

Blockchain Smart Contract Lab

Remix Ethereum Platform and UI

2-Hour Hands-On Workshop

Vangelis Malamas, Dimitris Koutras, Spiros Papagiannopoulos



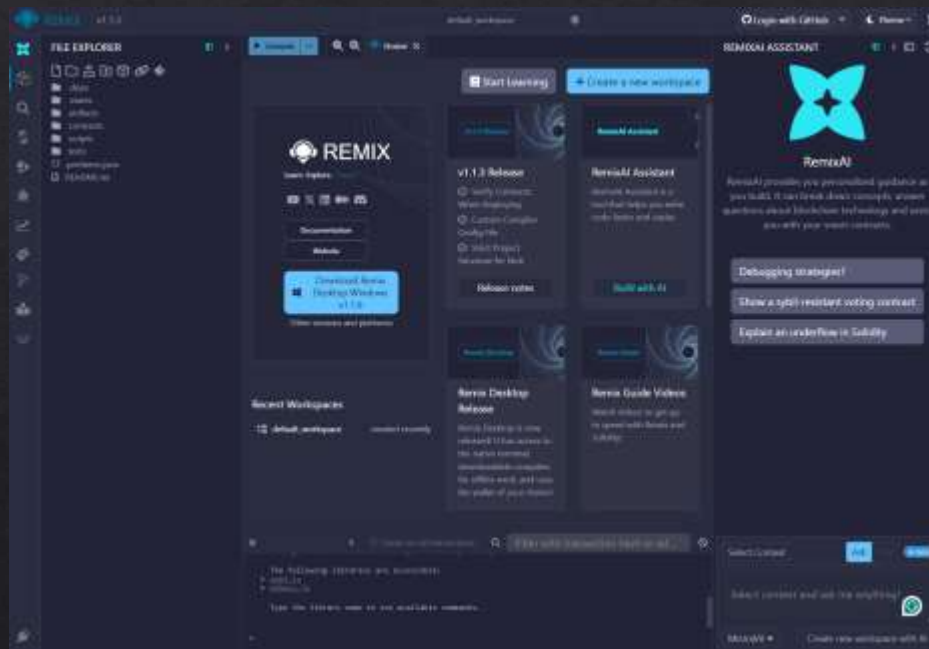
Quick Overview

- ❖ **Blockchain** is a distributed ledger technology that records transactions across multiple computers securely, immutably, and transparently.
- ❖ **Ethereum** is a blockchain platform that allows developers to create and deploy decentralized applications (dApps) using smart contracts written in Solidity.
- ❖ A **Smart Contract** is a self-executing digital agreement coded in Solidity. It enforces terms automatically without intermediaries.



Remix IDE Introduction

- ◆ **Remix IDE** is a browser-based development environment that supports writing, compiling, deploying, and debugging Solidity smart contracts.



Remix IDE Interface Components

◆ Begin by opening a browser and navigate to:

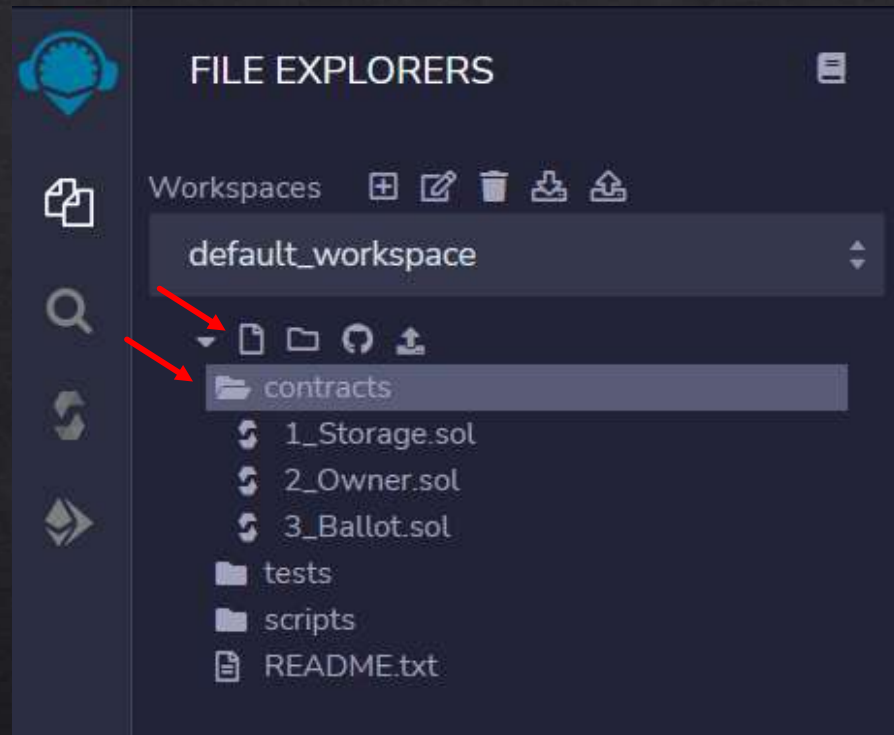
<https://remix.ethereum.org>

File Explorer
Code Editor
Compiler
Deploy & Run panel
Console.



