

PROF. **DANILO CURVELO**

Solidity

Smart Contracts são normalmente escritos em uma linguagem de alto nível, como o **Solidity**

Depois de compilada, é realizado o *deploy* na plataforma Ethereum usando uma transação especial de **criação de contrato**

E não podem ser mais modificado (imutável), mas pode ser deletado (se usar opcode SELFDESTRUCT)

Cada contrato é identificado por um endereço, mas não tem chaves privadas associadas (ou seja, não tem dono)

Compilador Solidity

```
pragma solidity ^0.4.0; contract Demo { // TODO } 60606040523415600e67...
```

Exemplo

Faucet.sol

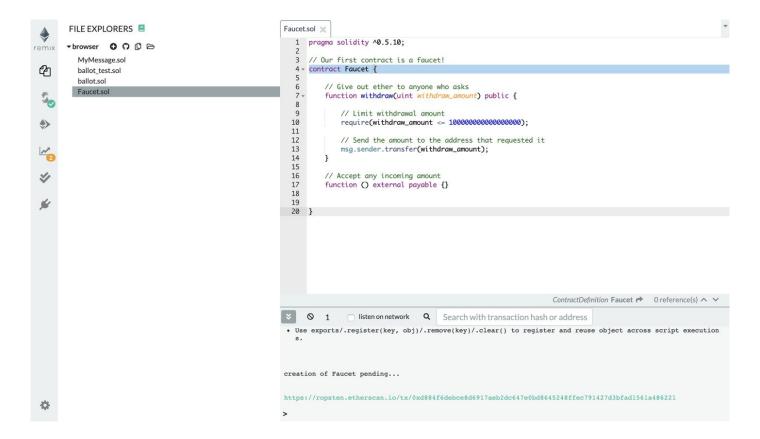
```
pragma solidity ^0.5.10;
contract Faucet {
    function withdraw(uint withdraw_amount) public {
        // Limita a quantidade de transferência
        require(withdraw_amount <= 100000000000000000);</pre>
        // Envia a quantidade para o endereço que requisitou
        msg.sender.transfer(withdraw amount);
    // Aceita qualquer quantidade enviada para o Faucet
    function () external payable {}
```

