

Solidity Development Essentials: Creare Smart Contract Sicuri su Blockchain Pubblica

Impara le basi della programmazione in Solidity per sviluppare smart contract sicuri ed efficienti

Docente

Antonio Seveso

Head of Engineering @ ODF Data



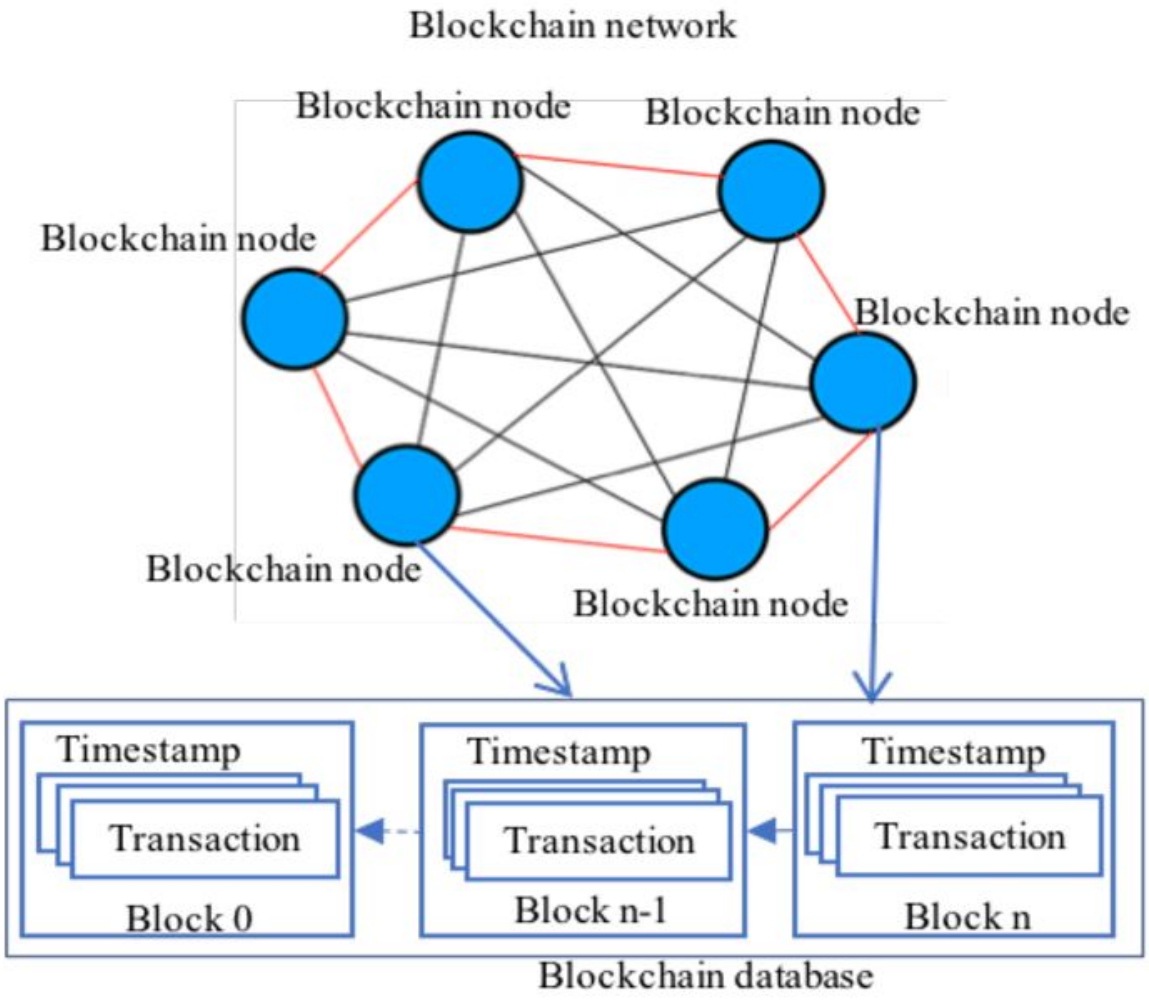
Agenda del corso

- Introduzione allo sviluppo di Solidity
- Creare il primo smart contract
- Smart contract Security