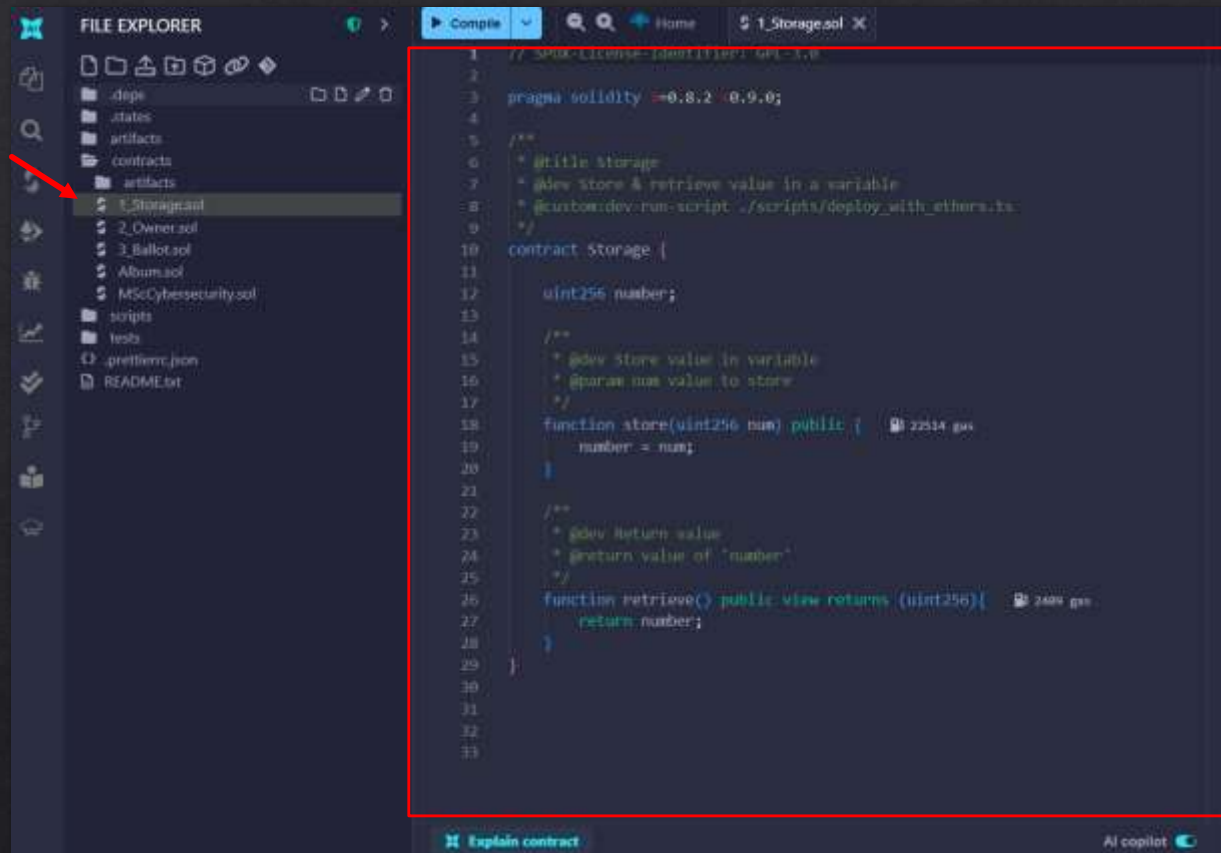
Remix IDE File Explorer

Create a new contract



Remix IDE Code Editor

Write the SC

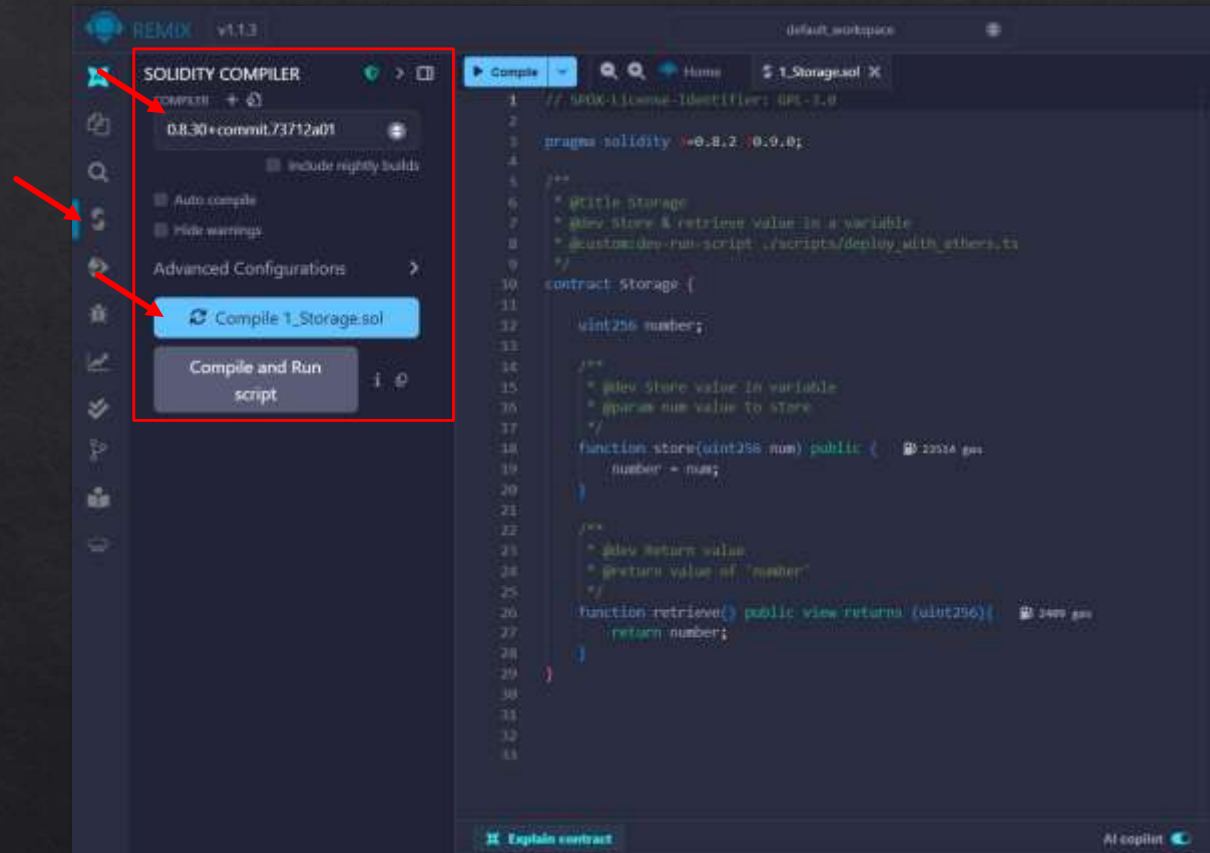


```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.2 ^0.9.0;
4
5 /**
6  * @title Storage
7  * @dev Store & retrieve value in a variable
8  * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
9  */
10 contract Storage {
11
12     uint256 number;
13
14     /**
15      * @dev Store value in variable
16      * @param num value to store
17      */
18     function store(uint256 num) public { 22514 gas
19         number = num;
20     }
21
22     /**
23      * @dev Return value
24      * @return value of 'number'
25      */
26     function retrieve() public view returns (uint256) { 2404 gas
27         return number;
28     }
29 }
30
31
32
33
```