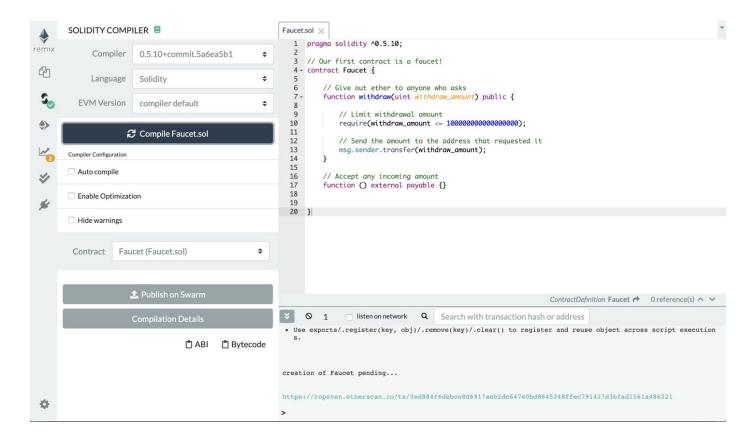
Exemplo

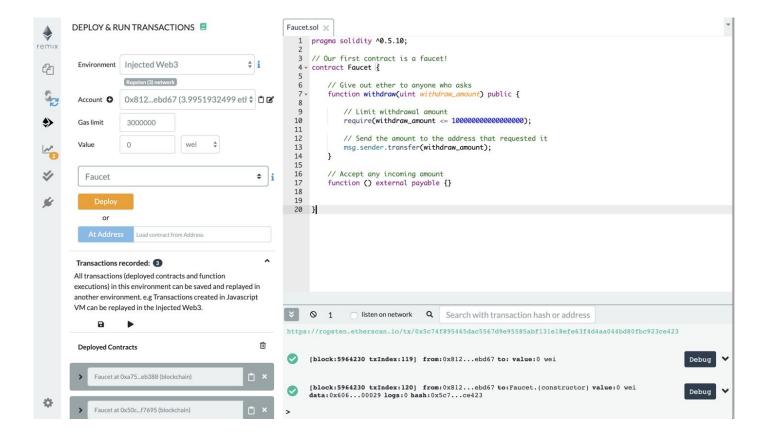
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0xE7 DUP1 PUSH2 0x1F PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x1C JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x2E1A7D4D EQ PUSH1 0x1E JUMPI JUMPDEST, STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH1 0x29 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x53 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO PUSH1 0x3E JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP PUSH1 0x55 JUMP JUMPDEST STOP JUMPDEST PUSH8 0x16345785D8A0000 DUP2 GT ISZERO PUSH1 0x69 JUMPI PUSH1 0x0 DUP1 0x8FC DUP3 SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO DUP1 ISZERO PUSH1 0xAE JUMPI RETURNDATASIZE PUSH1 0x0 DUP1 RETURNDATACOPY RETURNDATASIZE PUSH1 0x0 REVERT JUMPDEST POP POP JUMP INVALID LOG2 PUSH6 0x627A7A723058 KECCAK256 DUP12 SDIV DUP4 0x2d 0xc5 ADDMOD MOD BYTE 0xa7 PUSH13 0xF3B7C3499097383D5B9E9C83BD PUSH24 0x7CE0352811C9D3AC64736F6C634300050A00320000000000

