

COMP5568 Decentralized Finance

Lab 1: Smart Contracts for DeFi

BAI, Dongbin

dong-bin.bai@connect.polyu.hk



Outline

- Introduction
- Task 1: Deploy ERC-20 Token Contract
- Task 2: Transfer the ERC-20 Token



Introduction

- Remix: <https://remix.ethereum.org/>

The screenshot shows the Remix web3 IDE interface. At the top, there's a banner with the text "Build, Test, and Deploy Smarter with Remix". Below the banner, a subtext reads: "Remix is a rich toolset that can be used for the entire journey of contract development by users of any knowledge level, and as a learning lab for Web3 teaching and experimentation." There are two buttons: "Launch App" and "Start Learning Now".

The main workspace is titled "default_workspace". On the left, the "FILE EXPLORER" sidebar shows a file structure with contracts like 1_Storage.sol, 2_Owner.sol, 3_Ballot.sol, and scripts like 4_tx.js. The central area displays the Solidity code for the 1_Storage.sol contract:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;

contract Storage {
    uint256 number;
    /**
     * @dev Store a value in variable
     */
    function store(uint256 num) public {
        number = num;
    }
    /**
     * @dev Return value
     */
    function retrieve() public view returns (uint256) {
        return number;
    }
}
```

A tooltip for the "store" function lists AI-generated options: "Generate documentation", "Explain the function", "Explain this code", and "Toggle RemixAI Assistant".

To the right, the "RemixAI Assistant" panel features a blue butterfly icon and the text: "RemixAI provides you personalized guidance as you build. It can break down concepts, answer questions, and write code." It includes buttons for "Explain", "Generate", and "Debug". At the bottom, it says "Explore all RemixAI features".

At the very bottom, a footer bar includes "Welcome to Remix 0.50.8", "Listen on all transactions", "Filter with transaction hash or address", and a search bar.



Introduction

- The ERC-20 introduces a standard for **Fungible Tokens**, in other words, they have a property that makes each Token be exactly the same (in type and value) as another Token.
- For example, an ERC-20 Token acts just like the ETH, meaning that 1 Token is and will always be equal to all the other Tokens.
- Example functionalities ERC-20 provides:
 - **Transfer** tokens from one account to another.
 - Get the current token balance of an account.
 - Get the total supply of the token available on the network.
 - Approve whether an amount of token from an account can be spent by a third-party account.



Outline

- Introduction
- Task 1: Deploy ERC-20 Token Contract
- Task 2: Transfer the ERC-20 Token



Task 1: Create from template

Compile ▾ Home X

+ Create a new Workspace

Learn. Explore. Create.

Documentation Website

Download Remix Desktop Windows v1.1.6

Other versions and platforms

Recent Workspaces

default_workspace created 33 minutes ago

v1.5.0 Release

- Consider Supporting Remix
- Maximize Right Panel or Terminal
- Clone Now in Workspace Modal

Release Notes

Remix Desktop

Remix Desktop Release

Remix Desktop is now released! It has access to the native terminal, downloadable compilers for offline work, and uses the wallet of your choice!

Try it Out!

Most used plugins 🔥

Explore all plugins

Start Learning

RemixAI Assistant

RemixAI Assistant is a tool that helps you write code faster and easier.

Build with AI

Remix Guide Videos

Watch videos to get up to speed with Remix and Solidity!

Start Watching!



Task 1: Create from template

The screenshot shows the main workspace creation interface. At the top, there are two main buttons: 'Create blank' (Create an empty workspace) and 'Create with AI' (Generate a workspace with AI). Below these are two more options: 'Contract Wizard' (Create a new workspace with the OpenZeppelin Wizard) and 'Import Project' (Import an existing project). A search bar is at the top left. The bottom section is titled 'Workspace Templates' and includes filters for ZKP, Solidity, ERC721, ERC20, ERC1155, Cookbook, and Clear filter. The 'ERC20' filter is selected and highlighted with a red box. Under the 'OPENZEPPELIN' heading, the 'ERC20' template is listed with a red box around it. It is described as a 'customizable fungible token contract'. Below it, under 'COOKBOOK', is a 'Cookbook' section.

The screenshot shows the configuration page for the 'ERC20' template. At the top right are buttons for 'Solidity' and 'ERC20'. The main area is titled 'ERC20' with a back arrow. On the left is a sidebar with 'Contract settings' and several checkboxes for features: 'Mintable' (unchecked), 'Burnable' (unchecked), 'Pausable' (unchecked), 'ACCESS CONTROL' (radio button selected for 'Ownable'), 'Roles' (unchecked), 'Managed' (unchecked), 'UPGRADABILITY' (checkboxes for 'UUPS' and 'Transparent' both unchecked), and a checkbox for 'Initialize as a Git repository' which is unchecked. On the right, the Solidity code for the 'AwesomeToken' contract is displayed:

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.4.0
pragma solidity ^0.8.27;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract AwesomeToken is ERC20 {
    constructor() ERC20("AwesomeToken", "MTK") {}
}
```

At the bottom right is a blue button labeled 'Create workspace' with a checkmark icon.