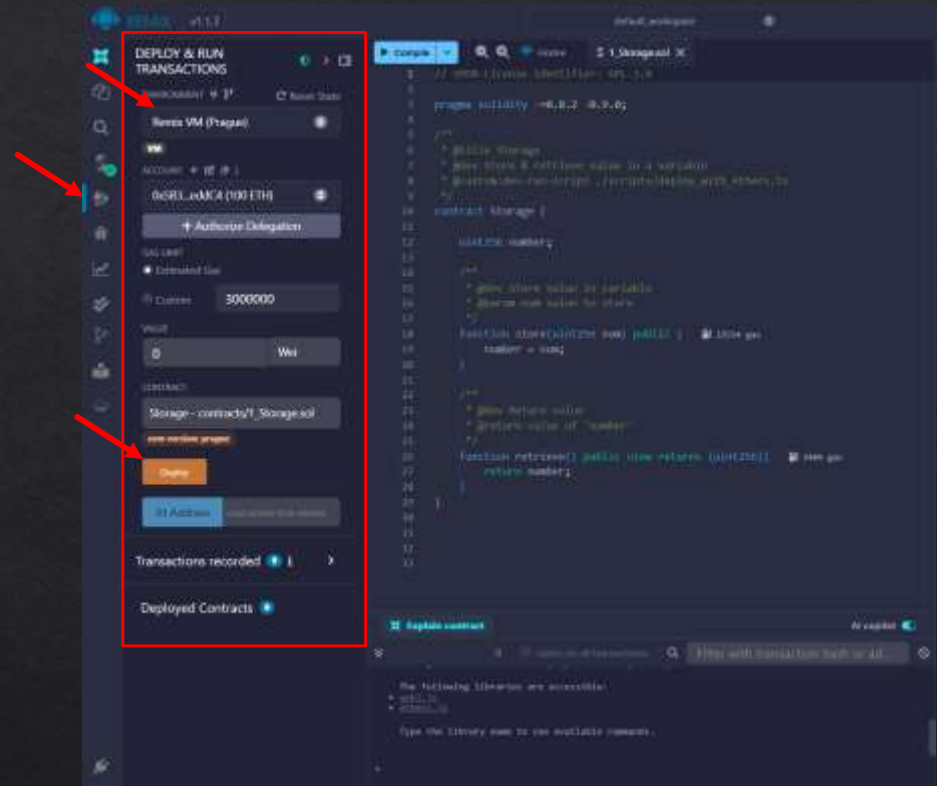
Remix IDE Compiler

Compile the SC

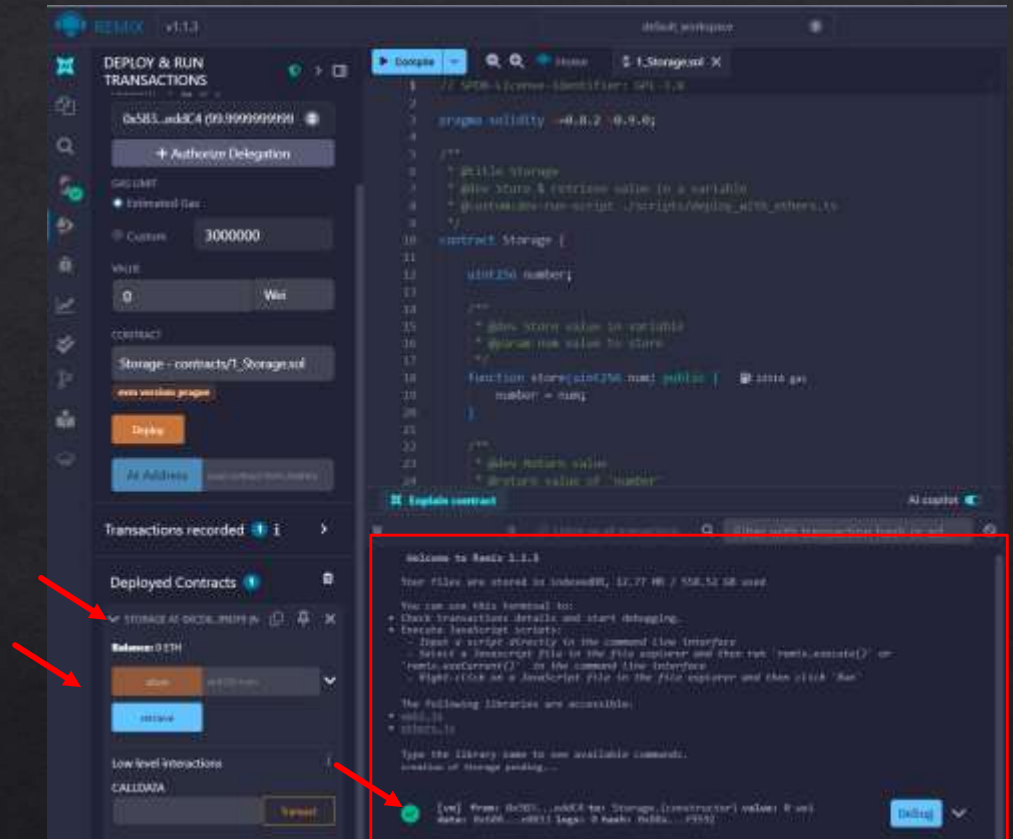


Remix IDE Deploy & Run

Deploy & Run the SC



Remix IDE Console



Developing dApp

Use Case: Jukebox

- ◆ General Description: *Build an app which uses an Ethereum blockchain network to keep track of the albums our Jukebox device is playing. This Jukebox allows users to request not just a song but an entire album.*