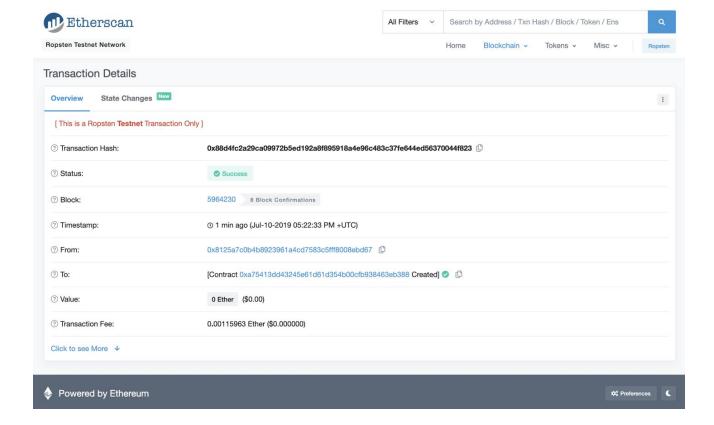
https://remix.ethereum.org

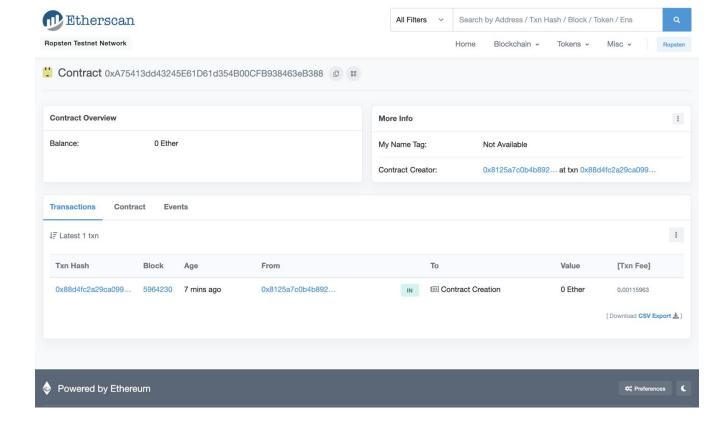
PROF. DANILO CURVELO

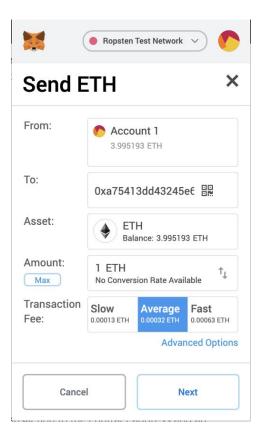


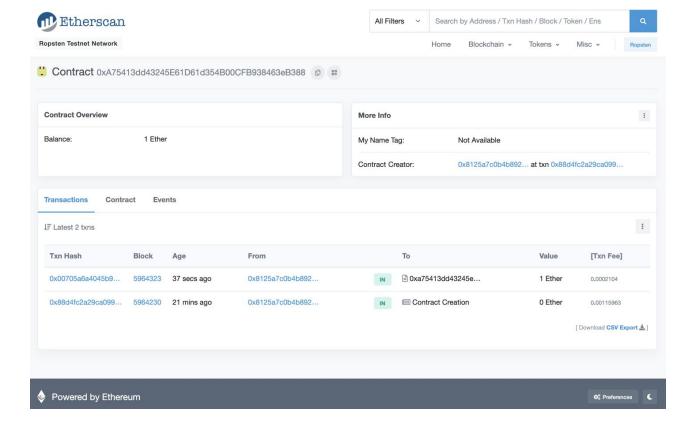






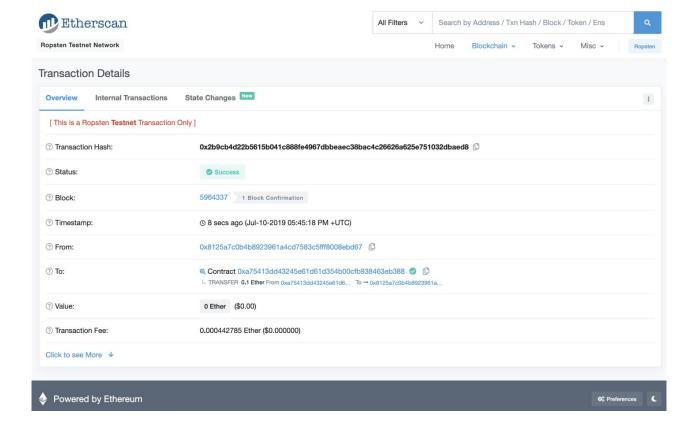




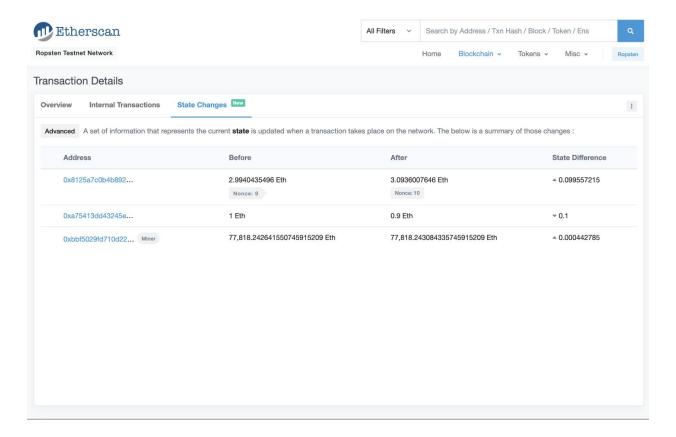


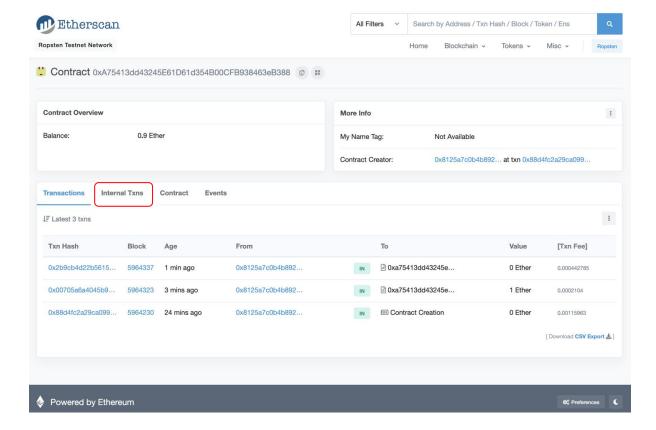
2023.2

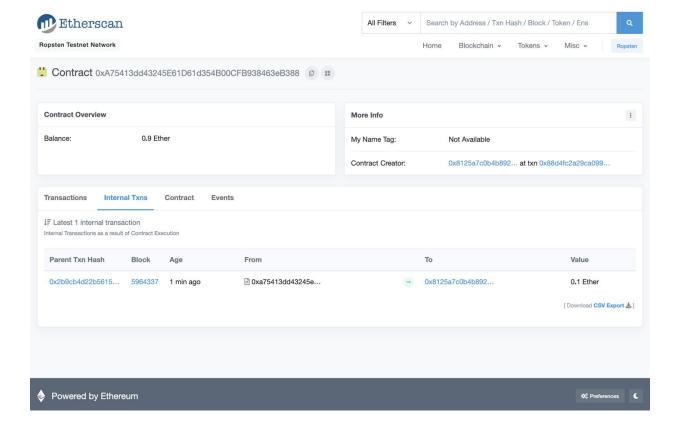




2023.2 Prof. **Danilo Curvelo**







Interface entre dois módulos

Similar a API, porém em linguagem de máquina

No Ethereum, o ABI define as funções em um contrato que podem ser invocadas e descreve como cada função irá receber argumentos e seus retornos

Especificado em JSON

Contrato em Solidity

```
pragma solidity ^0.5.10;
contract Faucet {
    function withdraw(uint withdraw_amount) public {
        require(withdraw amount <= 100000000000000000);</pre>
        msg.sender.transfer(withdraw amount);
    function () external payable {}
```

Contrato em ByteCode

PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0xE7 DUP1 PUSH2 0x1F PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN INVALID PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x1C JUMPI PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x2E1A7D4D EQ PUSH1 0x1E JUMPI JUMPDEST STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH1 0x29 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x53 PUSH1 0x4 DUP1 CALLDATASIZE SUB PUSH1 0x20 DUP2 LT ISZERO PUSH1 0x3E JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST DUP2 ADD SWAP1 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1 SWAP3 SWAP2 SWAP1 POP POP PUSH1 0x55 JUMP JUMPDEST STOP JUMPDEST PUSH8 0x16345785D8A0000 DUP2 GT ISZERO PUSH1 0x69 JUMPI PUSH1 0x0 DUP1 0x8FC DUP3 SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO DUP1 ISZERO PUSH1 0xAE JUMPI RETURNDATASIZE PUSH1 0x0 DUP1 RETURNDATACOPY RETURNDATASIZE PUSH1 0x0 REVERT JUMPDEST POP POP JUMP INVALID LOG2 PUSH6 0x627A7A723058 KECCAK256 DUP12 SDIV DUP4 0x2d 0xc5 ADDMOD MOD BYTE 0xa7 PUSH13 0xF3B7C3499097383D5B9E9C83BD PUSH24 0x7CE0352811C9D3AC64736F6C634300050A003200000000000

ABI do contrato

```
"constant": false,
"inputs": [
              "name": "withdraw_amount",
              "type": "uint256"
"name": "withdraw",
"outputs": [],
"payable": false,
"stateMutability": "nonpayable",
"type": "function"
"payable": true,
"stateMutability": "payable",
"type": "fallback"
```

