucet.slock.it/index.html
  - https://faucet.goerli.mudit.blog

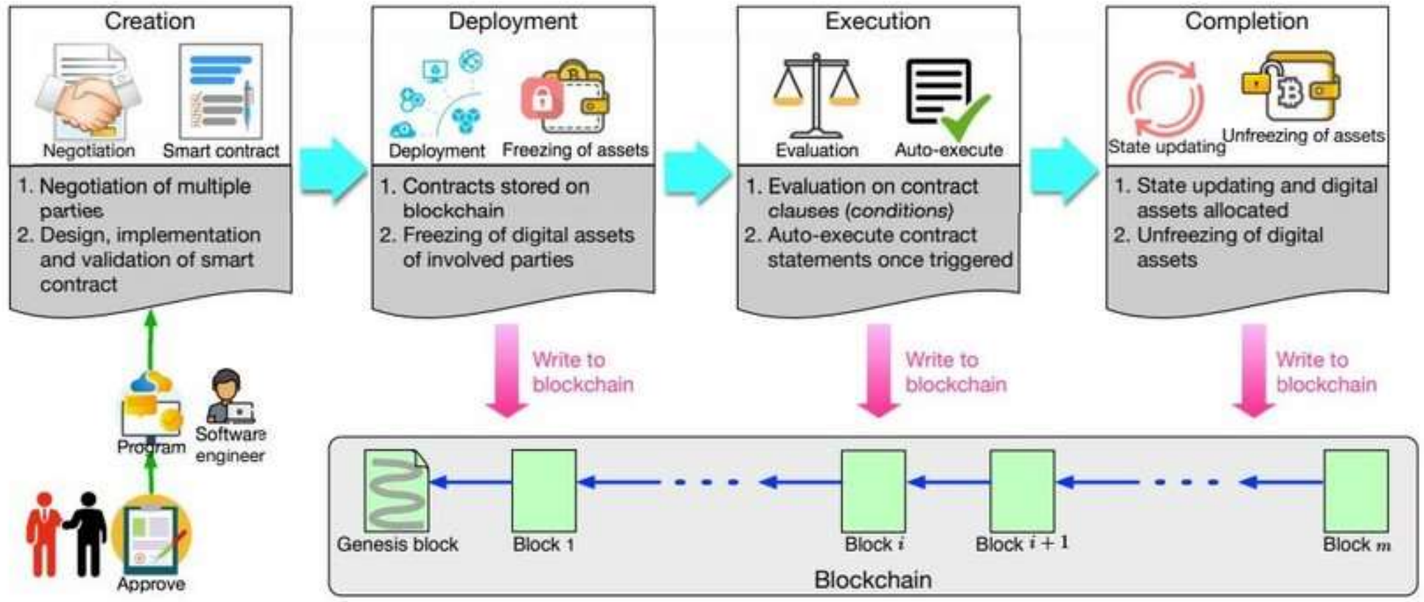
Todos já configurados?