01

Introduzione allo sviluppo Blockchain

Introduzione allo sviluppo Blockchain

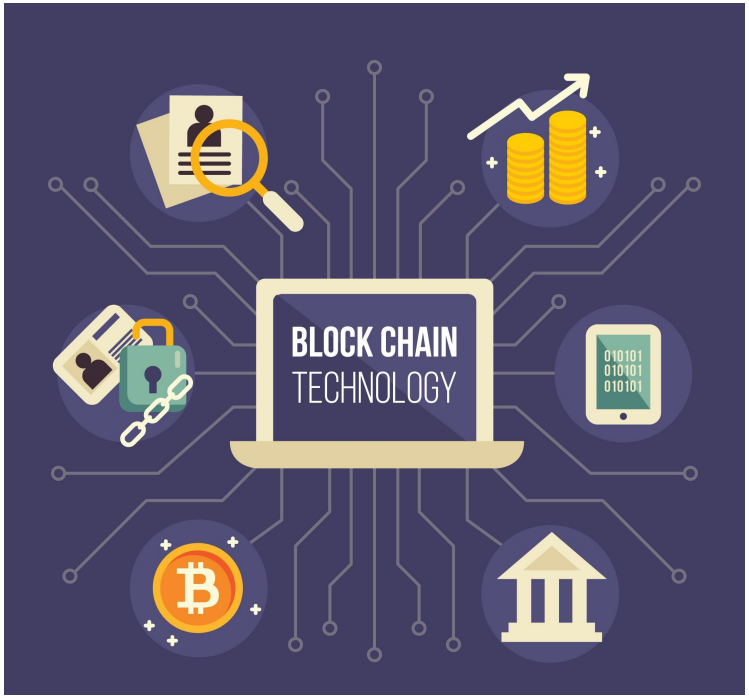


Introduzione allo sviluppo Blockchain

- Cos'è la blockchain?
- Caratteristiche principali
- Decentralizzazione
- Sicurezza e immutabilità

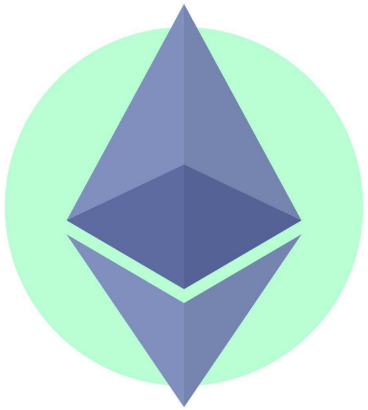
Applicazioni della Blockchain

- Criptovalute (Bitcoin, Ethereum)
- Smart contract
- Registri distribuiti
- Identità digitale



Ethereum e Smart contract

- Cos'è Ethereum?
- Cosa sono gli smart contract?
- EVM ed EVM compatibility
- Esempi di utilizzo degli smart contract