cryptozombies.io



```
1 pragma solidity >=0.5.0 <0.6.0;</pre>
```

```
1 contract Example {
    uint myUnsignedInteger = 100;
```

```
1 struct Person {
    uint age;
    string name;
```

```
1 // array com comprimento fixo de 2 elementos.
 2 uint[2] fixedArray;
 4 string[5] stringArray;
 6 uint[] dynamicArray;
    Person[] people; // array de structs dinâmico.
10 // você pode declarar seu array com 'public' e o Solidity cria automaticamente o getter.
12 Person[] public people;
```

```
5 function eatHamburgers(string memory _name, uint _amount) public {
  eatHamburgers("daniel", 100);
```

```
1 struct Person {
     uint age;
     string name;
   Person[] public people;
8 // cria uma nova 'Person':
 9 Person satoshi = Person(172, "Satoshi");
12 people.push(satoshi);
   people.push(Person(16, "Vitalik"));
17 uint[] numbers;
18 numbers.push(5);
19 numbers.push(10);
20 numbers.push(15);
```

```
1 // em solidity, funções são públicas por padrão.
 2 // se definido como 'private' somente outras funções dentro do mesmo contrato poderão chamar a função
 3 // por padrão, normalmente iniciamos o nome da função com quando ela é privada
   uint[] numbers;
   function addToArray(uint number) private {
     numbers.push( number);
11 // Para definir um retorno usamos a palavra-chave returns:
12 // modificador 'view': quando somente lemos dados do blockchain, e não escrevemos.
14 string greeting = "Hello!";
16 function sayHello() public view returns (string memory) {
      return greeting;
   // modificador 'pure': quando não precisamos nem ler do blockchain.
21 function multiply(uint a, uint b) private pure returns (uint) {
      return a * b;
```

```
1 uint8 a = 5;
 2 uint b = 6;
 4 uint8 c = a * b;
 6 uint8 c = a * uint8(b);
 8 // o Ethereum tem uma função keccak256 (SHA3). Gera uma hash de 256 bits.
11 keccak256(abi.encodePacked("aaaab")); //6e91ec6b618bb462a4a6ee5aa2cb0e9cf30f7a052bb467b0ba58b8748c00d2e5
12 keccak256(abi.encodePacked("aaaac")); //b1f078126895a1424524de5321b339ab00408010b7cf0e6ed451514981e58aa9
14 // Gerando números aleatórios entre 1 e 100:
    uint random = uint(keccak256(abi.encodePacked(now, msg.sender))) % 100;
```

```
2 event IntegersAdded(uint x, uint y, uint result);
4 function add(uint _x, uint _y) public returns (uint) {
    uint result = _x + _y;
    emit IntegersAdded(_x, _y, result);
    return result;
```

```
1 // se por exemplo, quisermos armazenar o saldo de um usuário no nosso app.
 2 mapping (address => uint) public accountBalance;
 3 // ou se precisamos armazenar usuarios baseados em seus ids.
 4 mapping (uint => string) userIdToName;
 6 // utilizando msg.sender
   mapping (address => uint) favoriteNumber;
10 function setMyNumber(uint _myNumber) public {
     // atualiza o mapping `favoriteNumber` para armazenar `myNumber` na chave `msg.sender`
     favoriteNumber[msg.sender] = myNumber;
     // ^ A sintaxe de atribuir valores em um mapping é similar a arrays.
16 function whatIsMyNumber() public view returns (uint) {
     // retorna o valor armazenado na chave 'endereço do remetente'
     // será `0` se o remetente ainda não chamou `setMyNumber`
      return favoriteNumber[msg.sender];
```

```
1 function sayHi(string memory _name) public returns (string memory) {
    // Compara se name é igual a "Ethereum". Lança um erro e a execução para se não for verdade.
    // Obs: Solidity não tem comparação de strings nativo, então comparamos a hash de cada string
    require(keccak256(abi.encodePacked(_name)) == keccak256(abi.encodePacked("Ethereum")));
    // se verdadeiro, continua a execução
```

```
1 contract Doge {
    function catchphrase() public returns (string memory) {
      return "So Wow CryptoDoge";
7 contract BabyDoge is Doge {
    function anotherCatchphrase() public returns (string memory) {
      return "Such Moon BabyDoge";
```

```
3 contract newContract is SomeOtherContract {
```