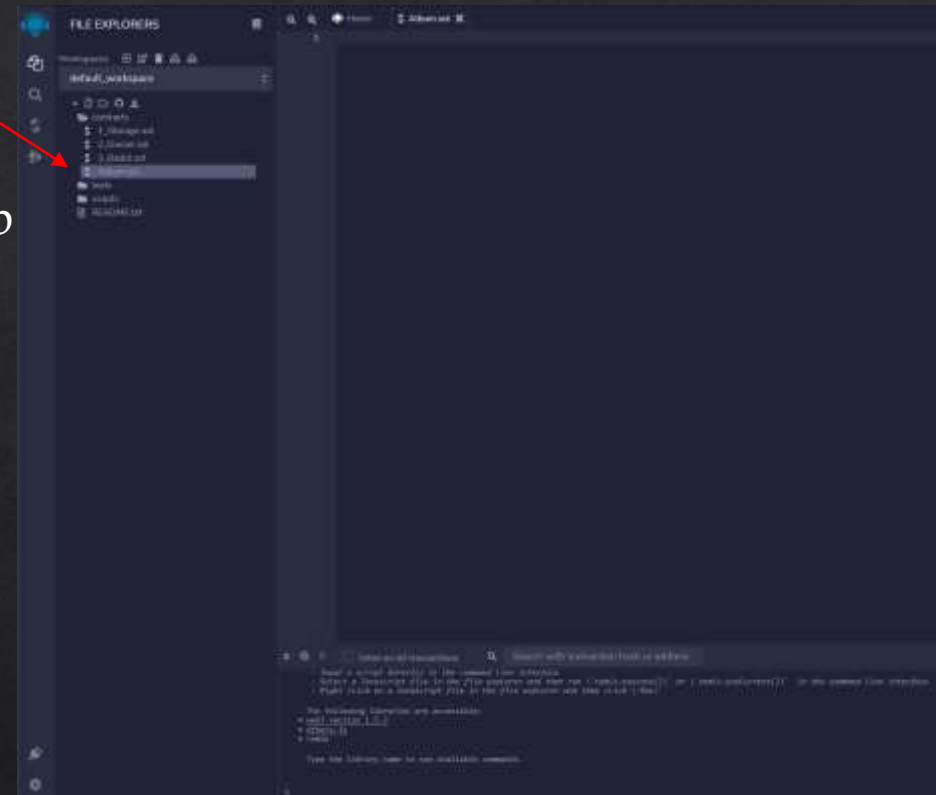
The decentralized, permanent ledger will serve as the official, trusted source of record for calculating royalty payments to the artists



Writing the First Smart Contract

- First, we create a new SC named “Album.sol” from the File Explorer tab

```
1 // SPDX-License-Identifier: CC-BY-1.0
2 // Creative Commons Attribution 1.0 Generic
3
4 // Contract will be compiled on version 0.7.0 or greater
5 pragma solidity ^0.7.0;
6
7 // A smart contract to model a music album
8 contract Album {
9
10 }
```



Writing the First Smart Contract

- First, we **create a new SC** named “*Album.sol*” from the File Explorer tab
- Then we **add variables**. We assume there are three critical properties of an album that should be tracked:
 - *The artist or group*
 - *The title of the album*
 - *The number of tracks on the album*

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;

contract Album {
    string public artist;
    string public albumTitle;
    uint public tracks;
}
```

