

# Blockchain Smart Contract Lab

Remix Ethereum Platform and UI  
2-Hour Hands-On Workshop

*Vangelis Malamas, Dimitris Koutras, Spiros Papagiannopoulos*



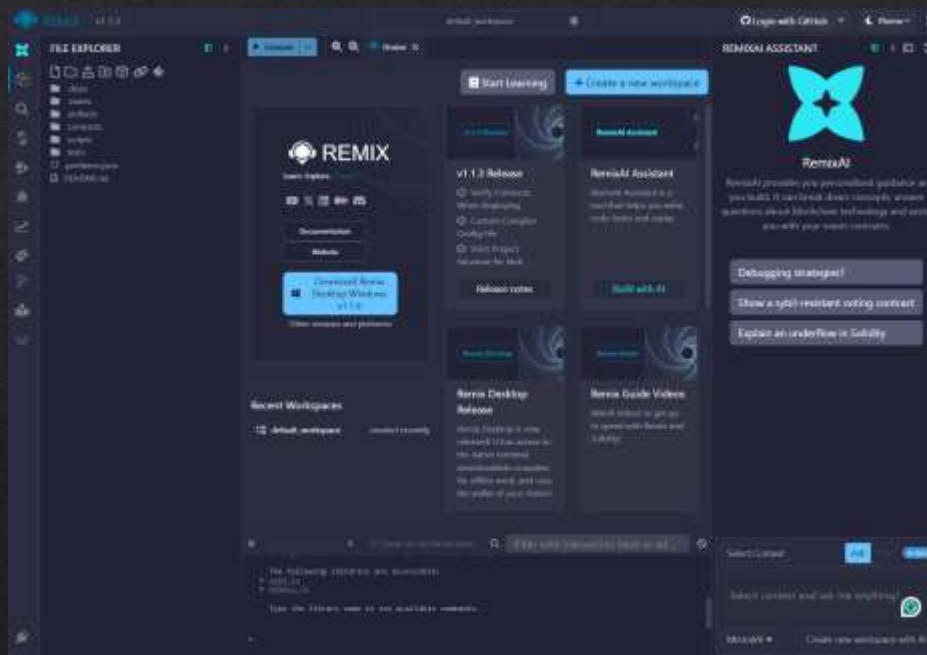
# Quick Overview

- ❖ **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:

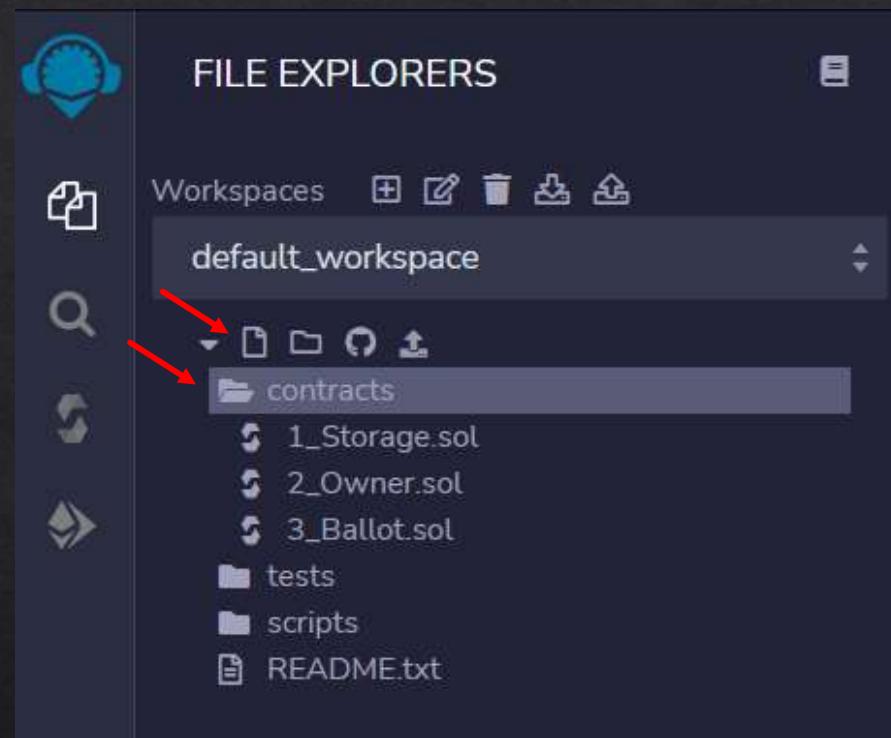
<https://remix.ethereum.org>

File Explorer  
Code Editor  
Compiler  
Deploy & Run panel  
Console.