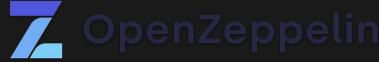
```
1 contract SandwichFactory {
     struct Sandwich {
      string name;
      string status;
    Sandwich[] sandwiches;
    function eatSandwich(uint index) public {
      // ^ parece bem direto, mas o Solidity vai ter um warning informando que
      // portanto você deve declarar com a palavra-chave `storage`:
      Sandwich storage mySandwich = sandwiches[ index];
      // ...onde no caso `mySandwich` é um ponteiro para `sandwiches[ index]
      // em storage, e...
      mySandwich.status = "Eaten!";
      // ...isso irá permanentemente modificar `sandwiches[ index]` no blockchain.
      // se você só quer uma cópia, você pode usar `memory`:
      Sandwich memory anotherSandwich = sandwiches[ index + 1];
      // ...que no caso `anotherSandwich` irá simplemente ser uma cópia do valor
      anotherSandwich.status = "Eaten!";
      // ...irá modificar somente a variável temporária e não terá efeito em
      // `sandwiches[ index + 1]`. Mas você pode fazer isso:
      sandwiches[ index + 1] = anotherSandwich;
```

```
1 contract Sandwich {
    uint private sandwichesEaten = 0;
    function eat() internal {
      sandwichesEaten++;
   contract BLT is Sandwich {
    uint private baconSandwichesEaten = 0;
    function eatWithBacon() public returns (string memory) {
      baconSandwichesEaten++;
      eat();
```

```
1 // contrato implantado no blockchain no endereço de contrato
2 // 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d
  contract LuckyNumber {
     mapping(address => uint) numbers;
     function setNum(uint num) public {
      numbers[msg.sender] = num;
     function getNum(address myAddress) public view returns (uint) {
       return numbers[ myAddress];
```

```
1 contract NumberInterface {
     function getNum(address myAddress) public view returns (uint);
5 contract MyContract {
     address NumberInterfaceAddress = 0x06012c8cf...;
     // ^ o endereço do contrato LuckyNumber no Ethereum
     NumberInterface numberContract =
   NumberInterface(NumberInterfaceAddress);
      function someFunction() public {
       // Agora podemos chamar `getNum` do outro contrato:
       uint num = numberContract.getNum(msg.sender);
17 }
```

```
contract Ownable {
  address private _owner;
  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
  constructor() internal {
    owner = msg.sender;
    emit OwnershipTransferred(address(0), owner);
  function owner() public view returns(address) {
    return owner;
  modifier onlyOwner() {
    require(isOwner());
  function isOwner() public view returns(bool) {
    return msg.sender == _owner;
  function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
  function transferOwnership(address newOwner) public onlyOwner {
    _transferOwnership(newOwner);
  function transferOwnership(address newOwner) internal {
    require(newOwner != address(0));
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
```



https://openzeppelin.com/

```
pragma solidity >=0.5.0 <0.6.0;</pre>
import "./ownable.sol";
contract ZombieFactory is Ownable {
    Zombie[] public zombies;
contract KittyInterface {
  function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
contract ZombieFeeding is ZombieFactory {
  KittyInterface kittyContract;
  function setKittyContractAddress(address address) external onlyOwner {
    kittyContract = KittyInterface(_address);
```

```
1 // um mapping para armazenar a idade do usuário:
 2 mapping (uint => uint) public age;
 5 modifier olderThan(uint _age, uint _userId) {
     require(age[_userId] >= _age);
12 function driveCar(uint _userId) public olderThan(18, _userId) {
     // a lógica da função
    function test() external view onlyOwner anotherModifier { /* ... */ }
```

```
1 uint lastUpdated;
 4 function updateTimestamp() public {
     lastUpdated = now;
10 function fiveMinutesHavePassed() public view returns (bool) {
     return (now >= (lastUpdated + 5 minutes));
```

```
1 contract OnlineStore {
     function buySomething() external payable {
      require(msg.value == 0.001 ether);
      transferThing(msg.sender);
```

```
1 contract GetPaid is Ownable {
      function withdraw() external onlyOwner {
        address payable _owner = address(uint160(owner()));
_owner.transfer(address(this).balance);
```