```
7 // A smart contract to model a music album
8 contract Album {
9
10     // local state variables
11     // The artist/group who recorded the album
12     string artist = 'Nirvana';
13     // The album's title
14     string albumTitle = 'Nevermind';
15     // The number of tracks on the album
16     uint tracks = 13;
17
18 } // Album
```



Types of Variables in Solidity

- ◆ We need to “characterize” each variable that we use depending on the use purpose:
 - **bool**: *This is a Boolean type that returns True or False.*
 - **int/uint**: *Both **int** and **uint** represent integers or number values. The main difference is that integer can hold negative numbers as values.*
 - **address**: *The address type represents a 20byte value, which is meant to store an Ethereum address.*
 - **string**: *a dynamically signed string (alpha-numeric)*
 - **mapping**: *Hash tables with key types and values types.*
 - **struct**: *structs allows us to define new types (objects that contain multiple variables)*
 - **bytes**: *A dynamically-sized array*



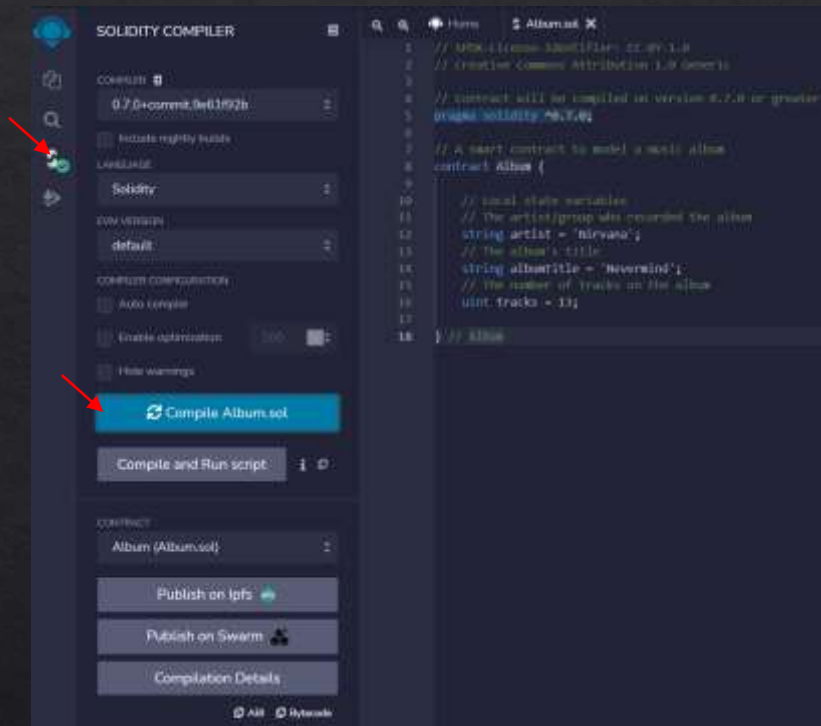
Visibility Specifiers in Solidity

◆ Solidity has four types of visibilities for both functions and variables:

- **public**: *This allows you to define functions or variables that can be called internally or through messages. [Visible to everyone]*
- **private**: *private variables and functions are only available to the current contract and not the derived contracts. [Visible only inside the current contract]*
- **internal**: *Functions and variables that can only be accessed internally (current contract and derived). [Visible only inside the current contract and the ones that inherit]*
- **external**: *functions that can be called from other contracts and transactions. They cannot be called internally, except with “this.functionName()”*

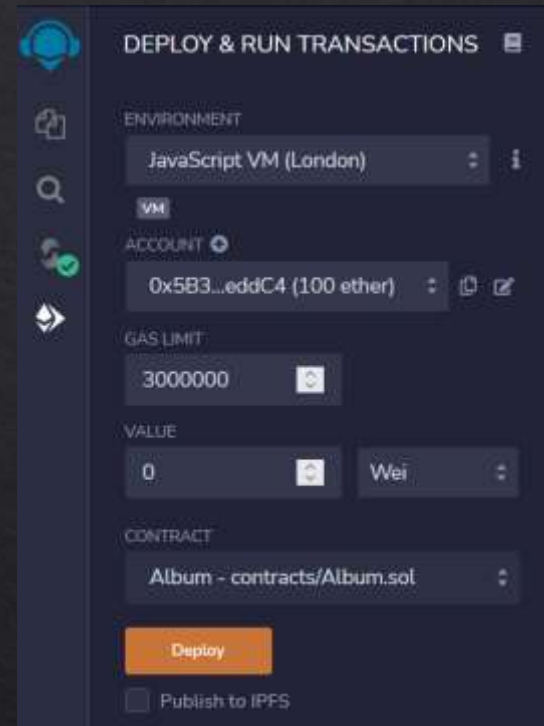


Compile & Deploy “Album.sol”



The screenshot shows the Solidity Compiler interface. On the left, the 'SOLIDITY COMPILER' sidebar is visible. It includes sections for 'Compiler' (version 0.7.0+commit.9e6192b), 'Language' (Solidity), 'EVM Version' (default), 'Compiler Configuration' (Auto compile, Enable optimization, Hide warnings), and 'Contract' (Album (Album.sol)). A red arrow points to the 'Compile Album.sol' button. Below it is a 'Compile and Run script' button. The main editor displays the code for 'Album.sol' with line numbers 1 through 18. The code includes a pragma statement for Solidity version 0.7.0 and defines a contract named 'Album' with variables for artist, album title, and number of tracks.

```
1 // SPDX-License-Identifier: CC-BY-1.0
2 // Creative Commons Attribution 1.0 Generic
3
4 // contract will be compiled at version 0.7.0 or greater
5 pragma solidity ^0.7.0;
6
7 // A smart contract to model a music album
8 contract Album {
9
10     // local state variables
11     // The artist/group who recorded the album
12     string artist = 'Nirvana';
13     // The album's title
14     string albumTitle = 'Nevermind';
15     // The number of tracks on the album
16     uint tracks = 13;
17
18 }
```



The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' interface. It includes sections for 'ENVIRONMENT' (JavaScript VM (London)), 'ACCOUNT' (0x5B3...eddC4 (100 ether)), 'GAS LIMIT' (3000000), 'VALUE' (0 Wei), and 'CONTRACT' (Album - contracts/Album.sol). A red arrow points to the 'Deploy' button. There is also a checkbox for 'Publish to IPFS'.



Compile & Deploy “Album.sol”

The screenshot displays the Remix IDE interface during the deployment of a smart contract named 'Album.sol'. The left sidebar contains the 'DEPLOY & RUN TRANSACTIONS' panel, which is configured with the following settings:

- ENVIRONMENT:** JavaScript VM (London)
- VM:** ACCOUNT: 0x5411... (99.99999999 ETH)
- GAS LIMIT:** 3000000
- VALUE:** 0 Wei
- CONTRACT:** Album - contracts/Album.sol

Three red arrows point to specific elements in the sidebar:

- The first arrow points to the 'Deploy' button.
- The second arrow points to the 'Deployed Contracts' section, which lists 'ALBUM AT 0x201... ALBUM PROXYING'.
- The third arrow points to the 'Deployed Contracts' section header.

The main editor area shows the source code of 'Album.sol' with the following content:

```
1 // SPDX-License-Identifier: CC-BY-1.0
2 // Creative Commons Attribution 1.0 Generic
3
4 // Contract will be compiled on version 0.7.0 or greater
5 pragma solidity ^0.7.0;
6
7 // A smart contract to model a music album
8 contract Album {
9
10     // Local state variables
11     // The artist/group who recorded the album
12     string artist = "Nirvana";
13     // The album's title
14     string albumTitle = "Nevermind";
15     // The number of tracks on the album
16     uint tracks = 15;
17
18 } // Album
```

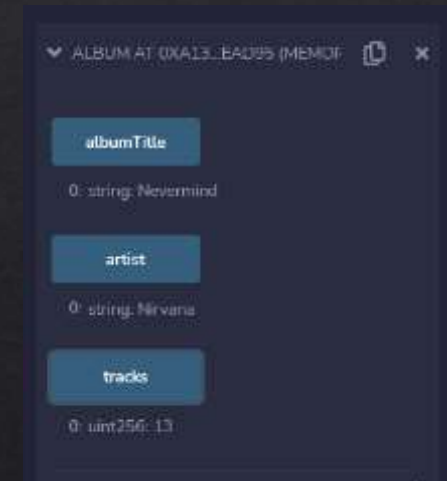
The bottom panel shows the console output, which includes a welcome message and instructions for using the IDE. A 'Debug' button is visible in the bottom right corner of the console area.



Change variable visibility

- ◆ Append the “public” scope modifier to the *artist*, *albumTitle* and *tracks* variables