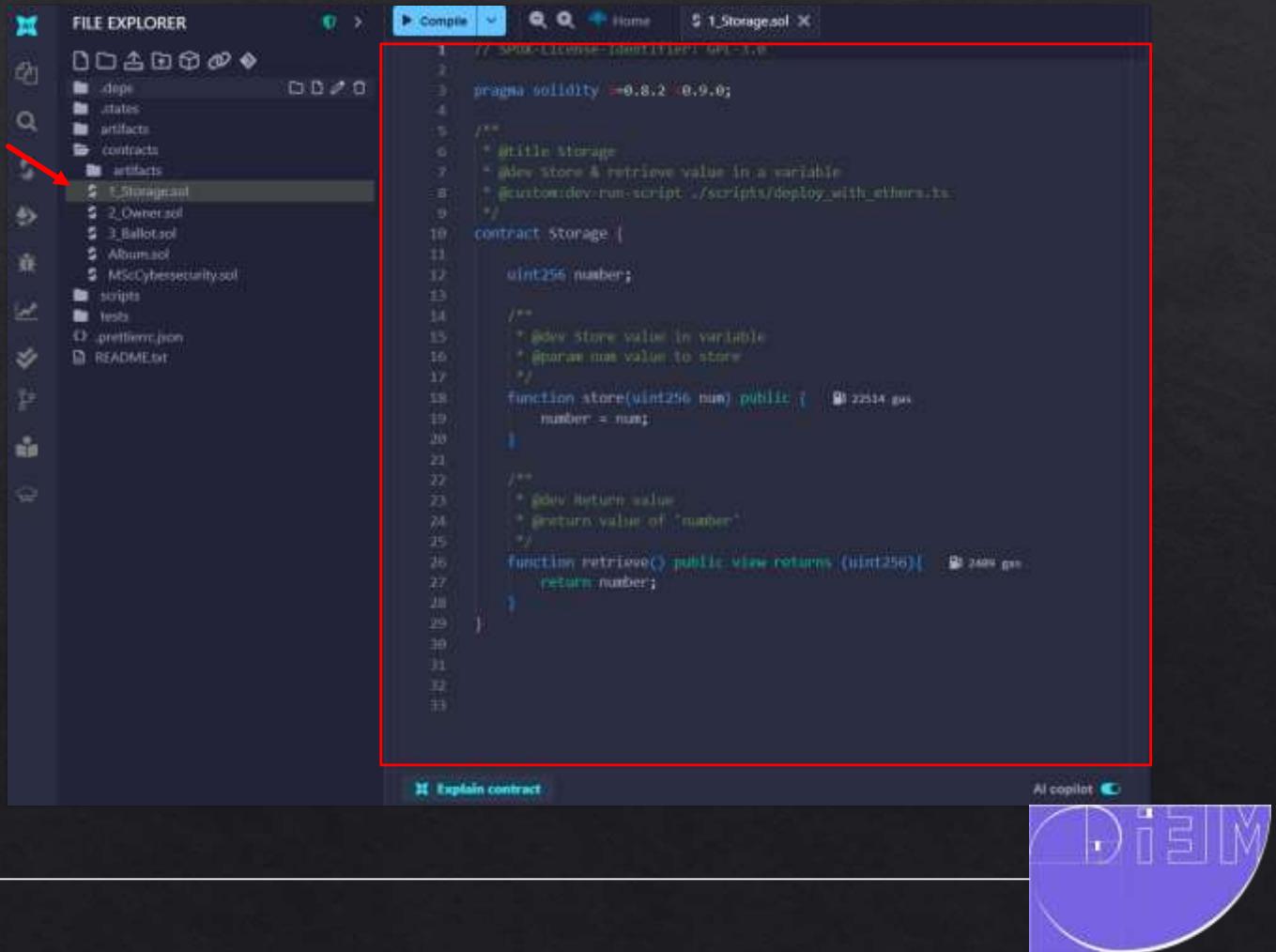


# Remix IDE File Explorer

Create a new contract



# Remix IDE Code Editor



The screenshot shows the Remix IDE interface. On the left is the **FILE EXPLORER** sidebar with a red arrow pointing to the **contracts** folder, which contains several Solidity source files: 1\_Storage.sol, 2\_Owner.sol, 3\_Ballot.sol, 4\_Album.sol, and MScCybersecurity.sol. The **scripts**, **tests**, and **README.txt** files are also listed. The main **Code Editor** window displays the **1\_Storage.sol** file. A red box highlights the code area. The code defines a `Storage` contract with two functions: `store(uint256 num)` and `retrieve()`. The `store` function stores a value in a variable, while the `retrieve` function returns the stored value.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @custom:dev-run-script ./scripts/deploy_with_ether.js
 */
contract Storage {
    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of "number"
     */
    function retrieve() public view returns (uint256) {
        return number;
    }
}
```

**FILE EXPLORER**

- deps
- states
- artifacts
- contracts
  - artifacts
  - 1\_Storage.sol
  - 2\_Owner.sol
  - 3\_Ballot.sol
  - 4\_Album.sol
  - MScCybersecurity.sol
- scripts
- tests
- .prettierignore
- README.txt

**1\_Storage.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @custom:dev-run-script ./scripts/deploy_with_ether.js
 */
contract Storage {
    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of "number"
     */
    function retrieve() public view returns (uint256) {
        return number;
    }
}
```

**Explain contract** **AI copilot** 

# Remix IDE Compiler

Compile the SC

The screenshot shows the Remix IDE interface. On the left, the Solidity Compiler panel is open, displaying version 0.8.30+commit.73712a01. A red box highlights the "Compile 1\_Storage.sol" button, which is blue and has a circular arrow icon. Below it is another button labeled "Compile and Run script". To the right of the compiler is the main workspace where a Solidity storage contract is written:

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

/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @author: dev-null-script . /scripts/deploy-with ethers.js
 */
contract Storage {
    uint256 number;