PUC Minas

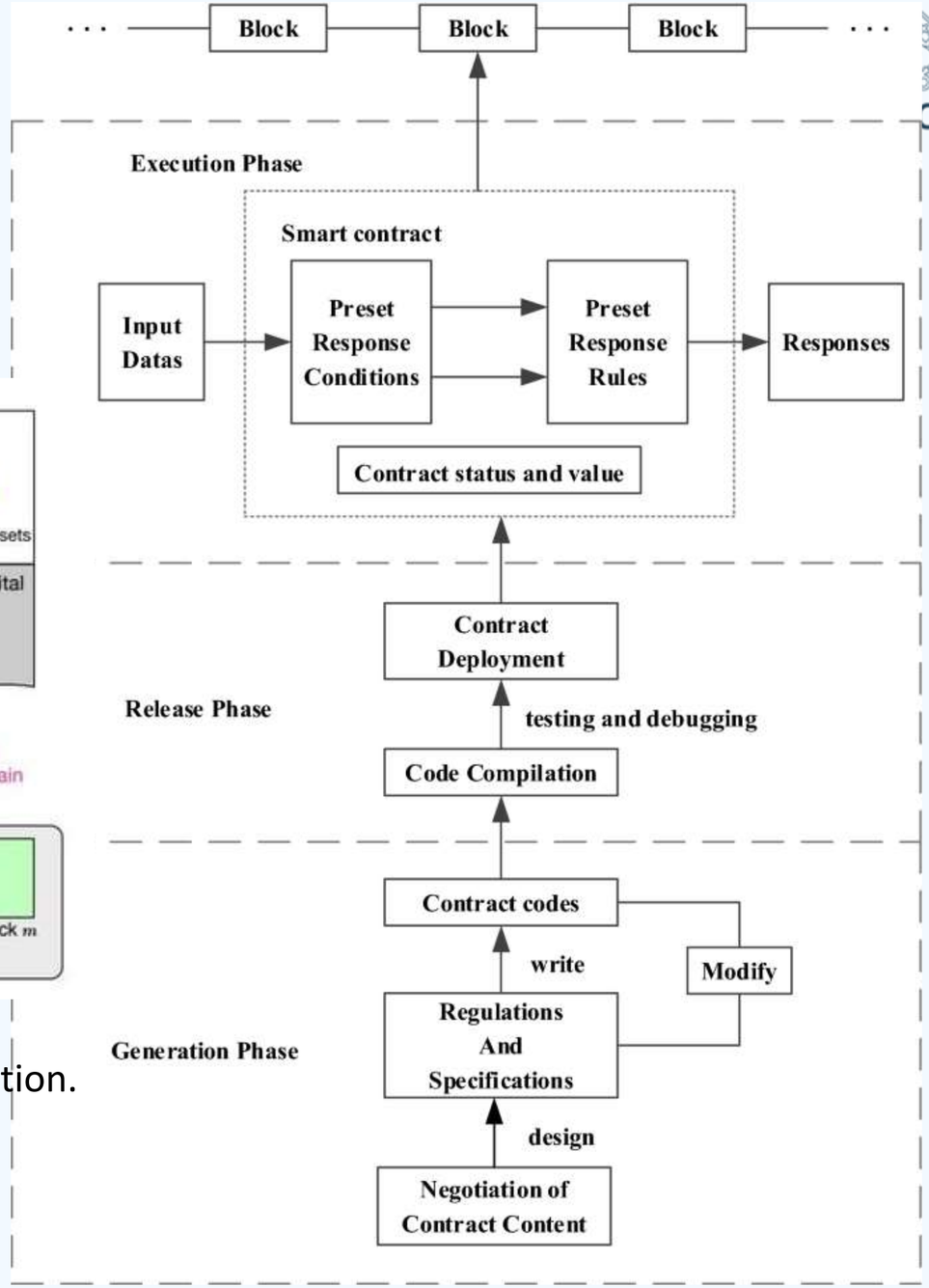
Remix

- http://remix.ethereum.org/
- Plugins
- Configuração do compilador
- Node + Remixd
  - https://nodejs.org/en/download/
  - npm install –g remix-project/remixd

Ciclo de Vida



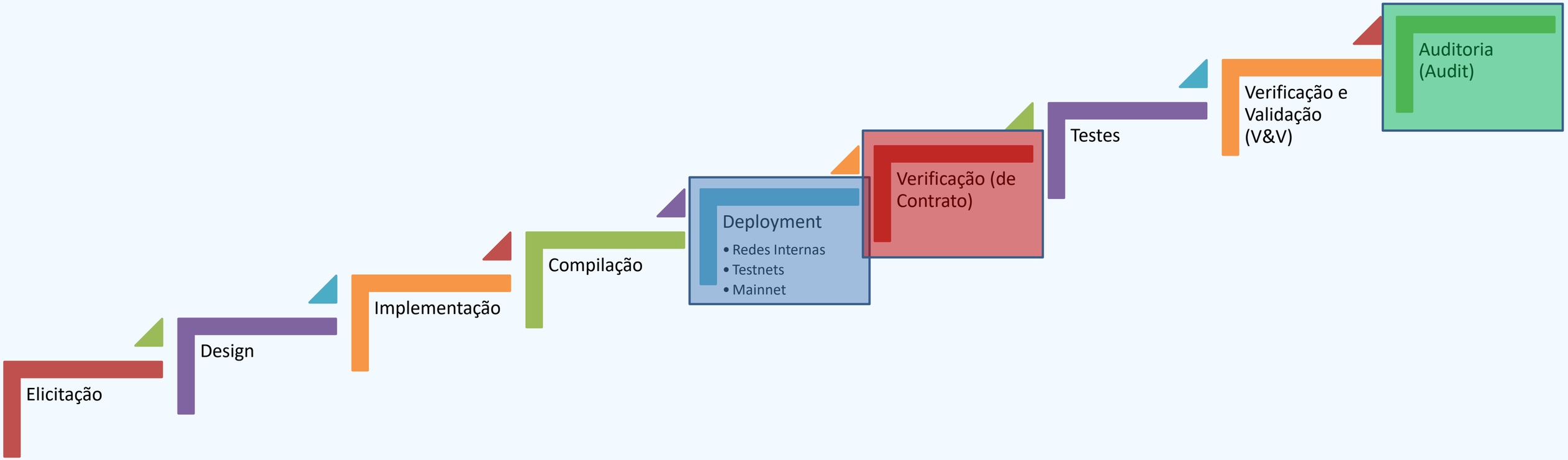
Jani, Shailak. (2020). Smart Contracts: Building Blocks for Digital Transformation. 10.13140/RG.2.2.33316.83847.