Linguaggi di programmazione

- Solidity
- Vyper
- Altri linguaggi





02

Solidity 101: Comprensione delle basi della programmazione di smart contract

Programmazione di smart contract

```
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint storedData;

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

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

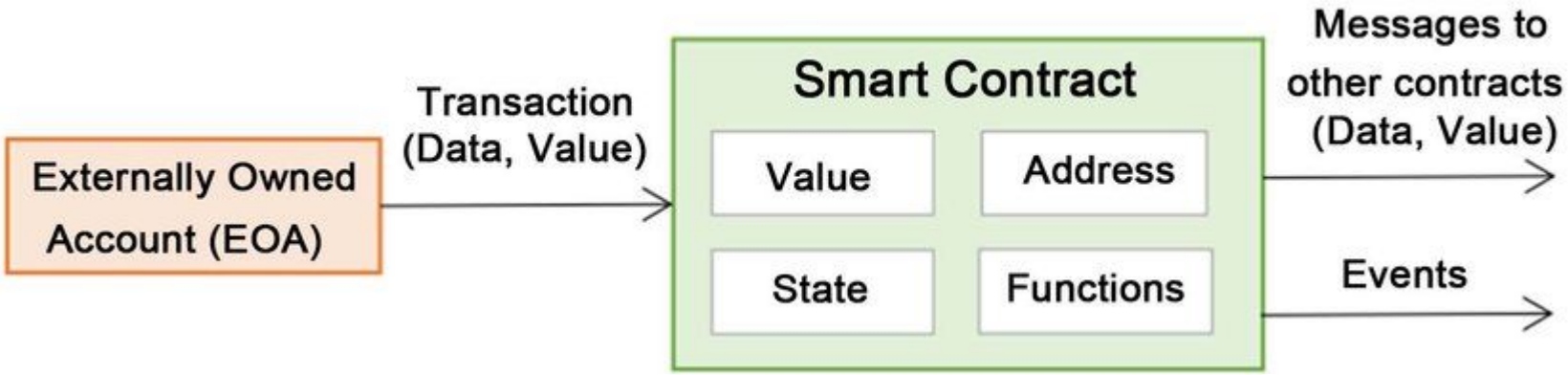
Solidity 101

- Cos'è Solidity?
- Perché è importante?
- Scopi principali
- Linguaggio di programmazione per Ethereum

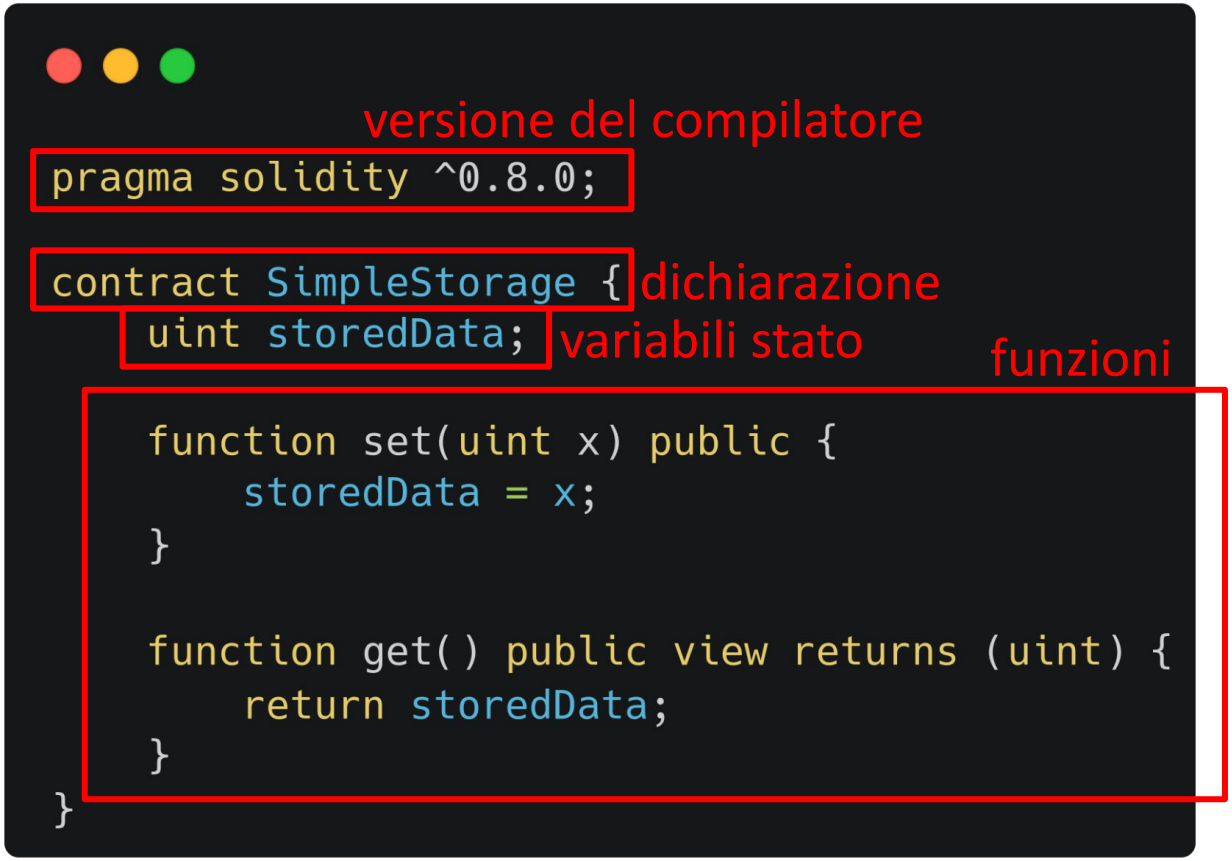


Struttura base di uno smart contract

- Versione del compilatore
- Dichiarazione dello smart contract
- Variabili di stato
- Funzioni

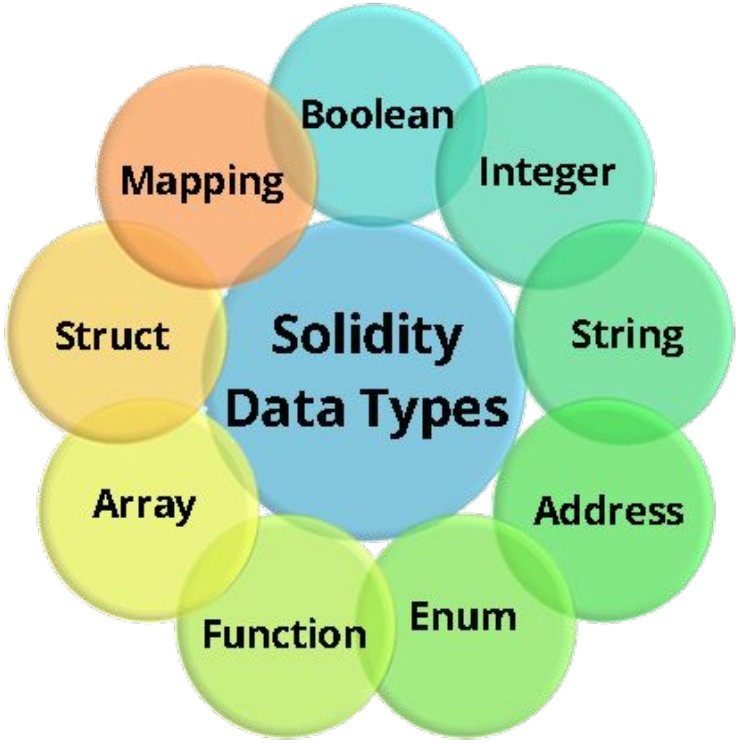


Struttura base di uno smart contract



Tipi di dati in Solidity

- Primitivi (uint, int, bool, address, bytes)
- Stringhe
- Array
- Struct



03

Configurazione dell'ambiente di sviluppo Blockchain

Ambiente di sviluppo Blockchain

- Cos'è un ambiente di sviluppo?
- Perché è importante configurarlo correttamente?
- Componenti essenziali

Strumenti di sviluppo smart contract

- IDE (Remix, Visual Studio Code)
- Framework (Hardhat, Truffle)
- Librerie (Web3.js, ethers.js)
- Testnet (Sepolia, Goerli)

Configurazione di Remix

- Cos'è Remix?
- Come usarlo?
- Esempio di creazione di uno smart contract con Remix

Configurazione di Remix

