Create Your Own Cryptocurrency: Every Concept Explained

What makes a token a token, and why does design matter more than code?

Prof. Dr. Jörg Osterrieder

BSc Blockchain, Crypto Economy & NFTs

Spring 2026

What If You Could Print Your Own Money?

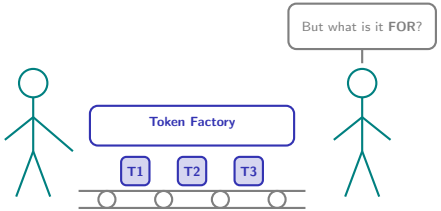
Situation: Governments create currencies and control their supply through central banks and monetary policy.

Complication: On Ethereum, anyone can create a new currency in 10 lines of code. Over 1 million tokens exist — most are worthless.

Key distinction: A **coin** (BTC, ETH) runs its own blockchain. A **token** (USDC, UNI) runs on someone else's blockchain.

Question: What makes a token worth something?

This workshop teaches you how to create an ERC-20 token. But the **real lesson** is understanding *why* each design choice matters.



Creating a token is easy. Creating a valuable token is the real challenge.

What Makes a Token a Token? The ERC-20 Standard

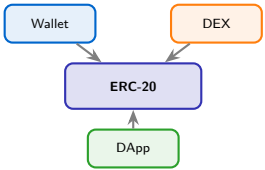
ERC-20 = Ethereum Request for Comments #20. A *standard interface* that every token must implement.

Fungible = every token is identical and interchangeable (like dollars). 1 USDC = 1 USDC, always.

Why standards matter: Wallets, exchanges, and DeFi protocols all expect the same 6 functions. Without a standard, every token would need custom integration.

Plus 2 mandatory events: `Transfer` and `Approval` — every move is permanently logged on-chain.

Function	What It Does
`totalSupply()`	How many tokens exist?
`balanceOf(addr)`	How many do I have?
`transfer(to, amt)`	Send tokens directly
`approve(spdr, amt)`	Allow someone to spend
`allowance(own, spdr)`	Check approved amount
`transferFrom(f,t,a)`	Spend on behalf



One standard, every tool works

ERC-20 was proposed by Fabian Vogelsteller in 2015. Over 500,000 tokens implement it.

Two Ways to Send Tokens

Pattern 1 — Direct Transfer:

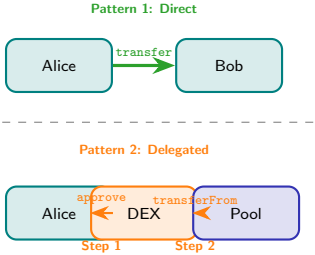
Alice calls `transfer(Bob, 100)`. Tokens move immediately from Alice to Bob. Simple, but limited.

Pattern 2 — Approve + TransferFrom:

Alice calls `approve(DEX, 100)`, then the DEX calls `transferFrom(Alice, Pool, 100)`.

This two-step pattern enables DeFi: smart contracts can move tokens *on your behalf*, but only up to the approved amount.

Without approve+transferFrom, there would be no Uniswap, no Aave, no lending protocols.



Approve+TransferFrom is what makes DeFi possible. Without it, no Uniswap, no Aave, no lending.

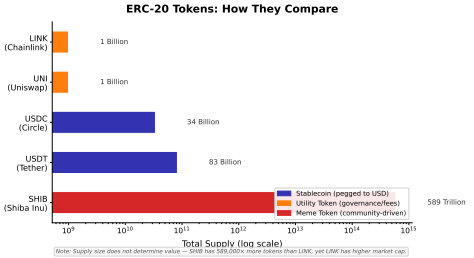
Designing Your Token: Four Decisions

Every token needs exactly 4 parameters:

1. **Name**: Human-readable (e.g., “Student Coin”)
2. **Symbol**: Ticker, 3–5 chars (e.g., “STU”)
3. **Decimals**: Precision level
   18 = standard, 6 = stablecoin, 0 = NFT-like
4. **Initial Supply**: How many tokens at launch

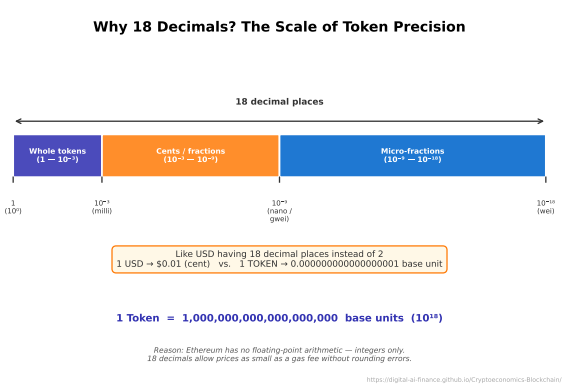
Look at how real tokens made these choices — supply size alone does not determine value.

SHIB has 589 trillion tokens; LINK has 1 billion. Both trade on every major exchange.



Supply doesn’t determine value. SHIB has 589 trillion tokens; LINK has 1 billion. Both are viable.

Why 18 Decimals? The Precision Problem



The problem: Solidity has no floating point — only integers.