Task 1: Select the solidity file

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it shows a project structure:
 - contracts** (highlighted with a red box)
 - AwesomeToken.sol** (selected item, highlighted with a dark grey bar)
 - scripts**
 - tests**
 - AwesomeToken_test.sol**
 - .prettierrc.json**
 - remix.config.json**
- Editor:** On the right, the code editor displays the Solidity contract 'AwesomeToken.sol':

```
1 // SPDX-License-Identifier: MIT
2 // Compatible with OpenZeppelin Contracts ^5.4.0
3 pragma solidity ^0.8.27;
4
5 import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
6
7 contract AwesomeToken is ERC20 {
8     constructor() ERC20("AwesomeToken", "MTK") {}    ┌ infinite gas 710800 gas
9 }
10
11 }
```
- Top Bar:** Includes icons for AI, Save, Compile, Home, and the current file tab.



Task 1: Implement the Awesome Token

The screenshot shows a Solidity IDE interface with the following details:

- Toolbar:** Includes "Compile" dropdown, search, refresh, and "Home" button.
- File Tabs:** "AwesomeToken.sol" (active), "AwesomeToken.json", "AwesomeToken_metadata.json", "ERC20.sol", and "ethers-lib.ts".
- Code Editor:** Displays the Solidity code for "AwesomeToken".

```
// SPDX-License-Identifier: MIT
// This tells the compiler which version of Solidity to use.
// ^0.8.27 means "version 0.8.27 or newer within the 0.8.x series".
pragma solidity ^0.8.27;

// We import the ERC20 contract from OpenZeppelin. ERC20 is a standard interface for tokens on Ethereum. It already includes functions like transfer(), balanceOf(), approve(), etc.
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// Our contract "AwesomeToken" inherits from ERC20. This means it automatically gets all ERC20 functions and state variables.
contract AwesomeToken is ERC20 {
    // "owner" stores the address that has special privileges (can mint tokens).
    // "public" means Solidity auto-creates a getter function so anyone can read it.
    address public owner;

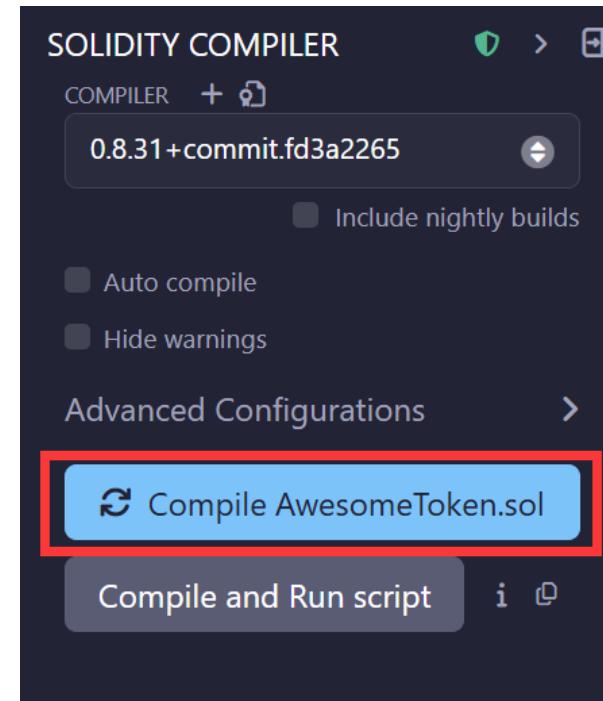
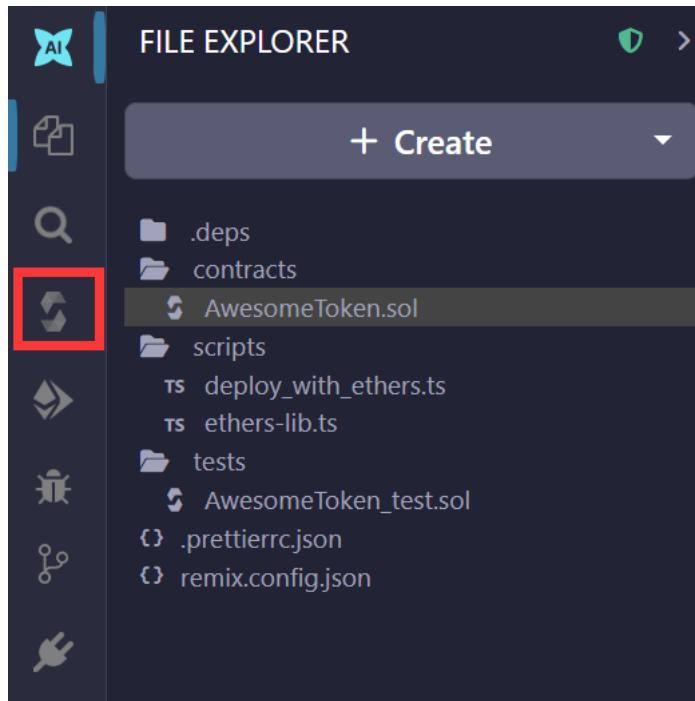
    // A modifier is reusable code that runs BEFORE a function executes.
    // "onlyOwner" checks if the person calling the function is the owner. If msg.sender is NOT the owner, the transaction stops (reverts) with the error message.
    // The underscore represents where the actual function code will run. If the require() check passes, execution continues to the function body.
    modifier onlyOwner() {
        require(msg.sender == owner, "Not the owner!");
        _;
    }

    // The constructor runs ONLY ONCE when the contract is deployed.
    constructor() ERC20("AwesomeToken", "ASM") {
        owner = msg.sender;

        // "_mint" is an internal ERC20 function that creates new tokens.
        // Parameters:
        //   1. msg.sender - WHO receives the tokens (the deployer's wallet)
        //   2. amount - HOW MANY tokens to create
        //
        // Why multiply by 10 ** decimals()?
        // - ERC20 tokens use "decimals" (default is 18) to handle fractions.
        // - To create 1,000,000 tokens, we calculate: 1000000 * 10^18
        _mint(msg.sender, 1000000 * 10 ** decimals());
    }

    // This function allows the owner to create NEW tokens after deployment. This increases totalSupply and the balance of the "to" address.
    // Example: To mint 500 tokens to address 0x123... call: mint(0x123... 500 * 10**18)
}
```