```
pragma solidity ^0.8.0;

contract Contatore {
    uint private conteggio;

    constructor() {
        conteggio = 0;
    }

    function incrementa() public {
        conteggio += 1;
    }

    function leggiConteggio() public view returns (uint) {
        return conteggio;
    }
}
```

Configurazione di Hardhat



- Cos'è Hardhat?
- Installazione e configurazione
- Esempio di inizializzazione di un progetto Hardhat

```
npm install --save-dev hardhat
npx hardhat
npx hardhat compile
npx hardhat test
npx hardhat run scripts/deploy.js
```

LIVE coding on Remix



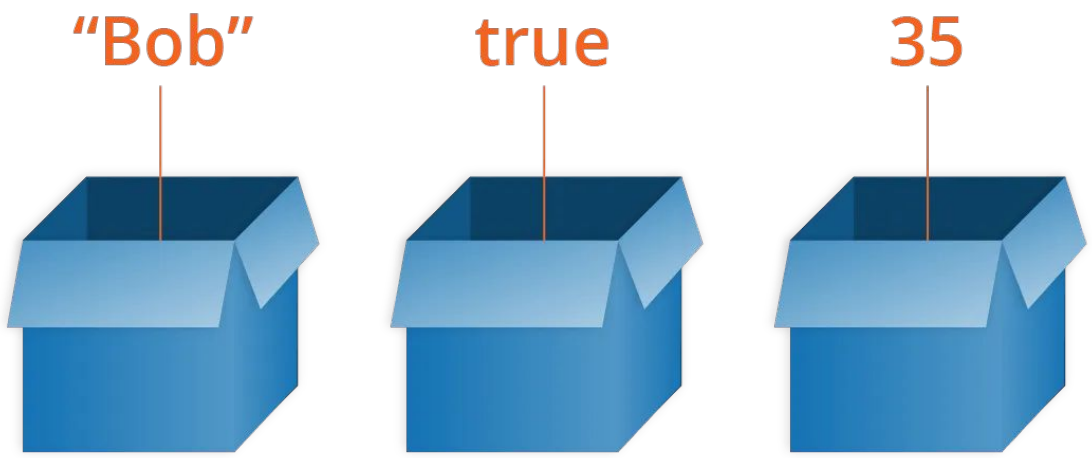
remix.ethereum.org

04

Variabili in Solidity

Variabili in Solidity

- Cos'è una variabile?
- Utilizzo delle variabili nello smart contract
- Assegnazione e modifica dei valori



Tipi di variabili in Solidity

- Variabili di stato
- Variabili locali
- Parametri di funzione
- Variabili globali

Tipi di variabili in Solidity

Variabili di stato e Variabili locali

