Your First Smart Contract: Every Concept Explained

What's really happening when you deploy code to a blockchain?

Prof. Dr. Jörg Osterrieder

BSc Blockchain, Crypto Economy & NFTs

Spring 2026

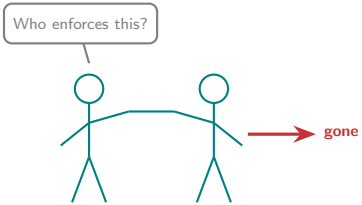
What If Code Could Enforce a Deal?

Situation: You shake hands on a deal — pay me €50 and I'll deliver the goods on Friday.

Complication: One party walks away. A lawyer costs more than the deal itself. Small-value agreements have *no practical enforcement*.

Question: What if a program could hold the money and release it *only* when conditions are met — automatically, with no intermediary?

That program is a **smart contract**: self-executing code on a blockchain. It is **immutable** (cannot be changed after deployment) and **permissionless** (anyone can deploy one).



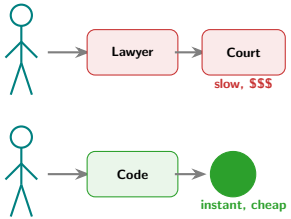
Code can't walk away.

This is the problem smart contracts solve.

Traditional Contract vs Smart Contract

Dimension	Traditional	Smart Contract
Enforcement	Courts, lawyers	Automatic (code)
Speed	Days to weeks	Seconds
Cost	Fees, intermediaries	Gas only
Trust	Trusted third party	Trust the code
Transparency	Private documents	Public on-chain

A smart contract is a **program stored on the blockchain** that runs automatically when conditions are met.



Ethereum pioneered general-purpose smart contracts in 2015.

Your Blockchain Identity: Wallets & Keys

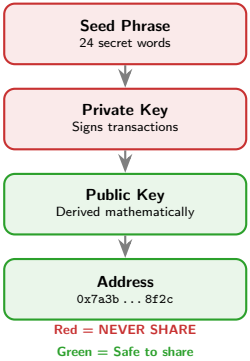
Private key = your signature. Whoever holds it controls the funds.

Seed phrase = 24 words that generate all your private keys. One backup to rule them all.

Public key / address = your identity on the blockchain. Safe to share — like an email address.

MetaMask stores your private key locally in your browser. It signs transactions on your behalf.

Rule: Never type your seed phrase into a website. Never share your private key. If someone has either, they own your funds.



MetaMask stores your private key locally. Whoever has the key controls the funds.

The Playground: Testnets

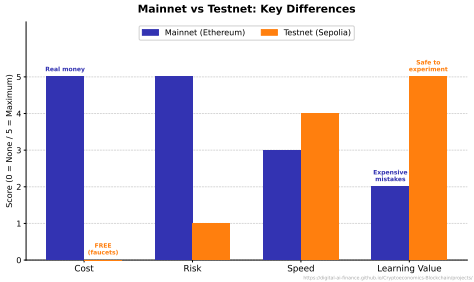
Mainnet = the real Ethereum blockchain. Every transaction uses real ETH with real monetary value.

Testnets = identical copies of Ethereum where the ETH is free and worthless. Same rules, zero risk.

Faucets = websites that give you free test ETH. Just paste your address and click "request."

Sepolia is Ethereum's primary testnet for application developers. It mirrors mainnet's consensus and gas mechanics.

Think of it like a flight simulator: identical controls, no passengers at risk.

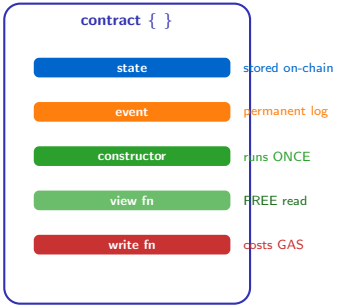


Testnets are identical to mainnet except the ETH has no monetary value.

Solidity in 60 Seconds

Minimal contract anatomy:

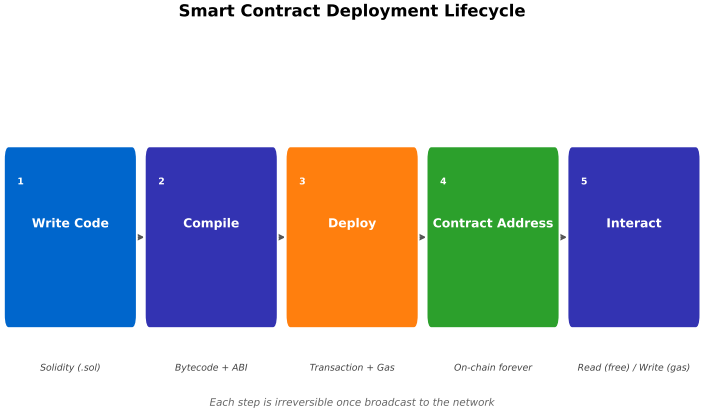
```
contract HelloWorld {
  string public greeting;   // state
  event GreetingChanged(...); // log
  constructor(...)  { ... }   // once
  function get() view { ... } // free
  function set(...)  { ... }    // gas
}
```



Key insight: `view` = free, state change = gas.