Task 1: Compile smart contract





Task 1: Compile smart contract

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it shows a tree view of project files:
 - .deps
 - artifacts
 - build-info
 - AwesomeToken_metadata.json
 - AwesomeToken.json** (highlighted)
 - contracts
 - AwesomeToken.sol
 - scripts
 - deploy_with_ETHERS.ts
 - ethers-lib.ts
 - tests
 - AwesomeToken_test.sol
 - .prettierrc.json
 - remix.config.json
- Code Editor:** The main area displays a JSON file, specifically `AwesomeToken.json`. The code is a large object definition with several nested properties and arrays. Some lines are numbered from 8610 to 8637.
- Top Bar:** It includes tabs for "Compile", "Home", "AwesomeToken.sol", "AwesomeToken.json", and "AwesomeToken_metadata.json".



Task 1: Deploy smart contract

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT Reset State

Remix VM (Osaka)

VM

ACCOUNT

0x5B3...eddC4 (100.0 ETH)

+ Authorize Delegation

GAS LIMIT

Estimated Gas

Custom 3000000

VALUE

0 Wei

CONTRACT

AwesomeToken - contracts/AwesomeT

evm version: osaka

Deploy

At Address Load contract from Address

Deploy & Run Transactions

AwesomeToken - contracts/AwesomeT

Deploy At Address Load contract from Address

Transactions recorded 3 i

Deployed Contracts 1

AWSOMEETOKEN AT 0x7BE...SF 4 x

BALANCE: 0 ETH

approve address spender: uint256 value

mint address to: uint256 amount

transfer address to: uint256 value

transferFrom address from: address to: uint256 value

allowance address owner: address type

balanceOf address account

decimals

name

owner

symbol

totalSupply

Low level interactions

CALLDATA

Transact

Compile Home Awesometoken.sol Awesometoken.json Awesometoken_metadata.json ERC20.sol ethers-lib.tz

```
9 // Our contract "AwesomeToken" inherits from ERC20. This means it automatically gets all ERC20 functions and state variables.
10 contract AwesomeToken is ERC20 {
11     // "owner" stores the address that has special privileges (can mint tokens).
12     // "public" means Solidity auto-creates a getter function so anyone can read it.
13     address public owner;
14
15     // A modifier is reusable code that runs BEFORE a function executes.
16     // "onlyOwner" checks if the person calling the function is the owner. If msg.sender IS NOT the owner, the transaction stops (reverts) with the error message.
17     // The underscore represents where the actual function code will run. If the require() check passes, execution continues to the function body.
18     modifier onlyOwner() {
19         require(msg.sender == owner, "Not the owner!");