```
pragma solidity ^0.8.0;

contract VariabiliEsempio {
    uint public statoUint;            // Variabile di stato
    address public statoIndirizzo;    // Variabile di stato

    function esempioFunzione(uint paramUint, address paramAddr) public {
        uint localeUint = paramUint + 1;        // Variabile locale
        address localeIndirizzo = paramAddr; // Variabile locale

        statoUint = localeUint;
        statoIndirizzo = localeIndirizzo;
    }
}
```

Tipi di variabili in Solidity

Parametri di funzione

```
pragma solidity ^0.8.0;

contract ParametriFunzioneEsempio {
    function calcolaSomma(uint a, uint b) public pure returns (uint) {
        uint somma = a + b;
        return somma;
    }
}
```

Tipi di variabili in Solidity

Variabili globali

```
pragma solidity ^0.8.0;

contract VariabiliGlobaliEsempio {
    function informazioniTransazione() public view returns (address, uint) {
        address mittente = msg.sender;
        uint valoreInviato = msg.value;
        return (mittente, valoreInviato);
    }
}
```

Visibilità delle variabili in Solidity

- Private
- Internal
- Public
- External

Visibilità delle variabili in Solidity

```solidity
pragma solidity ^0.8.0;

contract VisibilitaVariabili {
    uint private variabilePrivata;   // accessibile solo all'interno di questo contratto
    uint internal variabileInterna;  // accessibile all'interno di questo contratto e dei derivati
    uint public variabilePubblica;   // accessibile dall'esterno e genera automaticamente un getter
    uint external variabileEsterna;  // Non è possibile dichiarare variabili di stato external

    function esempioFunzione() public {
        variabilePrivata = 1;
        variabileInterna = 2;
        variabilePubblica = 3;
    }

    function esempioFunzioneEsterna(uint _valore) external {
        // Funzione con visibilità esterna, accessibile solo attraverso transazioni esterne
        variabilePubblica = _valore;
    }
}
```

LIVE coding on Remix



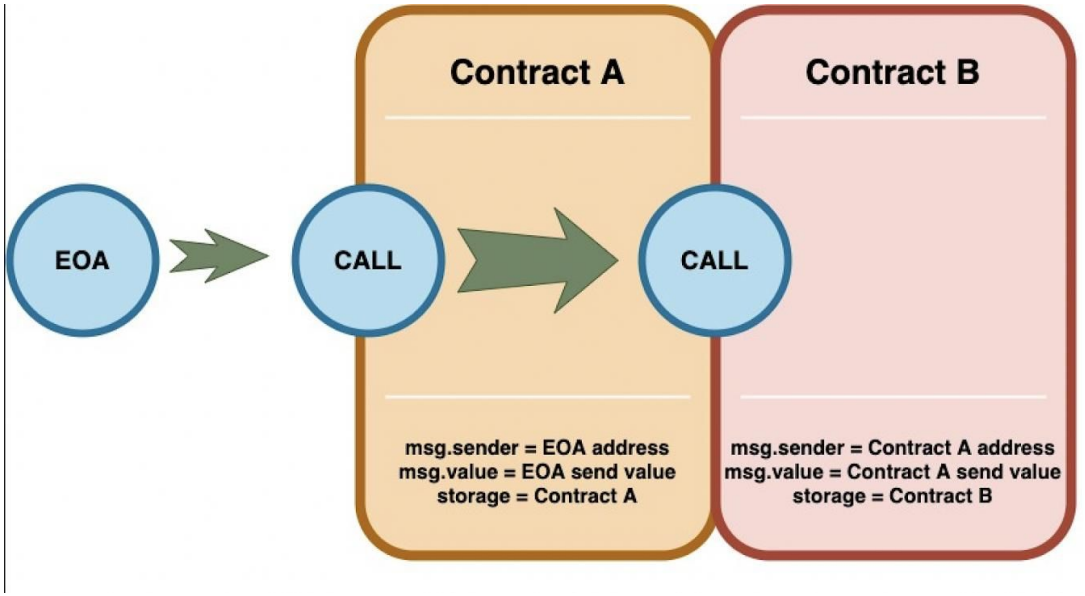
remix.ethereum.org

05

Constructors e Funzioni

Constructors e Funzioni in Solidity

- Cosa sono i constructors?
- Cosa sono le funzioni?
- Utilità dei constructors e delle funzioni nello smart contract



Constructors in Solidity

- Funzione speciale eseguita alla creazione del contratto
- Utilizzata per inizializzare lo stato del contratto
- Può avere modificatori come `public` o `internal`