Write in Remix IDE. SPDX = open-source license header. `pragma` = minimum compiler version.

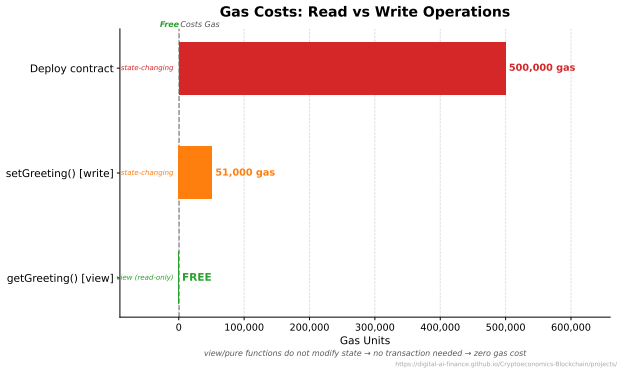
From Code to Blockchain: The Deployment Journey



1. Compilation in **Remix** (browser IDE) produces **bytecode** (machine code for the **EVM**) and an **ABI** (Application Binary Interface — tells wallets which functions exist)
2. Deployment is **irreversible** — once broadcast, the code is permanent and immutable

Deployment is a one-time transaction. Once deployed, the code cannot be changed.

Read vs Write: Why Some Calls Cost Money



View functions (`getGreeting`):

Read your local node's copy. No network broadcast, no miner consensus needed.

Cost: FREE

State changes (`setGreeting`):

Broadcast to all nodes. Validators must agree on the new state before it becomes permanent.

Cost: Gas fee

Reading is free because no validators need to agree. Writing costs gas because **every node must update**.

Gas prices fluctuate with network demand. Testnets use free gas.

The Transparent Ledger: Events & Block Explorers

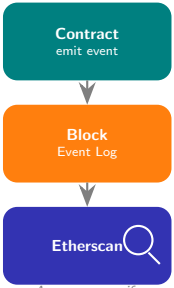
Events = permanent logs emitted by your contract. They record *what happened* without storing full data on-chain.

Blocks group transactions together. Every state change and its event log lives inside a block forever.

Block explorers (Etherscan) let anyone inspect any transaction, any contract, any event — in real time.

Contract verification = publishing your source code so the public can read exactly what the contract does.

This radical transparency is what makes blockchains trustworthy without requiring trust in any person.



Anyone can verify what a contract does by reading its events on Etherscan.

HelloWorld: The Key Lines

```
pragma solidity ^0.8.20;

contract HelloWorld {
    string public greeting;
    event GreetingChanged(
        string old, string new_);
    constructor(string memory _g) {
        greeting = _g;
    }
    function getGreeting()
        public view returns (string
        memory) { return greeting; }
    function setGreeting(string
        memory _new) public {
        emit GreetingChanged(
            greeting, _new);
        greeting = _new;
    }
}
```

leftm1rg1n=*, 1temsep=1pt `pragma` — only compile with Solidity 0.8.20+

leftm2rg2n=*, 2temsep=2pt `contract` — declares a new program on-chain

leftm3rg3n=*, 3temsep=3pt `string public` — state variable; Solidity auto-generates a getter

leftm4rg4n=*, 4temsep=4pt `event` — defines a log schema; cheap to emit, permanent to read

leftm5rg5n=*, 5temsep=5pt `constructor` — runs exactly once when the contract is deployed

leftm6rg6n=*, 6temsep=6pt `memory` — data lives in RAM only during the function call

leftm7rg7n=*, 7temsep=7pt `view` — promises not to modify state (so no gas needed)

leftm8rg8n=*, 8temsep=8pt `emit` — writes event log; `msg.sender` = caller's address

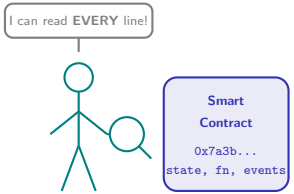
Pattern: constructor sets initial state, `view` reads for free, non-view writes cost gas and emit events.

Full contract in workshop notebook: projects/notebooks/05_first_contract.ipynb

Five Questions to Evaluate Any Smart Contract

1. What **state** does it store? *(and who can change it?)*
2. What **functions** does it expose? *(read vs. write?)*
3. What **events** does it emit? *(what can you track?)*
4. How much **gas** does it cost? *(deployment + usage)*
5. Is it **verified** on a block explorer?

These five questions work for *any* contract — from a simple HelloWorld to a billion-dollar DeFi protocol. Master them and you can audit anything.



Transparency is the superpower.

Next workshop: Create Cryptocurrency (W02) — build your own ERC-20 token.