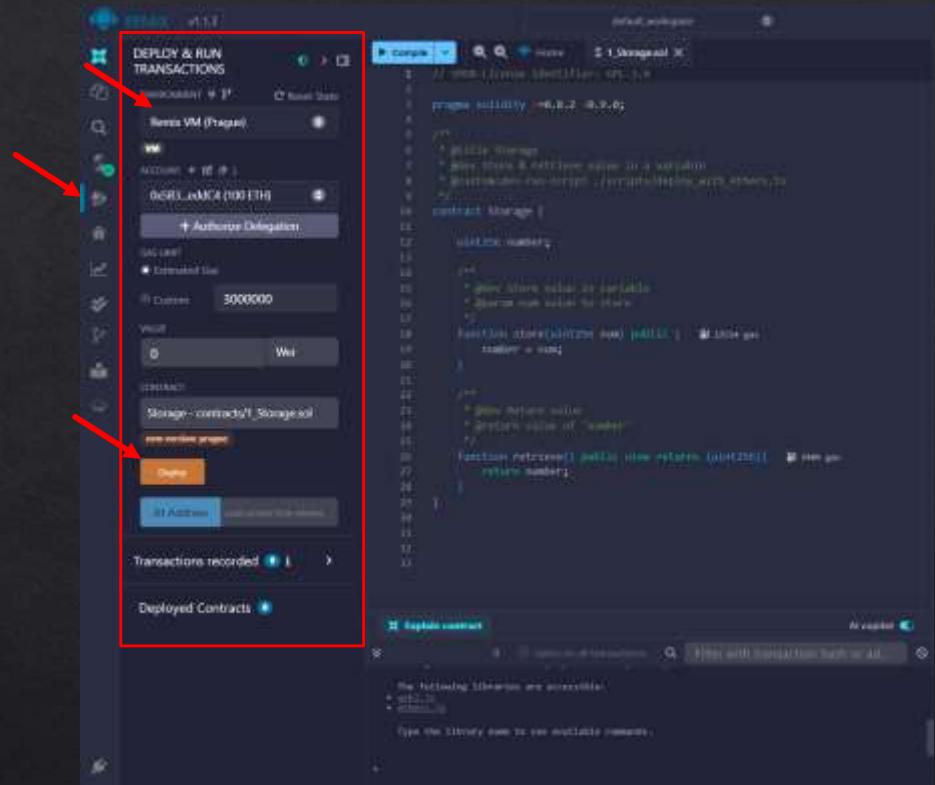
    /**
     * @dev store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint256) {
        return number;
    }
}
```

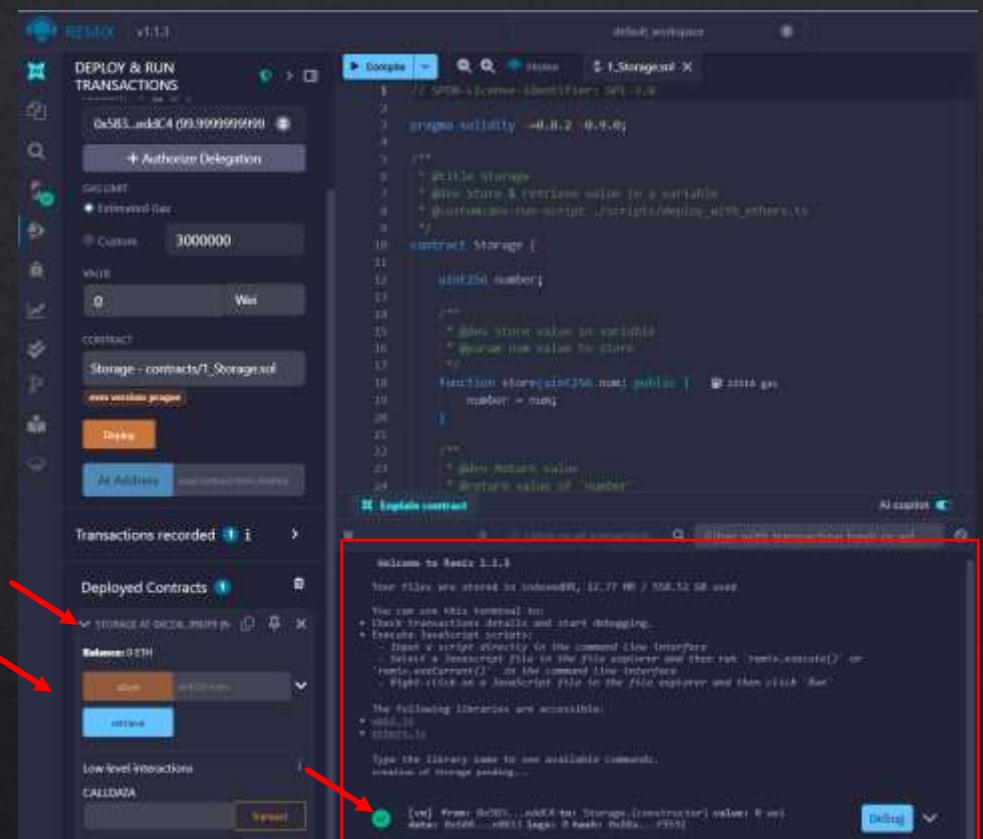
At the bottom of the workspace, there are two buttons: "Explain contract" and "AI copilot". A purple circular watermark for "DIME" is visible in the bottom right corner.

# Remix IDE Deploy & Run

Deploy & Run the SC



# Remix IDE Console



# Developing dApp

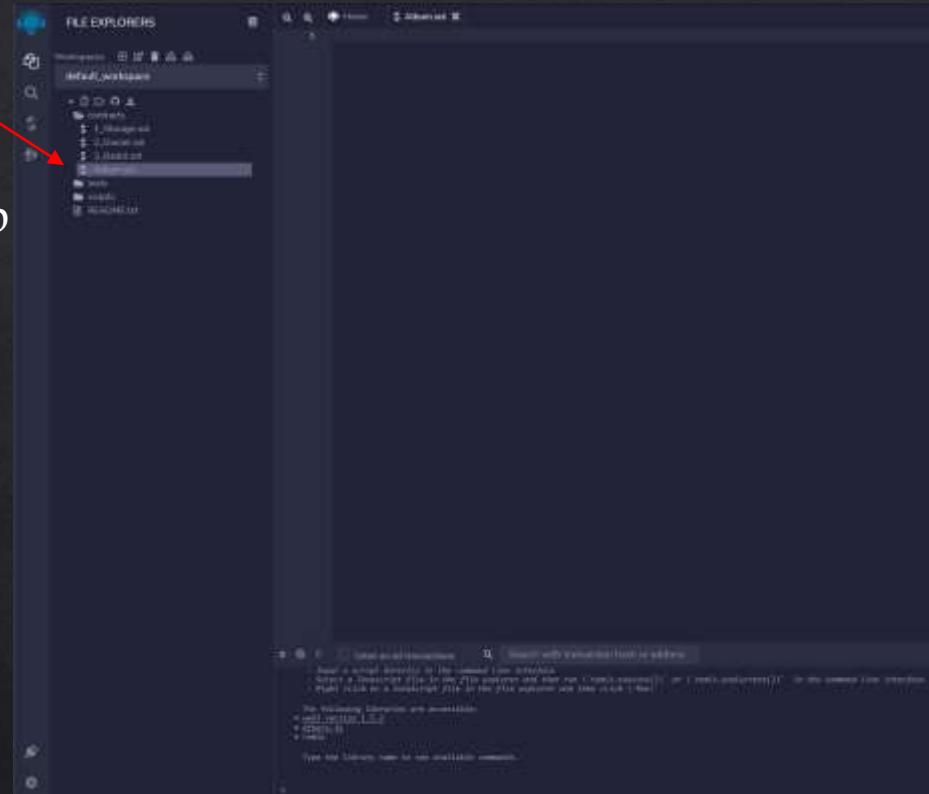
*Use Case: Jukebox*

- ❖ 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



The screenshot shows the Solidity IDE interface. On the left, the File Explorer tab displays a workspace named "default\_workspace" containing several contracts: "1.\Migrations.sol", "2.ERC20.sol", "3.LibSolidity.sol", "4.LibStorage.sol", "5.LibString.sol", and "6.LibMath.sol". A red arrow points to the "Album.sol" file, which is currently selected. On the right, the code editor shows the Solidity code for the "Album" contract:

```
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 } // Album
```



# 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-sixed 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 Truffle UI interface divided into two main sections: "SOLIDITY COMPILER" on the left and "DEPLOY & RUN TRANSACTIONS" on the right.

**SOLIDITY COMPILER:**

- Compiler:** 0.7.0+commit.9e6192b
- Languaje:** Solidity
- ENV Version:** default
- COMMIT COMPIRATION:** Auto compile
- Enable optimization:**
- Show warnings:**
- Compile Album.sol** (highlighted with a red arrow)
- Compile and Run script**
- CONTRACT:** Album (Album.sol)
- Publish on IPFS**
- Publish on Swarm**
- Compilation Details**
- All** **Hybrid**

**CODE (Album.sol):**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;
// This contract will be compiled at version 0.7.0 or greater
pragma solidity 0.7.0;

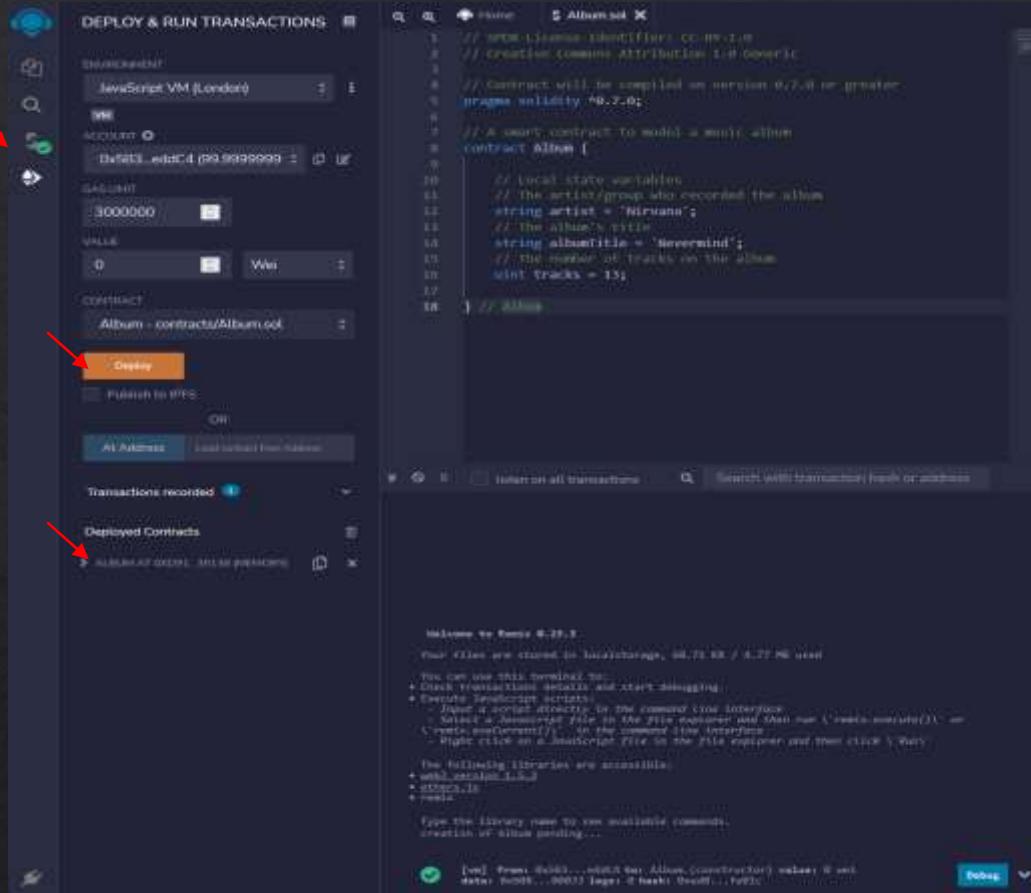
// A smart contract to model a music album
contract Album {
    // local state variables
    // The artist/group who recorded the album
    string artist = "Nirvana";
    // the album's title
    string albumTitle = "Nevermind";
    // the number of tracks on the album
    uint tracks = 13;
}
```