Lin, SY., Zhang, L., Li, J. *et al.* A survey of application research based on blockchain smart contract. *Wireless Netw* **28**, 635–690 (2022). https://doi.org/10.1007/s11276-021-02874-x

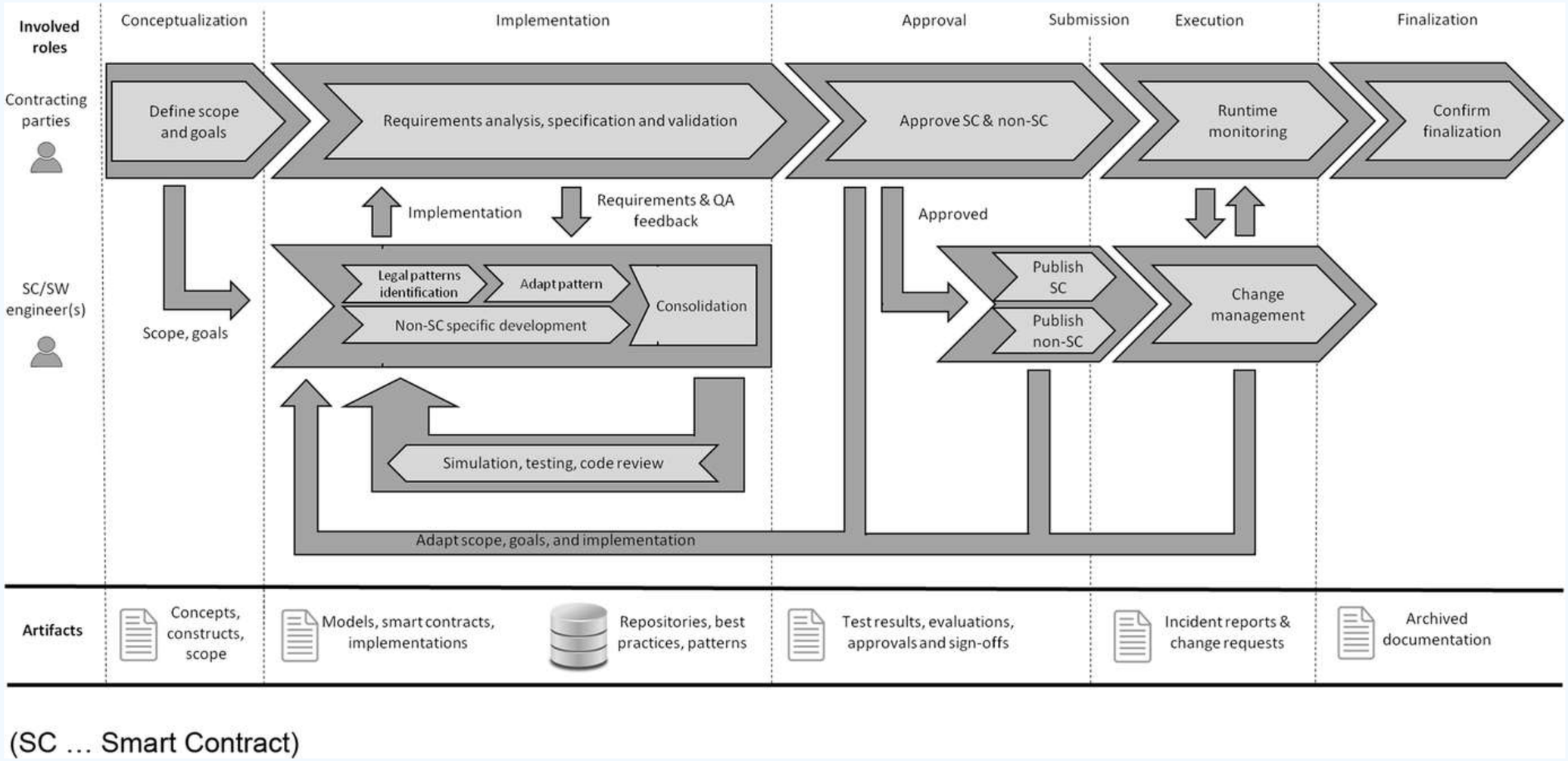
PUC Minas

Fases de Desenvolvimento no Ciclo de Vida



PUC Minas

SDLC & Smart Contracts



(SC … Smart Contract)

