Workshop: Create Your Own Cryptocurrency

Launch an ERC-20 Token in 30 Minutes

BSc Blockchain, Crypto Economy & NFTs

FS2026

Workshop Overview

What You'll Build:

1. Custom ERC-20 token
2. Deploy to Sepolia testnet
3. Add token to MetaMask
4. Transfer tokens between wallets

Prerequisites:

- Completed "First Smart Contract" workshop
- MetaMask with Sepolia test ETH
- 30 minutes

Related Lessons: W02, L17, L19

By the end, you'll have your own cryptocurrency!

Design Your Token

Choose Your Parameters:

Parameter	Example	Your Choice
Name	"Student Coin"	_____
Symbol	"STU"	_____
Decimals	18 (standard)	18
Initial Supply	1,000,000	_____

Optional Features:

- ☐ Mintable (create new tokens)
- ☐ Burnable (destroy tokens)
- ☐ Pausable (emergency stop)
- ☐ Capped (maximum supply)

Think about what your token represents

ERC-20 Standard

The Interface Every Token Implements:

Core Functions:

- `totalSupply()`
- `balanceOf(address)`
- `transfer(to, amount)`

Allowance Functions:

- `approve(spender, amount)`
- `allowance(owner, spender)`
- `transferFrom(from, to, amt)`

Events:

- `Transfer(from, to, amount)` - Every transfer
- `Approval(owner, spender, amount)` - Every approval

Why Standards Matter:

- Works with all wallets automatically
- Can be listed on exchanges
- Composable with DeFi protocols

ERC-20 is the most widely used token standard

Basic Token Code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

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

contract MyToken is ERC20 {
    constructor(
        string memory name,
        string memory symbol,
        uint256 initialSupply
    ) ERC20(name, symbol) {
        // Mint initialSupply tokens to deployer
        // Note: ERC20 uses 18 decimals by default
        _mint(msg.sender, initialSupply * 10**decimals());
    }
}
```

That's It! OpenZeppelin provides all ERC-20 functionality.

10 lines of code for a fully functional token

Enhanced Token Code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyToken is ERC20, ERC20Burnable, Ownable {
    uint256 public constant MAX_SUPPLY = 10_000_000 * 10**18;

    constructor(string memory name, string memory symbol, uint256 initialSupply)
        ERC20(name, symbol) Ownable(msg.sender) {
        _mint(msg.sender, initialSupply * 10**decimals());
    }

    function mint(address to, uint256 amount) public onlyOwner {
        require(totalSupply() + amount <= MAX_SUPPLY, "Exceeds max");
        _mint(to, amount);
    }
}
```

Added: mint function, max supply cap, burn capability

Deploy Your Token

In Remix:

1. Create `MyToken.sol` with your code
2. Compile with Solidity 0.8.20+
3. Deploy & Run → Injected Provider
4. Connect MetaMask (Sepolia)

Constructor Parameters:

- `name`: `"Student Coin"`
- `symbol`: `"STU"`
- `initialSupply`: `1000000`

Deploy:

1. Click **Deploy**
2. Confirm in MetaMask
3. Wait for confirmation
4. **Save your contract address!**

Deployment costs 0.002 ETH on mainnet

Add Token to MetaMask

Import Your Custom Token:

1. Open MetaMask
2. Scroll down, click **Import tokens**
3. Paste your contract address
4. Symbol and decimals auto-fill
5. Click **Add Custom Token**
6. Click **Import**

You Should See:

- Your token in asset list
- Full balance (all initial supply)
- Ready to send!

MetaMask can display any ERC-20 token

Transfer Tokens

Method 1: MetaMask

1. Click your token
2. Click **Send**
3. Enter recipient address
4. Enter amount
5. Confirm transaction

Method 2: Remix

1. Find deployed contract
2. Use `transfer(to, amount)`
3. **Important:** Amount includes decimals!
4. For 100 tokens: 100000000000000000000

Amount Calculation:

$$\text{Raw Amount} = \text{Tokens} \times 10^{18}$$

100 tokens = 100 followed by 18 zeros

Understanding Decimals

Why 18 Decimals?

- Matches ETH precision
- Allows very small fractions
- Industry standard

Conversion Examples:

Human Amount	Raw Amount (18 decimals)
1 token	1,000,000,000,000,000,000
0.5 tokens	500,000,000,000,000,000
100 tokens	100,000,000,000,000,000,000
1 million	1,000,000,000,000,000,000,000,000

Quick Formula:

```
raw = human_amount * 10**18
```

Wallets and UIs handle decimal conversion for users

Verify on Etherscan

Why Verify?

- Source code becomes public
- Users can interact via Etherscan
- Builds trust and transparency

Steps:

1. Go to sepolia.etherscan.io
2. Find your contract
3. Click **Contract** → **Verify**
4. Select compiler 0.8.20
5. Choose license: MIT
6. Paste flattened source code
7. Enter constructor arguments (ABI encoded)
8. Submit

Verified contracts show a green checkmark

Token Economics Thinking

Questions to Consider:

Supply:

- Fixed or inflationary?
- Maximum cap?
- Who can mint?

Distribution:

- Initial allocation?
- Vesting schedules?
- Community rewards?

Value Drivers:

- What utility does your token provide?
- Why would someone want to hold it?
- What creates demand?

Technology is easy; economics is hard

Congratulations!

You created your own cryptocurrency!

What You Accomplished:

- ✓ Designed token parameters
- ✓ Wrote ERC-20 contract
- ✓ Deployed to blockchain
- ✓ Added to wallet
- ✓ Transferred tokens

Next Steps:

- Deploy to mainnet (real ETH required)
- Add liquidity on Uniswap
- Build utility for your token
- Complete the Token Economy project

Welcome to the world of tokenization!

Resources

Documentation:

- OpenZeppelin: `docs.openzeppelin.com`
- EIP-20: `eips.ethereum.org/EIPS/eip-20`
- Token Wizard: `wizard.openzeppelin.com`

Project Materials:

- Notebook: `projects/notebooks/06_create_crypto.ipynb`
- Web Guide: `.../projects/create-crypto/`

Advanced Projects:

- Token Economy (full project)
- DeFi Protocol