**DEPLOY & RUN TRANSACTIONS:**

- ENVIRONMENT:** JavaScript VM (London)
- VM:**
- ACCOUNT:** 0x5B3...eddC4 (100 ether)
- GAS LIMIT:** 3000000
- VALUE:** 0 Wei
- CONTRACT:** Album - contracts/Album.sol
- Deploy** (highlighted with a red arrow)
- Publish to IPFS



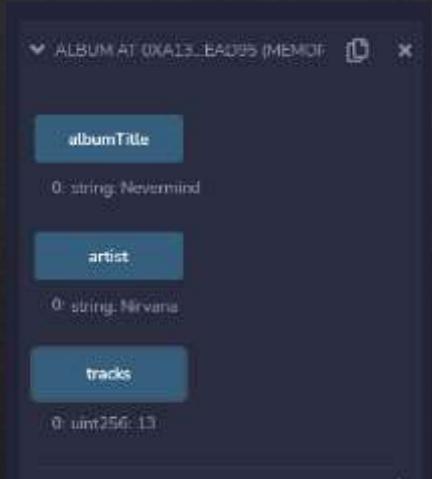
# Compile & Deploy “*Album.sol*”



# 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 } // Album
```



# 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 } // Album
```



# Recompile and Deploy

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

The screenshot shows the Truffle UI interface. On the left, the 'Deployed Contracts' sidebar lists the 'Album' contract with its address: 0x00...0000. It includes buttons for 'setAlbum', 'albumTitle', 'artist', 'contractAuthor', and 'getAlbum'. A red arrow points to the 'setAlbum' button. The main panel displays the transaction history:

- [CALL] from: 0x00...0000 to: Album.(constructor) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 creation of Album pending...
- [vm] from: 0x00...0000 to: Album.(constructor) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 creation of Album pending...
- [vm] from: 0x00...0000 to: Album.(constructor) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 call to Album.contractAuthor()
- [CALL] from: 0x00...0000 to: Album.(constructor) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 creation of Album pending...
- [vm] from: 0x00...0000 to: Album.(constructor) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 creation of Album pending...
- [vm] from: 0x00...0000 to: Album.setAlbum pending ...
  - [vm] from: 0x00...0000 to: Album.setAlbum(string,string,uint256) value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 call to /com/getAlbum
- [CALL] from: 0x00...0000 to: Album.getAlbum() value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 call to /com/getAlbum

A red arrow also points to the final transaction entry: '[CALL] from: 0x00...0000 to: Album.getAlbum() value: 0 wei data: 0x00...0000 logs: 0 hash: 0x00...0000 call to /com/getAlbum'.

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

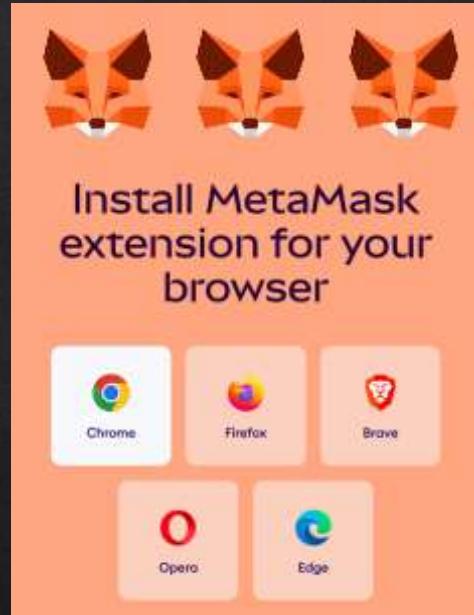


# Connecting the digital wallet MetaMask



Begin by opening a browser and navigate to:

<https://metamask.io/download>



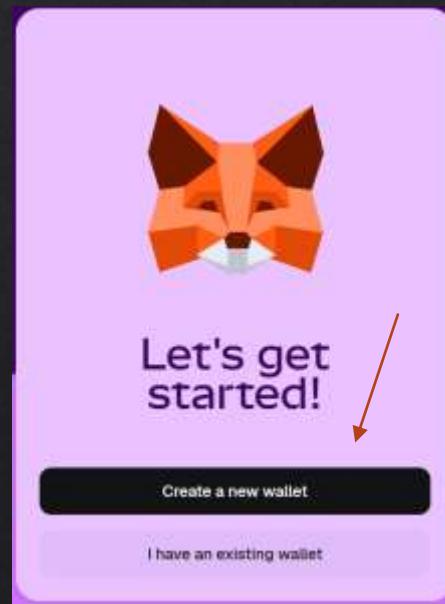
# Installing and Setting up MetaMask



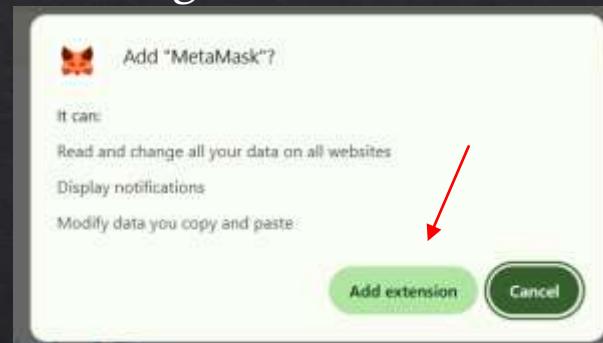
1. Click “Add to Chrome” to install the MetaMask browser extension.



3. Select “Create a new wallet.



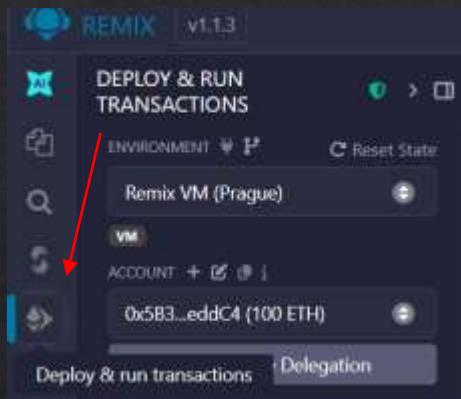
2. Confirm the installation by selecting “Add extension.”