[Laying the foundation for smart contract development: an integrated engineering process model (Sillaber et al, 2021)]

Linguagens de Programação de Contratos

- **Solidity**
- Vyper
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

- Linguagem de alto nível orientada a objetos para implementação de contratos inteligentes.
- Linguagem de colchetes que foi mais profundamente influenciada por C++.
- Estaticamente tipado (o tipo de uma variável é conhecido em tempo de compilação).
- Suporta:
  - Herança (você pode estender outros contratos).
  - Bibliotecas (você pode criar código reutilizável que pode chamar de diferentes contratos – como funções estáticas em uma classe estática em outras linguagens de programação orientadas a objetos).
  - Tipos complexos definidos pelo usuário.

Linguagens de Programação de Contratos

- **Solidity**
- Vyper
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

- https://docs.soliditylang.org/en/latest/
- https://soliditylang.org
- https://github.com/ethereum/solidity/
- https://reference.auditless.com/cheatsheet/
- https://blog.soliditylang.org
- https://twitter.com/solidity_lang

PUC Minas

Linguagens de Programação de Contratos

- **Solidity**
- Vyper
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.7.0;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Linguagens de Programação de Contratos

- Solidity
- **Vyper**
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

- Baseada em Python
- Strong Typed
- Código de compilador pequeno e compreensível
- Geração de bytecode eficiente
- Deliberadamente tem menos recursos que o Solidity com o objetivo de tornar os contratos mais seguros e fáceis de auditar.
- Vyper não suporta:
  - Modificadores
  - Herança
  - Montagem em linha
  - Sobrecarga de funções
  - Sobrecarga do operador
  - Chamada recursiva
  - Loops de comprimento infinito
  - Pontos fixos binários

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- **Vyper**
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

- https://vyper.readthedocs.io/en/stable/
- https://github.com/vyperlang/vyper
- https://reference.auditless.com/cheatsheet/
- https://github.com/SupremacyTeam/VyperPunk
- https://www.vyperexamples.com
- https://github.com/zcor/vyper-dev
- https://github.com/pynchmeister/vyper-greatest-hits/tree/main/contracts
- https://github.com/spadebuilders/awesome-vyper

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- **Vyper**
- Yul/+
- FE
- LLL
- Serpent
- Bamboo

```
# Open Auction

# Auction params
# Beneficiary receives money from the highest bidder
beneficiary: public(address)
auctionStart: public(uint256)
auctionEnd: public(uint256)

# Current state of auction
highestBidder: public(address)
highestBid: public(uint256)

# Set to true at the end, disallows any change
ended: public(bool)

# Keep track of refunded bids so we can follow the withdraw pattern
pendingReturns: public(HashMap[address, uint256])

# Create a simple auction with `_bidding_time`
# seconds bidding time on behalf of the
# beneficiary address `_beneficiary`.
@external
def __init__(_beneficiary: address, _bidding_time: uint256):
    self.beneficiary = _beneficiary
    self.auctionStart = block.timestamp
    self.auctionEnd = self.auctionStart + _bidding_time

# Bid on the auction with the value sent
# together with this transaction.
# The value will only be refunded if the
# auction is not won.
@external
@payable
def bid():
    # Check if bidding period is over.
    assert block.timestamp < self.auctionEnd
    # Check if bid is high enough
    assert msg.value > self.highestBid
    # Track the refund for the previous high bidder
    self.pendingReturns[self.highestBidder] += self.highestBid
    # Track new high bid
    self.highestBidder = msg.sender
    self.highestBid = msg.value
```

```
# Withdraw a previously refunded bid. The withdraw pattern is
# used here to avoid a security issue. If refunds were directly
# sent as part of bid(), a malicious bidding contract could block
# those refunds and thus block new higher bids from coming in.
@external
def withdraw():
    pending_amount: uint256 = self.pendingReturns[msg.sender]
    self.pendingReturns[msg.sender] = 0
    send(msg.sender, pending_amount)

# End the auction and send the highest bid
# to the beneficiary.
@external
def endAuction():
    # It is a good guideline to structure functions that interact
    # with other contracts (i.e. they call functions or send ether)
    # into three phases:
    # 1. checking conditions
    # 2. performing actions (potentially changing conditions)
    # 3. interacting with other contracts
    # If these phases are mixed up, the other contract could call
    # back into the current contract and modify the state or cause
    # effects (ether payout) to be performed multiple times.
    # If functions called internally include interaction with external
    # contracts, they also have to be considered interaction with
    # external contracts.

    # 1. Conditions
    # Check if auction endtime has been reached
    assert block.timestamp >= self.auctionEnd
    # Check if this function has already been called
    assert not self.ended

    # 2. Effects
    self.ended = True

    # 3. Interaction
    send(self.beneficiary, self.highestBid)
```