Variáveis globais

O objeto **msg** é a chamada da transação (se proveniente de EAO) ou chamada de mensagem (se proveniente de contrato) que lançou a execução do presente contrato

msg.sender representa o endereço que iniciou a chamada ao contrato
 msg.value o valor de ether (em wei) enviado por essa chamada
 msg.data a carga útil de dados dessa chamada para nosso contrato
 msg.sig os primeiros 4 bytes da carga útil de dados, que é o identificador de função

PROF. DANILO CURVELO

Variáveis globais

O objeto tx provê meios para acessar informações relacionadas a transação

tx.gasprice o gasPrice da transação de chamada

tx.origin o endereço da EOA que originou a transação

Variáveis globais

O objeto **block** contém informações sobre o bloco atual:

block.coinbase o endereço do destinatário das taxas e recompensa deste bloco

block.number O número do bloco (altura)

block.timestamp O timestamp do bloco

Funções globais

blockhash(uint blockNumber) returns (bytes32)

gasleft() returns (uint256)

Funções para o tipo address

Objetos do tipo address tem os seguintes membros:

<address>.balance saldo do endereço em wei

<address payable>.transfer(amount) envia amount em wei para o endereço, lança exceção se falhar

<address payable>.send(amount) envia amount em wei para o endereço e retorna false se falhar

PROF. **DANILO CURVELO**

Funções relacionadas ao contrato

Outras funções embutidas notáveis:

this o endereço do contrato em execução

selfdestruct(recipientAddress) Destrói o contrato em questão, enviando os fundos restantes para o endereço passado

```
1 pragma solidity ^0.5.10;
  contract Faucet {
      address payable owner;
      constructor() public {
          owner = msg.sender;
      function destroy() public {
           require(msg.sender == owner);
           selfdestruct(owner);
      function withdraw(uint withdraw_amount) public {
           require(withdraw_amount <= 100000000000000000);
          msg.sender.transfer(withdraw_amount);
      function () external payable {}
```

2022.2

```
1 pragma solidity ^0.5.10;
  contract Faucet {
     address payable owner;
     constructor() public {
         owner = msg.sender;
     function destroy() public onlyOwner {
         selfdestruct(owner);
     modifier onlyOwner {
         require(msg.sender == owner);
     function withdraw(uint withdraw_amount) public {
         msg.sender.transfer(withdraw_amount);
     function () external payable {}
```

2022.2

```
1 pragma solidity ^0.5.10;
    contract owned {
        address payable owner;
        constructor() public {
            owner = msg.sender;
        modifier onlyOwner {
            require(msg.sender == owner);
16 contract mortal is owned {
        function destroy() public onlyOwner {
            selfdestruct(owner);
   contract Faucet is mortal {
        function withdraw(uint withdraw_amount) public {
            require(withdraw_amount <= 0.1 ether);</pre>
            msg.sender.transfer(withdraw_amount);
        function () external payable {}
```

```
pragma solidity ^0.5.10;
    contract owned {
        address payable owner;
        constructor() public {
            owner = msg.sender;
        modifier onlyOwner {
            require(msg.sender == owner, "Somente o dono do contrato pode chamar essa função!");
16 contract mortal is owned {
        function destroy() public onlyOwner {
            selfdestruct(owner);
   contract Faucet is mortal {
        function withdraw(uint withdraw_amount) public {
            require(withdraw_amount <= 0.1 ether, "Quantidade máxima atingida!");</pre>
            msg.sender.transfer(withdraw amount);
        function () external payable {}
```

```
1 pragma solidity ^0.5.10;
  import "./mortal.sol";
  contract Faucet is mortal {
       event Withdrawal(address indexed to, uint amount);
       event Deposit(address indexed from, uint amount);
       function withdraw(uint withdraw_amount) public {
           require(withdraw_amount <= 0.1 ether, "Quantidade máxima atingida!");</pre>
           msg.sender.transfer(withdraw_amount);
           emit Withdrawal(msg.sender, withdraw_amount);
      function () external payable {
           emit Deposit(msg.sender, msg.value);
```



