Token standards

ERC-20, and a token of your own · Session 12 · 12 October 2026

Where we left off

- A **token** is a contract that keeps a table of balances and lets holders move them.
- We wrote one, and passed its 1,000 units around the class.
- A **stablecoin** is a token whose issuer promises to redeem each unit for a dollar.

```
mapping(address => uint256) public balances;

function transfer(address _dest, uint256 _amount) external { … }
```

What was awkward about our token

What we tried	What happened
See it in MetaMask	MetaMask did not know it was a token
Look it up on Blockscout	a contract like any other: no holders, no token transfers
Check a balance	*Read contract* → `balances` → paste the address
Send some to a classmate	*Write contract* → `transfer`

Why can’t MetaMask show our tokens, and send them with its *Send* button?

What would a wallet need to know?

To show and move **any** token, a wallet must know, for every contract:

- how to ask for a **balance**: `balances(addr)`? `amounts(addr)`? `getBalance(addr)`?
- how to **send**: `transfer`? `send`? With which arguments, in which order?
- how many **decimals** to display;
- how to learn that a transfer **happened**, without re-reading every transaction.

Nothing in our code was wrong: it was just different from everybody else's.

A standard is an agreed interface

- An **IBAN** lets any bank send money to any other.
- A token standard fixes the **names**, **arguments** and **behavior** of a few functions, and the **events** a token must emit. The rest of the contract is free.

Proposal	Changes	Example
EIP	the protocol: every node must upgrade	EIP-1559, fee burn
ERC	a convention for applications only	ERC-20, tokens

ERC-20

Proposed in November 2015 (EIP-20, Fabian Vogelsteller and Vitalik Buterin).

Function or event	Meaning
`name()`, `symbol()`, `decimals()`	how to display it
`totalSupply()`	units in existence
`balanceOf(owner)`	units held by an address
`transfer(to, value)`	send your own units
`approve(spender, value)` · `allowance(owner, spender)`	let someone else move your units
`transferFrom(from, to, value)`	move units you were allowed to move
event `Transfer` · event `Approval`	public logs of transfers and approvals

Decimals

Balances are **integers**. `decimals` only says where to put the point when displaying them.

Stored balance	`decimals`	Displayed
30	0	30
30000	3	30
15000000000000000000	18	1.5

18 is the usual choice, as for ether (1 ETH = 10^{18} wei). Our coffee token had none.

Events

An **event** is a record that a contract writes into the transaction's **log**.

```
event Transfer(address indexed from, address indexed to, uint256 value);

emit Transfer(msg.sender, to, value);
```

- A wallet finds your incoming tokens by searching the log, not by re-reading every transaction.
- Contracts cannot read logs; wallets and explorers can, cheaply.

Writing a token without writing it

OpenZeppelin: libraries of audited, widely used contract components. Don’t write money code from scratch.

wizard.openzeppelin.com → *ERC20*: fill in a form, get the contract.

Setting	Our choice
Name, symbol	CryptoLessonGov, CLG
Decimals	18
Premint	100
Features	Mintable, Permit, Votes

Premint, and premine

Premint: units created when the contract is deployed, as our 1,000 coffee tokens.

	At launch	Afterwards
Bitcoin, 2009	no bitcoin	every bitcoin is a mining reward, same rules for all
Ethereum, 2015	~72 million ether in the first block	new ether as block rewards

About 60 million ether went to the buyers of the 2014 public sale: an **initial coin offering**.

Mint and burn

- **Mintable**: a privileged account can create new units. A stablecoin issuer mints when dollars arrive.
- **Burnable**: units can be destroyed. A redeemed stablecoin must be burned, or it would be backed by nothing.