20     }
21
22
23     // The constructor runs ONLY ONCE when the contract is deployed.
24     constructor() ERC20("AwesomeToken", "ASW") {
25         owner = msg.sender;
26
27         // "_mint" is an internal ERC20 function that creates new tokens.
28         // Parameters:
29         // 1. msg.sender - WHO receives the tokens (the deployer's wallet)
30         // 2. amount - HOW MANY tokens to create
31         //
32         // Why multiply by 10 ** decimals()?
33         // - ERC20 tokens use "decimals" (default is 18) to handle fractions.
34         // - To create 1,000,000 tokens, we calculate: 1000000 * 10**18
35         _mint(msg.sender, 1000000 * 10 ** decimals());
36     }
37
38     // This function allows the owner to create NEW tokens after deployment. This increases totalSupply and the balance of the "to" address.
39     // Example: To mint 500 tokens to address 0x123..., call: mint(0x123..., 500 * 10**18)
40     function mint(address to, uint256 amount) public onlyOwner {
41         _mint(to, amount);
42     }
43
44     // INHERITED FUNCTIONS (automatically available from ERC20)
45     // You DON'T need to write these - they come from the ERC20 import:
46     // ...
47     // ... +msg.sender+address to: uint256 amount)
```

Explain contract

0 Listen on all transactions Filter with transaction hash or address

creation of AwesomeToken pending...

0x5B3...eddC4 from: 0x5B3...eddC4 to: AwesomeToken (constructor) value: 0 wei data: 0x606...f0033 logs: 1 hash: 0xa4...e9417

AI copilot



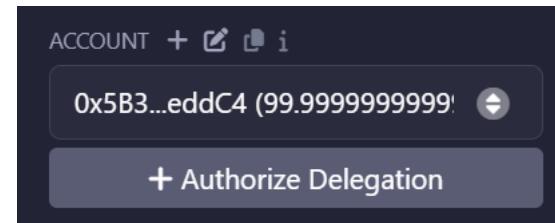
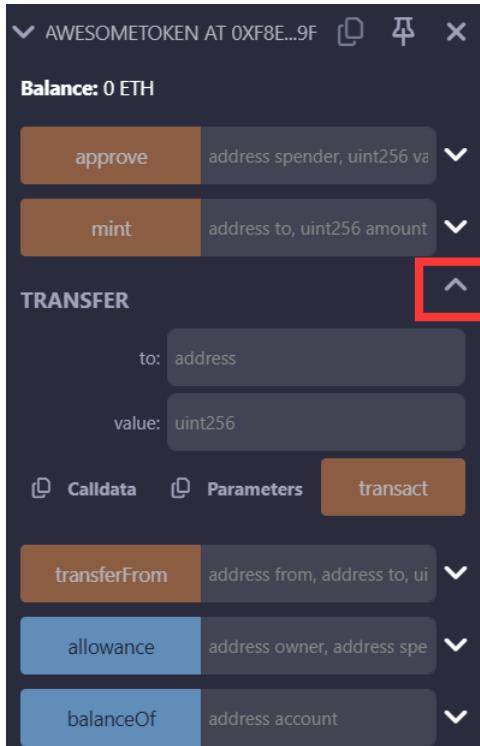
Outline

- Introduction
- Task 1: Deploy ERC-20 Token Contract
- Task 2: Transfer the ERC-20 Token

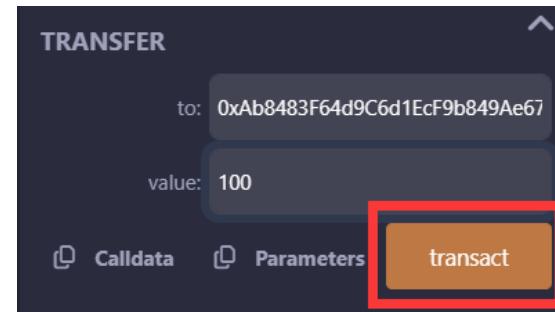


Task 2: Transfer the ERC-20 Token

- **Account 1 (Owner):** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
- **Account 2:** 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2



**Make sure to use
the correct account
to sign the TxN.**



Task 2: Check the token balance



- **Account 1 (Owner):** 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
 - **Account 2:** 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2

BALANCEOF
account: 0xAB8483F64d9C6d1EcF9b849Ae67

Check the balance of Account 2.



- PolyLink is a universal paradigm, providing a practical path from DePINs to RWA interconnections: <http://polylink.evan.cafe/>
- If you are interested in this project, you may contact me and **GET INVOLVED!**
 - Email: dong-bin.bai@connect.polyu.hk



Scan to check PolyLink's detailed introduction.

Thank You!