```
5 pragma solidity ^0.7.0;
6
7 // A smart contract to model a music album
8 contract Album {
9     // Local state variables
10    // The artist/group who recorded the album
11    string public artist = 'Nirvana';
12    // The album's title
13    string public albumTitle = 'Nevermind';
14    // The number of tracks on the album
15    uint public tracks = 13;
16 } // Album
```



Adding a Constructor

- ◆ Add a *constructor* to initialize the album data when the contract is deployed:

Constructors are intended to contain any logic that should be performed once initially to ready the application usage

```
1 // SPDX-License-Identifier: CC-BY-1.0
2 // Creative Commons Attribution 1.0 Generic
3
4 // Contract will be compiled on version 0.7.0 or greater
5 pragma solidity ^0.7.0;
6
7 // A smart contract to model a music album
8 contract Album {
9
10     // Local state variables
11     // The artist/group who recorded the album
12     string public artist;
13     // The album's title
14     string public albumTitle;
15     // The number of tracks on the album
16     uint public tracks;
17
18     constructor() {
19         artist = 'Nirvana';
20         albumTitle = 'Nevermind';
21         tracks = 13;
22     } // constructor
23
24 }
```



Adding Getter and Setter Functions

- The get function (*getAlbum*) will return the current album information
- The set function (*setAlbum*) will allow the user to change the current album information

```
5  pragma solidity ^0.7.0;
6
7  // A smart contract to model a music album
8  contract Album {
9
10     // local state variables
11     // The artist/group who recorded the album
12     string public artist;
13     // The album's title
14     string public albumTitle;
15     // The number of tracks on the album
16     uint public tracks;
17     // The author of this smart contract
18     string public constant contractAuthor = 'Vangelis Malamas';
19
20     constructor() {
21         artist = 'Nirvana';
22         albumTitle = 'Nevermind';
23         tracks = 13;
24     } // constructor
25
26     // Returns the current album information
27     function getAlbum() public view returns (string memory, string memory, uint) {
28         return (artist, albumTitle, tracks);
29     } // getAlbum
30
31     // set the album information
32     function setAlbum(string memory _artist, string memory _albumTitle, uint _tracks) public {
33         artist = _artist;
34         albumTitle = _albumTitle;
35         tracks = _tracks;
36     } // setAlbum
37
38 }
```



Recompile and Deploy

Recompile and redeploy the updated contract. Use the UI to call `getAlbum()` and `setAlbum()` to verify behavior.

The image shows a web application interface on the left and a transaction log on the right. The web application has a sidebar with buttons: `setAlbum`, `albumTitle`, `artist`, `contractAuthor`, `getAlbum`, and `tracks`. Red arrows point to the `setAlbum` and `getAlbum` buttons. The transaction log on the right shows a series of transactions, including `creation of Album pending...`, `[vm] from 0x08...35c2 to: Album.(constructor) values: 0 wei data: 0x08...0011 logs: 0 hash: 0x0c...34c6`, `creation of Album pending...`, `[vm] from 0x08...35c2 to: Album.(constructor) values: 0 wei data: 0x08...0011 logs: 0 hash: 0x03...3612`, `call to Album.contractAuthor`, `[call] from: 0xb084E3642Kb1E1F70048e77455115815c2 to: Album.contractAuthor() data: 0x734...e1ac`, `creation of Album pending...`, `[vm] from 0x08...35c2 to: Album.(constructor) values: 0 wei data: 0x08...0011 logs: 0 hash: 0x0c...34c6`, `transaction of Album.setAlbum pending ...`, `[vm] from 0x08...35c2 to: Album.setAlbum(string,string,uint256) (0x08...0011 values: 0 wei data: 0x08...0000 logs: 0 hash: 0x49...1325c`, `call to Album.getAlbum`, and `[call] from: 0xb084E3642Kb1E1F70048e77455115815c2 to: Album.getAlbum() data: 0xd1c...781f4`. Red arrows point to the `setAlbum` and `getAlbum` transactions in the log.



Adding and Raising an Event

Defining an Event

Begin by adding the following code to define your new event, albumEvent. Define the event after your local state variables but before your constructor.

```
// Event which will be raised anytime the current album information is updated.  
event albumEvent(string albumEvent_Artist, string albumEvent_Title, uint  
albumEvent_Tracks);
```

Raising an Event

Add the following code to the setAlbum function. This will raise the event anytime the current album information is changed.

```
// Raise the albumEvent to let any event subscribers know the current album  
information has changed.  
emit albumEvent(_artist, _albumTitle, _tracks);
```



Adding Access Control

- ◆ We will be adding function modifier to our SC which will only allow the contract's owner to update the current album information.

To start, we'll need a variable to store the account number of the contract's owner. Declare a new variable, `owner`, using the `address` data type using the code below.