```
contract Saluto {
    string public saluto;

    constructor(string memory _salutoIniziale) {
        saluto = _salutoIniziale;
    }
```

Funzioni in Solidity

- Le funzioni permettono di eseguire il codice nel contratto
- Tipi di funzioni: `view`, `pure`, `payable`
- Visibilità delle funzioni: `public`, `private`, `internal`, `external`

Funzioni in Solidity

```
pragma solidity ^0.8.0;

contract Saluto {
    string public saluto;

    constructor(string memory _salutoIniziale) {
        saluto = _salutoIniziale;
    }

    function impostaSaluto(string memory _nuovoSaluto) public {
        saluto = _nuovoSaluto;
    }

    function leggiSaluto() public view returns (string memory) {
        return saluto;
    }

    function salutaConNome(string memory _nome, string memory _cognome) public view returns (string memory) {
        return string(abi.encodePacked(saluto, ", ", _nome, " ", _cognome));
    }
}
```

LIVE coding on Remix



remix.ethereum.org

Q & A

Domande?

06

Pubblicazione del tuo primo smart contract su Ethereum

Funzioni in Solidity

- Cos'è la pubblicazione di uno smart contract?
- Utilizzo di Remix per pubblicare uno smart contract su Ethereum



Passaggi per la pubblicazione

1. Creare o importare il codice sorgente dello smart contract (file `.sol`)
2. Compilare lo smart contract
3. Selezionare un ambiente di esecuzione (JavaScript VM, Injected Web3, Web3 Provider)
4. Scegliere il contratto da pubblicare
5. Fornire eventuali parametri del costruttore
6. Pubblicare il contratto e interagire con esso

Esempio di pubblicazione

- Utilizzo dello smart contract `Saluto` dalla lezione precedente
- Pubblicazione del contratto su una rete Ethereum locale
- Interazione con le funzioni del contratto

GO TO



Considerazioni sulla pubblicazione

- Costi del **gas** per la pubblicazione e l'interazione con gli smart contract
- Importanza di **testare** accuratamente il codice del contratto prima della pubblicazione
- Utilizzo di **reti di prova** come Sepolia e Goerli per testare gli smart contract prima di passare alla rete principale (Mainnet)

07

La libreria OpenZeppelin

Introduzione alla libreria OpenZeppelin

- Cos'è OpenZeppelin?
- Standard ERC
- Perché utilizzare OpenZeppelin nello sviluppo di smart contract?
- Componenti principali di OpenZeppelin



Vantaggi di OpenZeppelin

- Modularità
- Sicurezza
- Sviluppo più rapido di smart contract
- Community e supporto



OpenZeppelin

OpenZeppelin Contracts

- Raccolta di smart contract sicuri ed efficienti
- Contratti per token ERC20, ERC721 e altri standard
- Contratti per gestire ruoli, accesso e autorizzazioni
- Supporto per la creazione di meccanismi di voto e governance

Esempio utilizzo Contracts OZ

```
pragma solidity ^0.8.0;

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

contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("MyToken", "MTK") {
        _mint(msg.sender, initialSupply);
    }
}
```

LIVE coding on Remix



remix.ethereum.org

08

Variabili private e pubbliche: gestione e memorizzazione dei dati su Blockchain

Gestione e memorizzazione dei dati

- Variabili private e pubbliche
- Utilizzo delle variabili di stato per memorizzare i dati
- Modificare e accedere ai dati su Blockchain

Variabili private e pubbliche

- Visibilità delle variabili: `private`, `public`
- Funzioni `getter` generate automaticamente per le variabili pubbliche
- Utilizzo delle variabili private per **proteggere i dati sensibili**

Esempio di gestione dei dati