Linguagens de Programação de Contratos

- Solidity
- Vyper
- **Yul/+**
- FE
- LLL
- Serpent
- Bamboo

- Yul/Julia/Yul+
- Linguagem intermediária para Ethereum.
- Suporta o EVM e Ewasm, um WebAssembly com sabor de Ethereum, e foi projetado para ser um denominador comum utilizável de ambas as plataformas.
- Bom alvo para estágios de otimização de alto nível que podem beneficiar igualmente as plataformas EVM e Ewasm.

- Yul+
  - Uma extensão de baixo nível e altamente eficiente para Yul.
  - Inicialmente projetado para um contrato cumulativo otimista.
  - O Yul+ pode ser visto como uma proposta de atualização experimental para o Yul, adicionando novos recursos a ele.

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- Vyper
- **Yul/+**
- FE
- LLL
- Serpent
- Bamboo

- https://docs.soliditylang.org/en/latest/yul.html
- https://github.com/fuellabs/yulp
- https://yulp.fuel.sh
- https://fuellabs.medium.com/introducing-yul-a-new-low-level-language-for-ethereum-aa64ce89512f

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- Vyper
- **Yul/+**
- FE
- LLL
- Serpent
- Bamboo

```
{
    function power(base, exponent) -> result
    {
        switch exponent
        case 0 { result := 1 }
        case 1 { result := base }
        default
        {
            result := power(mul(base, base), div(exponent, 2))
            if mod(exponent, 2) { result := mul(base, result)
}
        }
    }
    let res := power(calldataload(0), calldataload(32))
    mstore(0, res)
    return(0, 32)
}
```

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- **FE**
- LLL
- Serpent
- Bamboo

- Linguagem estaticamente tipada para a Ethereum Virtual Machine (EVM).
- Inspirado em Python e Rust.
- Tem como objetivo ser fácil de aprender - mesmo para desenvolvedores que são novos no ecossistema Ethereum.
- O desenvolvimento do Fe ainda está em seus estágios iniciais, a linguagem teve seu lançamento alfa em janeiro de 2021.

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- **FE**
- LLL
- Serpent
- Bamboo

- https://github.com/ethereum/fe
- https://blog.fe-lang.org/posts/fe-a-new-language-for-the-ethereum-ecosystem/
- https://notes.ethereum.org/LVhaTF30SJOpkbG1iVw1jg
- https://twitter.com/official_fe

PUC Minas

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- **FE**
- LLL
- Serpent
- Bamboo

```
type BookMsg = bytes[100]

contract GuestBook:
    pub guest_book: map<address, BookMsg>

    event Signed:
        book_msg: BookMsg

    pub def sign(book_msg: BookMsg):
        self.guest_book[msg.sender] = book_msg

        emit Signed(book_msg=book_msg)

    pub def get_msg(addr: address) -> BookMsg:
        return self.guest_book[addr].to_mem()
```

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- FE
- **LLL**
- Serpent
- Bamboo

- Uma linguagem de programação funcional (declarativa), com sintaxe semelhante ao Lisp.
- Foi a primeira linguagem de alto nível para contratos inteligentes Ethereum, mas raramente é usada hoje.

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- FE
- LLL
- **Serpent**
- Bamboo

- Uma linguagem de programação procedural (imperativa) com uma sintaxe semelhante ao Python.
- Também pode ser usado para escrever código funcional (declarativo), embora não seja totalmente livre de efeitos colaterais.

Linguagens de Programação de Contratos

- Solidity
- Vyper
- Yul/+
- FE
- LLL
- Serpent
- **Bamboo**

- Uma linguagem recém-desenvolvida, influenciada por Erlang, com transições de estado explícitas e sem fluxos iterativos (loops).
- Destina-se a reduzir os efeitos colaterais e aumentar a auditabilidade.
- Muito novo e ainda a ser amplamente adotado.