```
// The owner of the current instance of this smart contract  
address owner;
```

```
// Set the owner property of this contract instance to the constructor of this contract deployment  
owner = msg.sender;
```



Modifier

```
// This function modifier ensures that the initiator of any transaction  
// it is attached to matches the address of the contract's owner.  
// Use this function modifier for functions that should only  
// be performed by the owner of this contract instance.  
modifier onlyOwner {  
    if (msg.sender != owner) {  
        // The initiator of this transaction is NOT the contract instance's owner!  
    } else {  
        _;  
    } // else  
} // modifier onlyOwner
```

```
49  
50 // Set the album information  
51 function setAlbum(string memory _artist, string memory _albumTitle, uint _tracks) onlyOwner public {  
52     artist = _artist;  
53     albumTitle = _albumTitle;  
54     tracks = _tracks;  
55  
56     // Raise the albumEvent to let any event subscribers know the current album information has changed.  
57     emit albumEvent(artist, albumTitle, _tracks);  
58     // setAlbum  
59  
60     ↑ // Album
```



Connecting the digital wallet MetaMask



Begin by opening a browser and navigate to:

<https://metamask.io/download>



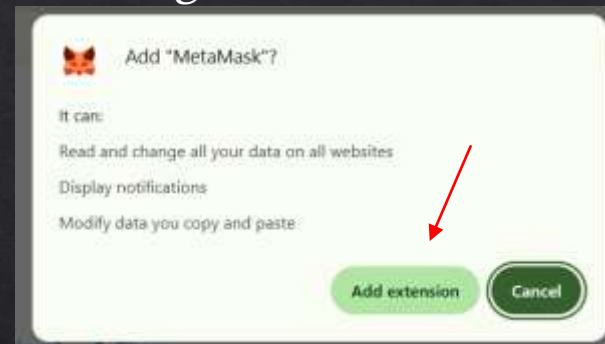
Installing and Setting up MetaMask



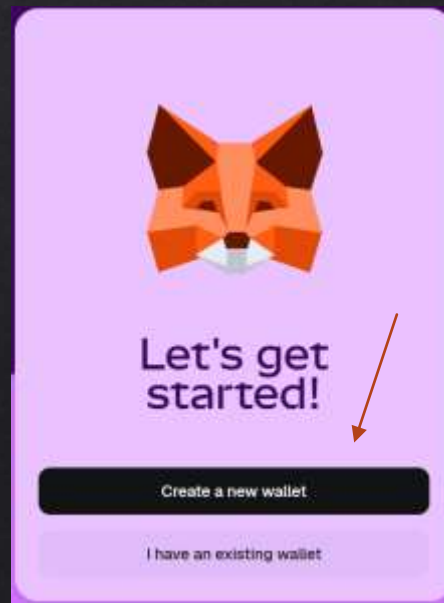
1. Click “Add to Chrome” to install the MetaMask browser extension.



2. Confirm the installation by selecting “Add extension.”



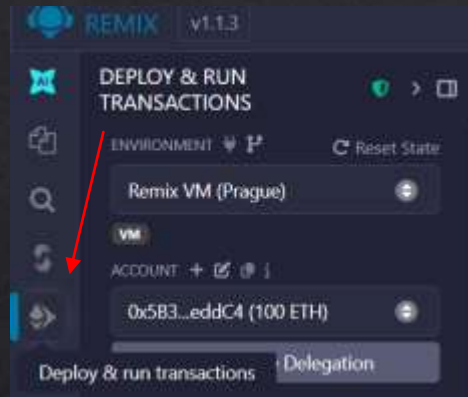
3. Select “Create a new wallet.”



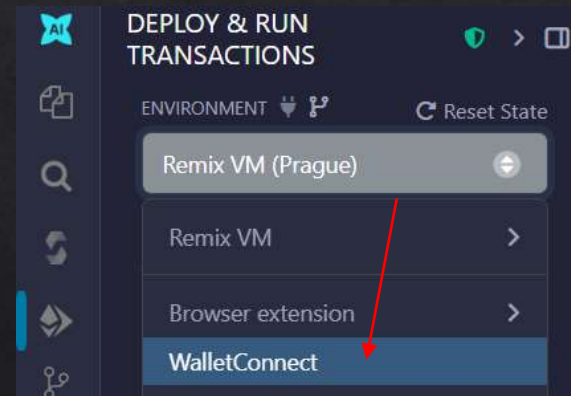
Connecting MetaMask to Remix IDE



1. Open Remix and navigate to the Deploy & Run Transactions panel.



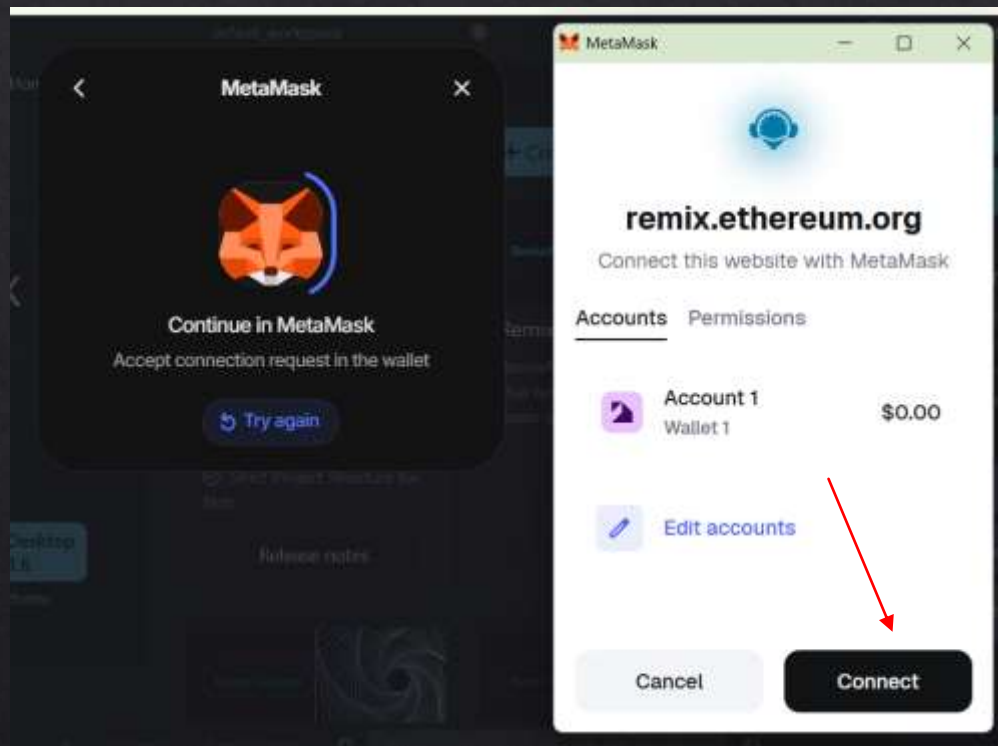
2. Choose Injected Provider MetaMask or WalletConnect from the list.



Signing Transactions with MetaMask



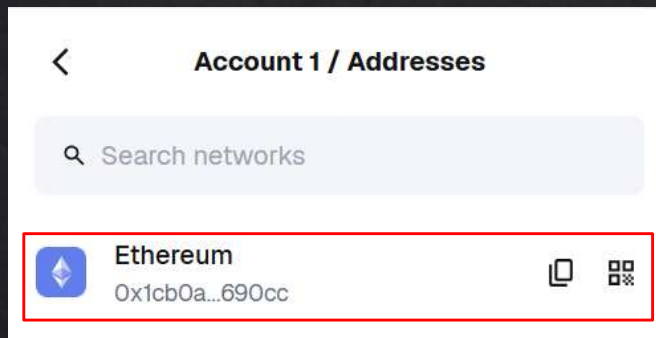
Confirm the wallet connection request that appears from remix.ethereum.org. Click Connect to authorize Remix to use your MetaMask account.



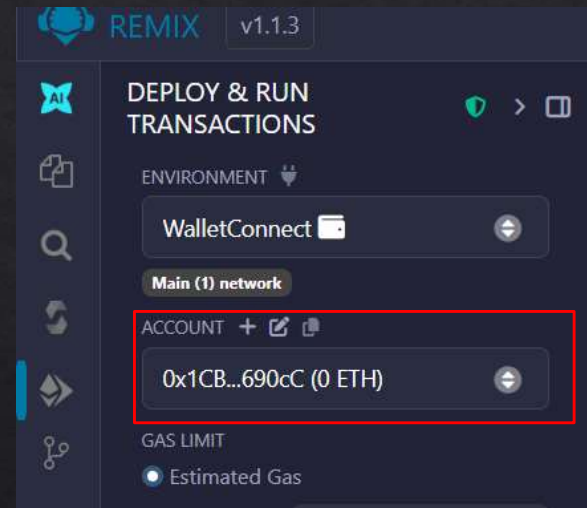
Confirming MetaMask Connection



1. Check your MetaMask account address.



2. Verify the same address in Remix.



Installing Node.js and Web3.js on Windows



1. Right-click on the Start Menu → Windows PowerShell → Run as Administrator.
2. Install Node.js and npm (Node Package Manager)