```
pragma solidity ^0.8.0;

contract GestioneDati {
    string private datoPrivato;
    string public datoPubblico;

    function impostaDati(string memory _datoPrivato, string memory _datoPubblico) public {
        datoPrivato = _datoPrivato;
        datoPubblico = _datoPubblico;
    }

    function leggiDatoPrivato() public view returns (string memory) {
        return datoPrivato;
    }
}
```

Esempio di gestione dei dati

- Attenzione alle “false” protezioni
  - Su blockchain è tutto **pubblico**
  - “private” evita la lettura tramite smart contract in modalità non autorizzata
  - **il contenuto della variabile private è SEMPRE leggibile off-chain**

Memorizzazione dei dati

- Dati memorizzati in modo immutabile
- Considerazioni sul costo del gas per la memorizzazione dei dati
- Limitazioni sulla quantità di dati che possono essere memorizzati

09

Costruzione di applicazioni decentralizzate: Il futuro del web

Applicazioni decentralizzate (dApp)

- Cos'è una dApp?
- Caratteristiche delle dApp
- Esempi di dApp nel mondo reale



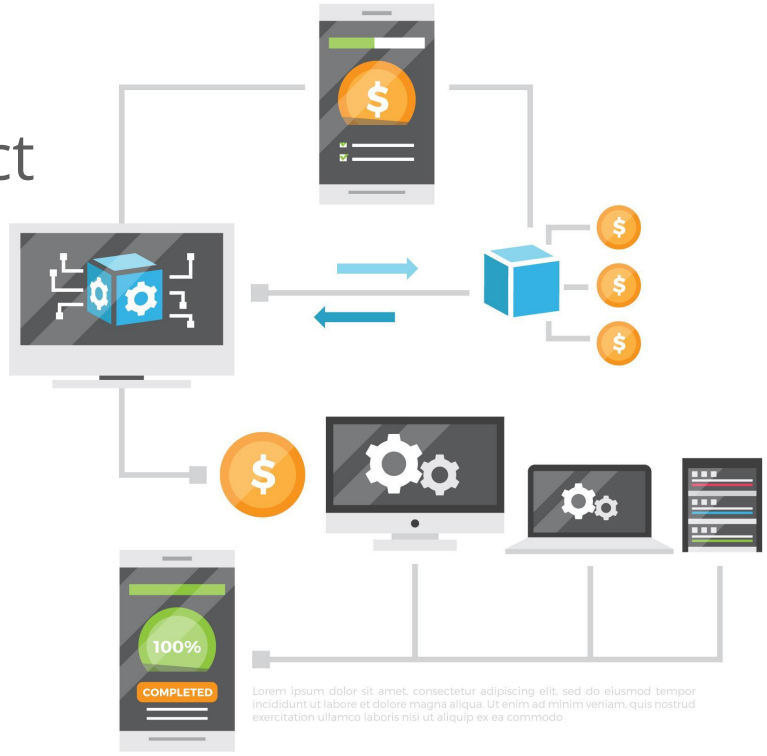
Vantaggi delle dApp

- Resistenza alla censura
- Trasparenza
- Sicurezza
- Tokenizzazione e incentivi



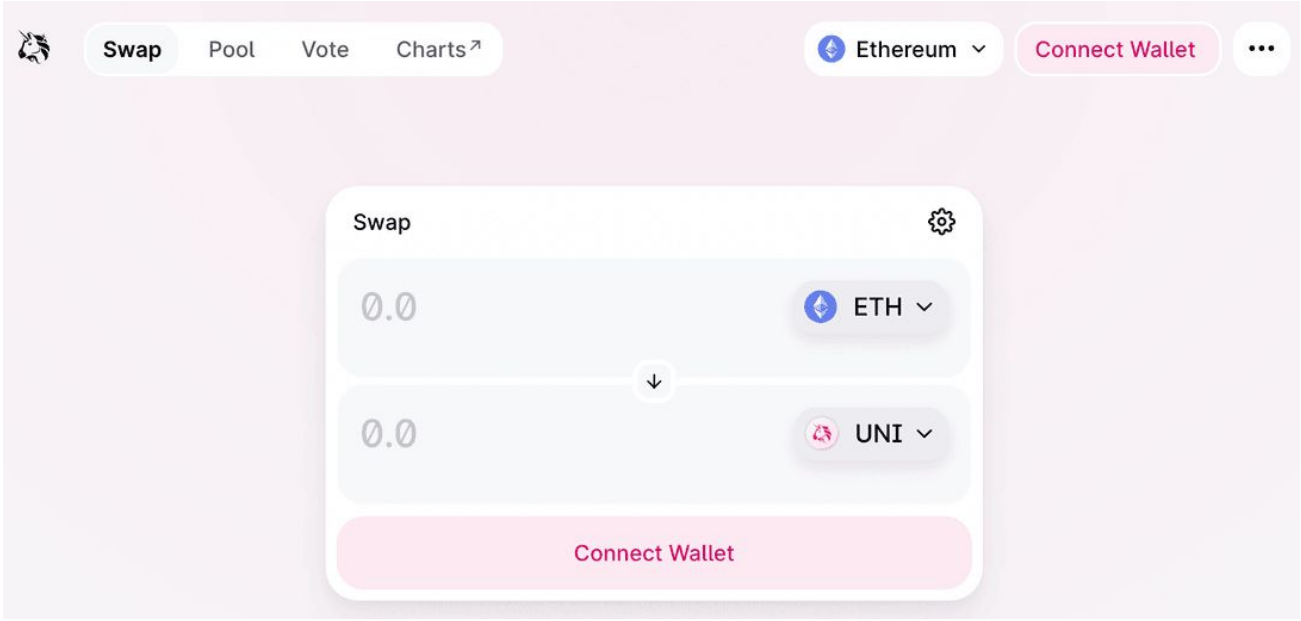
Componenti delle dApp

- **Smart contract** (logica di business)
- **Front-end** (interfaccia utente)
- **Comunicazione** tra front-end e smart contract (Web3.js, Ethers.js, ecc.)
- **Infrastruttura** per l'interazione con la blockchain (Metamask, WalletConnect, ecc.)



Esempio di dApp: Uniswap

- Cos'è Uniswap?
- Come funziona Uniswap
- Interazione tra utenti, smart contract e interfaccia utente



10

Utilizzo degli explorer per interagire con gli smart contract

Gli explorer di blockchain

- Cos'è un explorer di blockchain?
- Utilità degli explorer per monitorare e interagire con la rete
- Esempi di explorer: Etherscan, Etherchain, Blockchair



Funzionalità degli explorer