Outras Linguagens

- Ethereum for Dart developers
- Ethereum for Delphi developers
- Ethereum for .NET developers
- Ethereum for Go developers
- Ethereum for Java developers
- Ethereum for JavaScript developers
- Ethereum for Python developers
- Ethereum for Ruby developers
- Ethereum for Rust developers

PUC Minas

Tópico :

SOLIDITY : FUNDAMENTOS

PUC Minas

Estrutura do Smart Contract

- Preâmbulo
  - Licença
  - Versão
  - Imports
- Contracts
- Estruturas de dados
- Escopo

PUC Minas

Anatomia das Estruturas de Dados

DATA

- Storage
- Memory
- Address e Tipos Básicos
- Environment variables

- Assignments : storage / memory
- É caro modificar o armazenamento em um contrato inteligente, portanto, você precisa considerar onde seus dados devem residir.

PUC Minas

Anatomia das Estruturas de Dados

DATA

- **Storage**
- Memory
- Address e Tipos Básicos
- Environment variables

- Dados persistentes
- Variáveis de estado.
- Armazenados permanentemente no blockchain.
- Você precisa declarar o tipo para que o contrato possa calcular quanto armazenamento no blockchain ele precisa quando compilar.

PUC Minas

Anatomia das Estruturas de Dados

DATA

- **Storage**
- Memory
- Address e Tipos Básicos
- Environment variables

```
// Solidity example
contract SimpleStorage {
    uint storedData; // State variable
    // ...
}
```

PUC Minas

Anatomia das Estruturas de Dados

DATA

- Storage
- **Memory**
- Address e Tipos Básicos
- Environment variables

- Armazenados apenas durante a vida útil da execução de uma função de contrato
- Muito mais baratos de usar.

PUC Minas

Anatomia das Estruturas de Dados

DATA

- Storage
- Memory
- **Address e Tipos Básicos**
- Environment variables

- Address pode conter um endereço Ethereum que equivale a 20 bytes ou 160 bits.
- Retorna em notação hexadecimal com um 0x inicial.
- Outros tipos incluem:
  - boolean
  - integer
  - fixed point numbers
  - fixed-size byte arrays
  - dynamically-sized byte arrays
  - Rational and integer literals
  - String literals
  - Hexadecimal literals
  - Enums

PUC Minas

Anatomia das Estruturas de Dados

DATA

- Storage
- Memory
- Address e Tipos Básicos
- **Environment variables**

- Variáveis globais especiais.
- Fornecer informações sobre a blockchain ou a transação atual.

Prop	State variable	Description
block.timestamp	uint256	Current block epoch timestamp
msg.sender	address	Sender of the message (current call)

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- Built-in functions
- WRITING FUNCTIONS

- Obtenção de informações ou definir informações em resposta a transações recebidas.

Tipos de chamadas

- Internal – estes não criam uma chamada EVM
  - Funções internas e variáveis de estado só podem ser acessadas internamente (ou seja, de dentro do contrato atual ou contratos derivados dele)
- External – estes criam uma chamada EVM
  - As funções externas fazem parte da interface do contrato, o que significa que podem ser chamadas de outros contratos e por meio de transações. Uma função externa f não pode ser chamada internamente (ou seja, f() não funciona, mas this.f() funciona).

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- Built-in functions
- WRITING FUNCTIONS

Eles também podem ser públicos ou privados

- funções públicas podem ser chamadas internamente de dentro do contrato ou externamente por meio de mensagens
- funções privadas são visíveis apenas para o contrato em que são definidas e não em contratos derivados

Tanto as funções quanto as variáveis de estado podem ser públicas ou privadas

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- Built-in functions
- WRITING FUNCTIONS

- O valor do parâmetro do tipo string é passado para a função
- É declarado público, o que significa que qualquer pessoa pode acessá-lo
- Não é uma view declarada, então pode modificar o estado do contrato

```
// Solidity example
function update_name(string value) public {
    dapp_name = value;
}
```

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- **View functions**
- Constructor functions
- Built-in functions
- WRITING FUNCTIONS

- Essas funções prometem não modificar o estado dos dados do contrato.
- Exemplos comuns são funções "getter" – você pode usar isso para receber o saldo de um usuário, por exemplo.

```
// Solidity example
function balanceOf(address _owner) public
view returns (uint256 _balance) {
    return ownerPizzaCount[_owner];
}
```

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- **View functions**
- Constructor functions
- Built-in functions
- WRITING FUNCTIONS

O que é considerado modificação de estado:

- Escrevendo em variáveis de estado.
- Emissão de eventos.
- Criação de outros contratos.
- Utilizar autodestruição.
- Envio de Eth através de chamadas.
- Chamando qualquer função não marcada como view ou pura.
- Utilizar chamadas de baixo nível.
- Utilizar o assembly embutido que contém determinados opcodes.