```
npm install -g node
```

3. Check Node.js and npm installation

```
node -v
```

```
npm -v
```

4. Install Web3.js library:

```
npm install web3
```



Connecting the SC with the UI

For the frontend we need an **html** and a **css** file



index.html



```
// Update these variables with YOUR account number and contract address
var myAccountNumber = 'PASTE YOUR ACCOUNT NUMBER HERE (FROM GANACHE)';
var myContractAddress = 'PASTE YOUR DEPLOYED CONTRACT ADDRESS HERE (FROM GANACHE)';
```

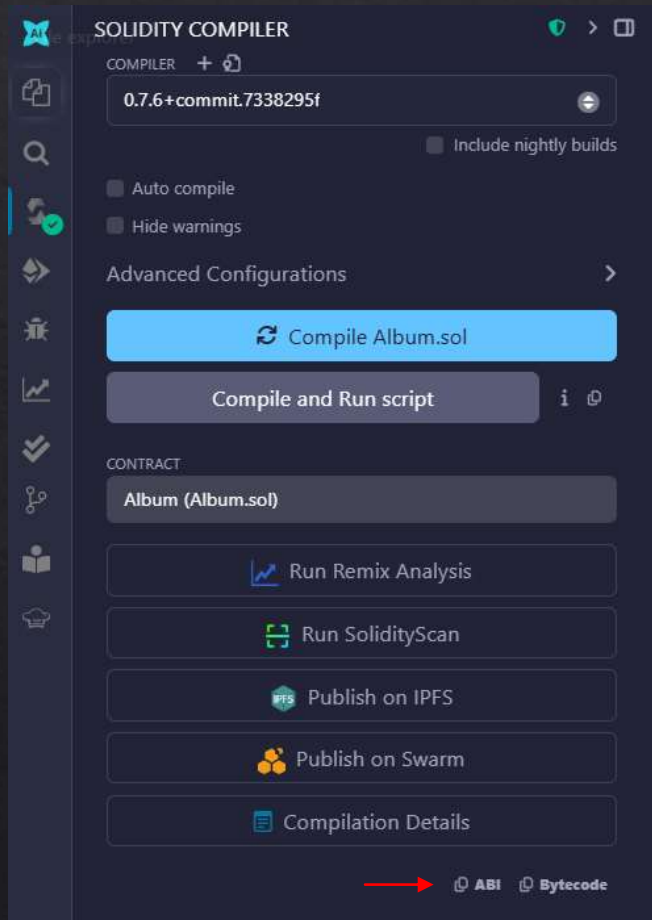


```
// Create a new web3 reference
// This syntax uses Web3's Http. This doesn't support subscriptions
// web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));

// The syntax below uses Web3's WebsocketProvider instead of the HttpProvider. This provider DOES support event subscriptions.
let web3 = new Web3(new Web3.providers.WebsocketProvider('ws://localhost:7545'))
```



Inserting the ABI



Copy ABI from the compiler tab
in Remix and paste it in the
index.html file

```
// Build a reference to the smart contract.  
var albumContract = new web3.eth.Contract('PASTE YOUR ABI HERE (without the quotes)');  
albumContract.options.address = myContractAddress;
```

Keep in mind that every time you recompile the SC
you need to adjust the settings in the html file



Resources

Resources:

- ◊ Solidity Documentation
- ◊ Remix IDE Tutorials
- ◊ Ethereum Developer Portal