The solution: Store everything with 18 decimal places of precision.

1 token = 1,000,000,000,000,000,000 base units
(like cents, but 10^{18} of them)

Formula:
Raw = Human $\times$ 10^{18}

Common mistake: Sending 100 instead of 100 * 10**18 — you'd send 0.000000000000000001 tokens.

The decimal system is a UI illusion. On-chain, there are only integers. Wallets divide by 10^{18} to show "human" amounts.

ETH uses 18 decimals too. USDC uses only 6. This is a design choice, not a rule.

Standing on Giants: OpenZeppelin

OpenZeppelin = battle-tested, audited smart contract library used by most production tokens.

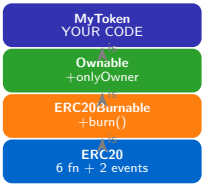
Instead of writing 200 lines, you write:

```
import "@openzeppelin/.../ERC20.sol";
contract MyToken is ERC20 { ... }
```

The keyword `is` means **inheritance**: your contract gets ALL ERC-20 functions for free.

You only write the custom logic. OpenZeppelin handles the standard plumbing.

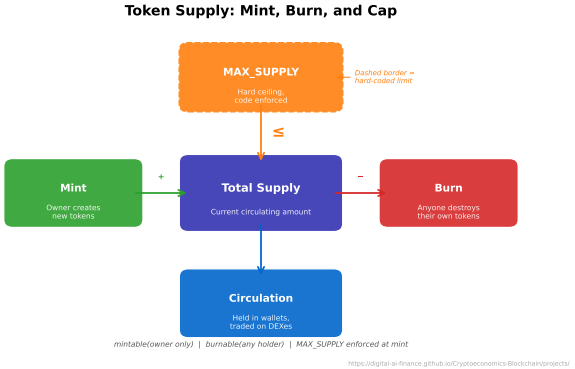
You Write	You Get Free
`is ERC20`	6 functions + 2 events
`is ERC20Burnable`	`burn()` function
`is Ownable`	`onlyOwner` modifier
`_mint(...)`	Token creation logic



Audited by top firms, trusted by billions in TVL.

OpenZeppelin contracts have been audited by the best security firms in blockchain.

Supply Rules: Mint, Burn, and Cap



- **Mint** = create new tokens; owner (`msg.sender` at deploy) via `onlyOwner`
- **MAX_SUPPLY** enforced by code: `require(totalSupply() + amt <= MAX_SUPPLY)`
- **Burn** = destroy tokens permanently (anyone can burn their own)
- **Ownable** = access control restricting sensitive functions to deployer

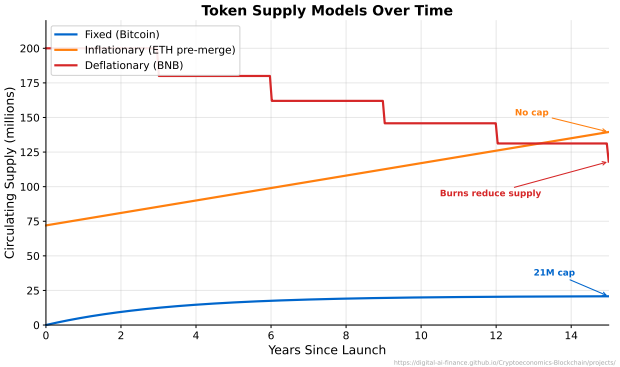
Bitcoin's 21M cap is a supply rule. Advanced: `airdrop()` **sends tokens to many addresses at once.**

From Code to Wallet: The Full Journey

Example: Deploy "Student Coin" (STU) to Sepolia — the `constructor` sets name, symbol, and mints 1,000,000 tokens to your address (`msg.sender`). **ABI** = Application Binary Interface (the contract's API).

All 5 steps take under 30 minutes on a testnet. Mainnet deployment costs real ETH.

Token Economics: The Hard Part



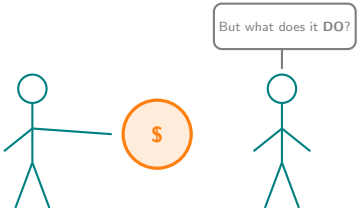
The cryptoeconomics lens applied to YOUR token: (1) What **problem** does it solve? (2) Why would people **hold** it? (3) What are the **costs/benefits**? (4) What if nobody wants it? — Three utility types: governance (UNI, AAVE), payment (LINK, FIL), staking (ETH, SOL).

Technology is easy; economics is hard. Code takes 30 minutes. Tokenomics takes months.

Six Questions to Evaluate Any Token

1. What **standard** does it follow? (ERC-20? ERC-721? Custom?)
2. Who controls the **supply**? (Fixed? Mintable? By whom?)
3. What **utility** does it provide? (Governance? Payment? Access?)
4. How are tokens **distributed**? (Fair launch? Team allocation? Vesting?)
5. What are the **decimals**? (18? 6? 0?)
6. Is the contract **verified** and audited?

These six questions work for *any* token — from a student project to a billion-dollar protocol. Master them before creating or investing in any token.



No utility = no value.

Related course content: L17 Token Standards, L19 Token Design, L29 Tokenomics.