- Ricerca di indirizzi, transazioni e blocchi
- Informazioni sulle transazioni, come gas utilizzato e costo del gas
- Visualizzazione e verifica del codice degli smart contract
- Interazione con gli smart contract pubblicati e verificati

Esempio di utilizzo di Etherscan

- Etherscan come esempio di explorer di Ethereum
- Ricerca di un indirizzo o uno smart contract
- Visualizzazione delle informazioni e delle transazioni correlate
- Interazione con gli smart contract tramite l'interfaccia "Read Contract" e "Write Contract"

Considerazioni su utilizzo explorer

- Importanza di **verificare le informazioni** sugli smart contract
- **Uso responsabile** degli explorer per evitare abusi e rispettare la privacy degli altri utenti
- Affidarsi a **più explorer** per ottenere informazioni accurate e aggiornate



Q & A

Domande?

11

I principali rischi di sicurezza per gli smart contract

Sicurezza degli smart contract

- Importanza della sicurezza nello sviluppo di smart contract
- Conseguenze di attacchi e vulnerabilità negli smart contract
- Esempi di incidenti di sicurezza: The DAO hack, Parity wallet

Principali rischi di sicurezza

- Reentrancy
- Overflow e underflow aritmetici
- Accesso non autorizzato
- Abuso di delegatecall
- Problemi di visibilità delle funzioni e delle variabili

Misure di prevenzione e good practice

- Utilizzo di librerie testate come OpenZeppelin
- Implementazione di pattern di sicurezza (es. checks-effects-interactions)
- Auditing del codice e revisione da parte di esperti
- Test e verifica formale del codice

Risorse per la sicurezza

- Documentazione ufficiale di Solidity
- Smart Contract Weakness Classification (SWC) Registry
- Blog e community di sicurezza, come ConsenSys Diligence e Trail of Bits
- Conferenze e workshop sulla sicurezza degli smart contract



Q & A

Domande?

CONTATTI



Antonio Seveso

as@odfdata.com

www.odfdata.com