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- **Constructor functions**
- Built-in functions
- WRITING FUNCTIONS

Construtores

- Executados apenas uma vez quando o contrato é implantado pela primeira vez.
- Como construtor em muitas linguagens de programação baseadas em classes, essas funções geralmente inicializam variáveis de estado para seus valores especificados.

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- **Constructor functions**
- Built-in functions
- WRITING FUNCTIONS

```
// Solidity example
// Initializes the contract's data, setting the `owner`
// to the address of the contract creator.
constructor() public {
    // All smart contracts rely on external transactions to trigger its functions.
    // `msg` is a global variable that includes relevant data on the given transaction,
    // such as the address of the sender and the ETH value included in the transaction.
    // Learn more: https://solidity.readthedocs.io/en/v0.5.10/units-and-global-variables.html#block-and-transaction-properties
    owner = msg.sender;
}
```

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- **Built-in functions**
- WRITING FUNCTIONS

Além das variáveis e funções que você define em seu contrato, existem algumas funções especiais incorporadas. O exemplo mais óbvio é:

```
address.send()
```

Isso permite que os contratos enviem ETH para outras contas.

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- Built-in functions
- **WRITING FUNCTIONS**

Sua função precisa:

- Variável e tipo de parâmetro (se aceitar parâmetros)
- Declaração de interno/externo
- Declaração de puro/view/payable
- Retorna tipo (se retornar um valor)

PUC Minas

Anatomia das Estruturas de Dados

FUNCTIONS

- View functions
- Constructor functions
- Built-in functions
- **WRITING FUNCTIONS**

```
pragma solidity >=0.4.0 <=0.6.0;

contract ExampleDapp {
    string dapp_name; // state variable

    // Called when the contract is deployed and
initializes the value
    constructor() public {
        dapp_name = "My Example dapp";
    }

    // Get Function
    function read name() public view
returns(string) {
        return dapp_name;
    }

    // Set Function
    function update_name(string value) public {
        dapp_name = value;
    }
}
```

PUC Minas

Anatomia das Estruturas de Dados

EVENTS AND LOGS

- Os eventos permitem que você se comunique com seu contrato inteligente a partir de seu front-end ou de outros aplicativos de assinatura.
- Quando uma transação é extraída, os contratos inteligentes podem emitir eventos e gravar logs no blockchain que o frontend pode processar.

Anatomia das Estruturas de Dados

LIBRARIES

- Blocos de construção em bibliotecas :
  - Reusable behaviors
  - Implementação de padrões
  - Libraries

Anatomia das Estruturas de Dados

REUSABLE BEHAVIORS

- Ao escrever contratos inteligentes, uma boa chance de você se encontrar com padrões idênticos, como designar um endereço de admin para realizar consultas protegidas em um contrato ou adicionar um botão de emergência no caso de um problema inesperado.

- As bibliotecas de contrato inteligente geralmente fornecem implementações reutilizáveis desses comportamentos como bibliotecas ou por meio de herança no Solidity.

PUC Minas

Anatomia das Estruturas de Dados

REUSABLE BEHAVIORS

- Como exemplo, a seguir está uma versão simplificada do contrato Ownable da biblioteca OpenZeppelin Contracts, que designa um endereço como o proprietário de um contrato e fornece um modificador para restringir o acesso a um método somente a esse proprietário.

```
contract Ownable {
    address public owner;

    constructor() internal {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(owner == msg.sender, "Ownable: caller is
not the owner");
        _;
    }
}
.....

import ".../Ownable.sol"; // Path to the imported library

contract MyContract is Ownable {
    // The following function can only be called by the
owner
    function secured() onlyOwner public {
        msg.sender.transfer(1 ether);
    }
}
```

PUC Minas

Anatomia das Estruturas de Dados

REUSABLE BEHAVIORS

- Quando utilizar?

- Usar uma biblioteca de contrato inteligente para seu projeto tem vários benefícios. Em primeiro lugar, você economiza tempo fornecendo blocos de construção prontos para uso que você pode incluir em seu sistema, em vez de ter que codificá-los você mesmo.

- A segurança também é uma grande vantagem. As bibliotecas de contratos inteligentes de código aberto também costumam ser fortemente examinadas. Dado que muitos projetos dependem deles, há um forte incentivo por parte da comunidade para mantê-los sob constante revisão. É muito mais comum encontrar erros no código do aplicativo do que em bibliotecas de contrato reutilizáveis. Algumas bibliotecas também passam por auditorias externas para segurança adicional.