Ropsten Testnet Network

Search by Address / Txn Hash / Block / Token / Ens All Filters Misc v Home Blockchain v Tokens ~





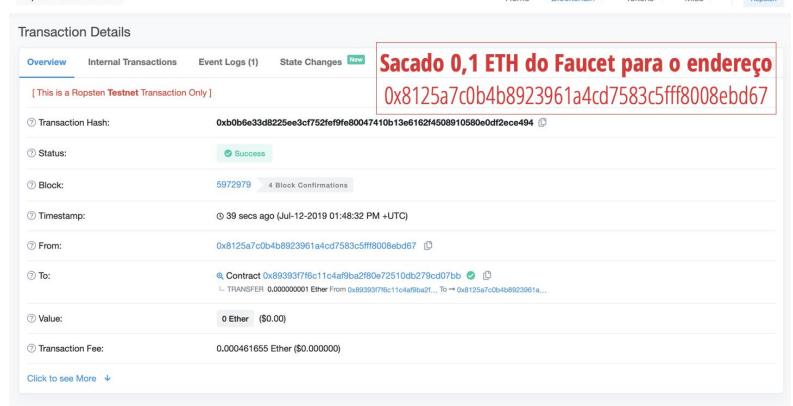
Ropsten Testnet Network

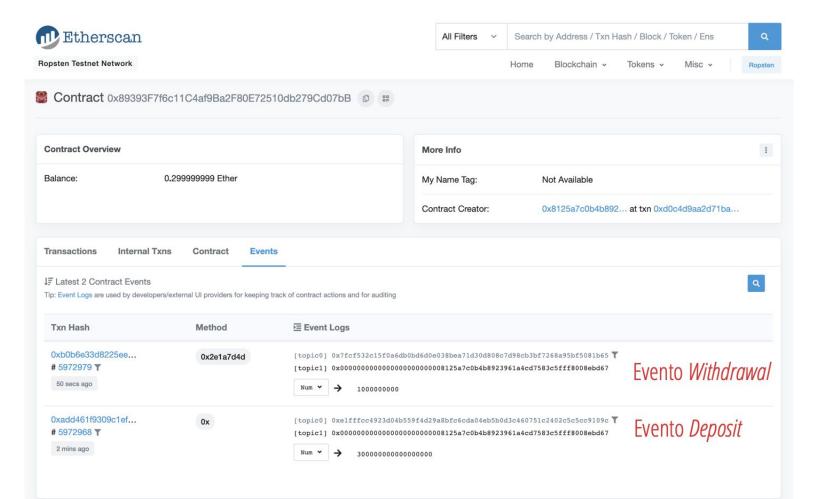
All Filters

Search by Address / Txn Hash / Block / Token / Ens

Q

Home Blockchain Tokens Misc Roosten







Função **destroy()**



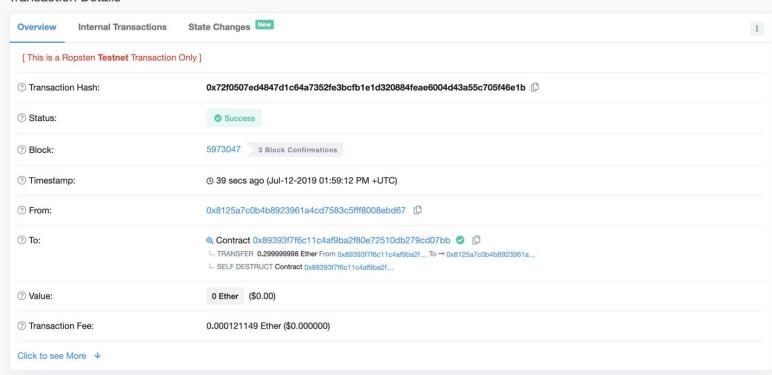
Search by Address / Txn Hash / Block / Token / Ens

Blockchain - Tokens -

Misc v

Ropsten

Transaction Details



All Filters

Home