Does ordinary money get burned? Every day:

	Bank's books
Loan granted	+1,000 in your account, +1,000 debt
Loan repaid	both erased: that money no longer exists

Burning bank money: three caveats

- Only **bank deposits** disappear. Banknotes and central-bank reserves are untouched; the interest becomes the bank's income.
- "Out of thin air", but **not without limits**: capital requirements, profitability and demand for loans constrain lending.
- The closest blockchain analogue is a **crypto-backed** stablecoin (DAI):

	Bank	**DAI**
Borrow	deposit created, against a debt	DAI minted, against locked ETH
Repay	deposit and debt erased	DAI burned, ETH released

Pausable: who holds the power?

A privileged account can **stop every transfer**, and restart them.

- For: an emergency brake if a bug lets someone steal tokens.
- Against: one key that can freeze everyone's money.

Censorship resistance holds at the **protocol** level: nobody can freeze ether. A token has whatever rules its author wrote.

USDC and USDT can pause, and can freeze single addresses.

Permit: tokens without ether

To send a token you need **ether** for the fee. A new user with only stablecoins is stuck.

Permit (EIP-2612): the holder **signs** an authorization; someone else submits it and pays the fee in ether, and is paid back in the token.

The coffee drinker never needs to know what ether is.

Flash minting

Arbitrage: shop A sells gold at 90, shop B buys at 100. You have no money.

In **one** transaction:

1. borrow a billion, no questions asked;
2. buy gold at A, sell it at B;
3. repay the loan with interest, or the whole transaction **reverts**.

The lender bears no default risk: a **flash loan**. A flash-mintable token (ERC-3156) lets the issuer mint any amount, as long as it is burned in the same transaction.

Votes

Vote on the break, weighted by CLG held. The flaw:

- Alice collects everyone’s tokens and votes; passes them to Bob, who votes; and so on.

Votes keeps a history of balances: each vote is weighted by the tokens held at a past block, when the ballot opened.

The code the Wizard wrote

```
contract CryptoLessonGov is ERC20, Ownable, ERC20Permit, ERC20Votes {
    constructor(address recipient, address initialOwner)
        ERC20("CryptoLessonGov", "CLG")
        Ownable(initialOwner)
        ERC20Permit("CryptoLessonGov")
    {
        _mint(recipient, 100 * 10 ** decimals());
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
    // … overrides required by Solidity
}
```

Deploying it

1. *Open in Remix* → compile → environment: MetaMask, Sepolia.
2. The constructor asks for two addresses: **recipient** of the premint, **initialOwner**, who can mint more.
3. Deploy. Blockscout shows all the functions of the standard.

CLG on Blockscout

What the standard buys

Nobody told these tools about CLG in advance:

- **MetaMask**: *Tokens* → *Manage tokens* → *Import*, paste the address. Symbol and decimals are filled in automatically; *Send* works as for ether.
- **Blockscout**: a token page with every **holder** and their share of the supply.
- On any address page, Blockscout lists all the tokens it holds, found through `Transfer` events.

Pass it on

Two achievements: **receive** some CLG, **send** some CLG.

- Fractions are fine: 0.5 CLG is a valid amount.
- Do **not** send tokens to the token’s contract, or to any contract that does not expect them: they are lost.

Your own token

1. **wizard.openzeppelin.com** → *ERC20*, choose name, symbol, premint, features.
2. *Open in Remix*, deploy on Sepolia, set yourself as recipient and owner.
3. Import it in MetaMask.
4. Get it **held by as many people as possible**.

Live: cryptoassets2026-tokens.bruschi.dev

Do all addresses count?

Creating an address is **free**, and takes a second.

- If every address counts, the winner is whoever creates **the most addresses**.
- One person pretending to be many is a **Sybil attack**.
- Real token **airdrops** face exactly this: "airdrop farming" with thousands of wallets.

For our challenge, only addresses in the **class sheet** count.

Next session

Suppose your token is useful: a café accepts it. Can you **sell** it?

- How to find a buyer, and agree on a price, without an intermediary?
- Exchanges, and contracts that trade tokens: why they need `approve` and `transferFrom`.
- The real code of tokens and stablecoins that hold billions.