- No entanto, o uso de bibliotecas de contrato inteligente acarreta o risco de incluir código com o qual você não está familiarizado em seu projeto. É tentador importar um contrato e incluí-lo diretamente em seu projeto, mas sem uma boa compreensão do que esse contrato faz, você pode estar introduzindo inadvertidamente um problema em seu sistema devido a um comportamento inesperado. Certifique-se sempre de ler a documentação do código que está importando e, em seguida, revise o próprio código antes de torná-lo parte do seu projeto!

- Por último, ao decidir se deve incluir uma biblioteca, considere seu uso geral. Um amplamente adotado tem os benefícios de ter uma comunidade maior e mais olhos analisando os problemas. A segurança deve ser seu foco principal ao construir com contratos inteligentes!

PUC Minas

Anatomia das Estruturas de Dados

REUSABLE BEHAVIORS

OpenZeppelin Contracts

- GitHub
- Community Forum

DappSys

- Documentation
- GitHub

HQ20

- GitHub

Compilação

- Você precisa compilar seu contrato para que seu aplicativo da Web e a máquina virtual Ethereum (EVM) possam entendê-lo.

Para que o EVM possa executar seu contrato, ele precisa estar em bytecode.

- A compilação transforma isso:

```
pragma solidity 0.4.24;
contract Greeter {
    function greet() public constant returns (string) {
        return "Hello";
    }
}
```

- Nisto:

PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH2 0x41 JUMPI PUSH1 0x0 CALLDATALOAD PUSH29 0x100000000000000000000000000000000000000000000000000000000 SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0xCFAE3217 EQ PUSH2 0x46 JUMPI JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0x52 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x5B PUSH2 0xD6 JUMP JUMPDEST PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3 DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1 PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0 JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x9B JUMPI DUP1 DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20 DUP2 ADD SWAP1 POP PUSH2 0x80 JUMP JUMPDEST POP POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1 0x1F AND DUP1 ISZERO PUSH2 0xC8 JUMPI DUP1 DUP3 SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2 0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1 0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST PUSH1 0x60 PUSH1 0x40 DUP1 MLOAD SWAP1 DUP2 ADD PUSH1 0x40 MSTORE DUP1 PUSH1 0x5 DUP2 MSTORE PUSH1 0x20 ADD PUSH32 0x48656C6C6F0000000000000000000000000000000000000000000000000000 DUP2 MSTORE POP SWAP1 POP SWAP1 JUMP STOP LOG1 PUSH6 0x627A7A723058 KECCAK256 SLT 0xec 0xe 0xf5 0xf8 SLT 0xc7 0x2d STATICCALL ADDRESS SHR 0xdb COINBASE 0xb1 BALANCE 0xe8 0xf8 DUP14 0xda 0xad DUP13 LOG1 0x4c 0xb4 0x26 0xc2 DELEGATECALL PUSH7 0x8994D3E002900

Deployment

- Você precisa implantar seu contrato inteligente para que ele esteja disponível para usuários de uma rede Ethereum.

- Para implantar um contrato inteligente, basta enviar uma transação Ethereum contendo o código compilado do contrato inteligente sem especificar nenhum destinatário.

O que você precisará

- o bytecode do seu contrato – isso é gerado através da compilação
- ETH para gás - você definirá seu limite de gás como outras transações, portanto, esteja ciente de que a implantação do contrato precisa de muito mais gás do que uma simples transferência de ETH
- um script de implantação ou plug-in
- acesso a um nó Ethereum, executando o seu próprio, conectando-se a um nó público ou por meio de uma chave de API usando um serviço de nó como Infura ou Alchemy

PUC Minas

Ferramentas para Deployment

- **Remix**
  - O Remix IDE permite desenvolver, implantar e administrar contratos inteligentes para Ethereum como blockchains
  - Remix
- **Tenderly**
  - Simule, depure e monitore qualquer coisa em cadeias compatíveis com EVM, com dados em tempo real
  - tenderly.co
  - Docs
  - GitHub
- **Hardhat**
  - Um ambiente de desenvolvimento para compilar, implantar, testar e depurar seu software Ethereum
  - hardhat.org
  - Docs on deploying your contracts
  - GitHub
- **Truffle**
  - Um ambiente de desenvolvimento, estrutura de teste, pipeline de construção e outras ferramentas.
  - trufflesuite.com
  - Docs on networks and app deployment
  - GitHub

Próximos Tópicos

- Módulo #5 : Goto Solidity

→Verificação Formal

→Testes

→Upgradability

→Vanity, Shell Deployment, Segurança

→Melhores Práticas

Além de :

- Redes de Desenvolvimento
- Frameworks de Desenvolvimento
- Ethereum APIs
- Data Mining