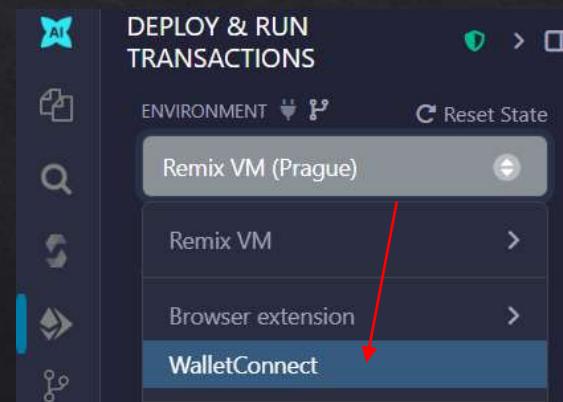
# Connecting MetaMask to Remix IDE



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



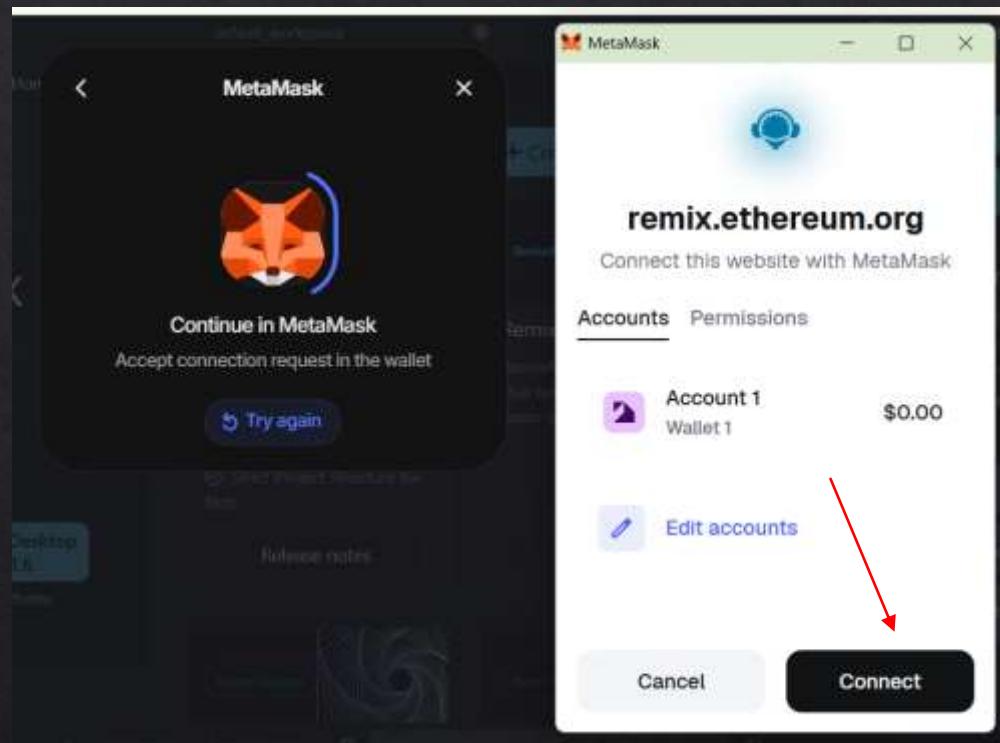
2. Choose Injected Provider  
MetaMask or WalletConnect  
from the list.



# Signing Transactions with MetaMask



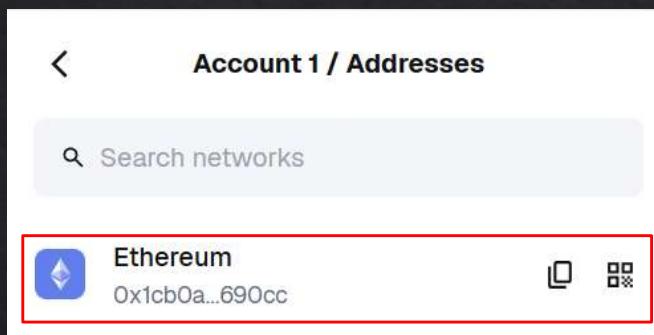
Confirm the wallet connection request that appears from remix.ethereum.org.  
Click Connect to authorize Remix to use your MetaMask account.



# Confirming MetaMask Connection



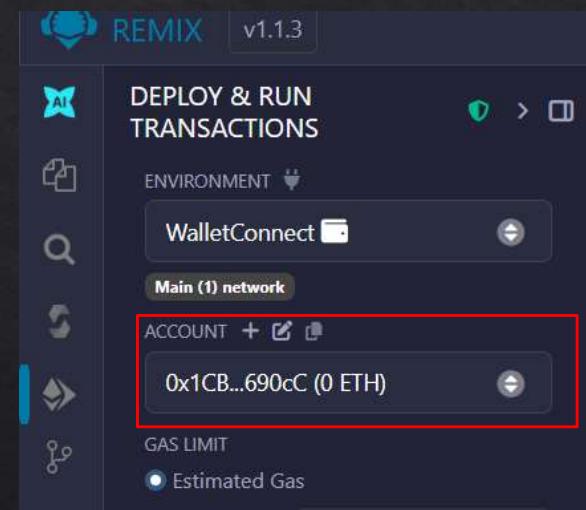
1. Check your MetaMask account address.



The screenshot shows the MetaMask extension's "Account 1 / Addresses" page. At the top, there is a search bar labeled "Search networks". Below it, a list of accounts is displayed, with the first account highlighted by a red box. The account details are as follows:

- Ethereum
- Address: 0x1cb0a...690cc

2. Verify the same address in Remix.



The screenshot shows the Remix IDE interface, version v1.1.3. On the left, there is a sidebar with various icons. The main area is titled "DEPLOY & RUN TRANSACTIONS" and includes sections for "ENVIRONMENT" (set to "WalletConnect") and "ACCOUNT" (set to "Main (1) network"). The account dropdown is also highlighted by a red box and shows the same Ethereum address: 0x1CB...690cC (0 ETH). Below the account section, there are buttons for "GAS LIMIT" and "Estimated Gas".



# 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
```

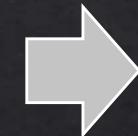
4. Install Web3.js library:

```
npm install web3
```



# Connecting the SC with the UI

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



```
// 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)';
```



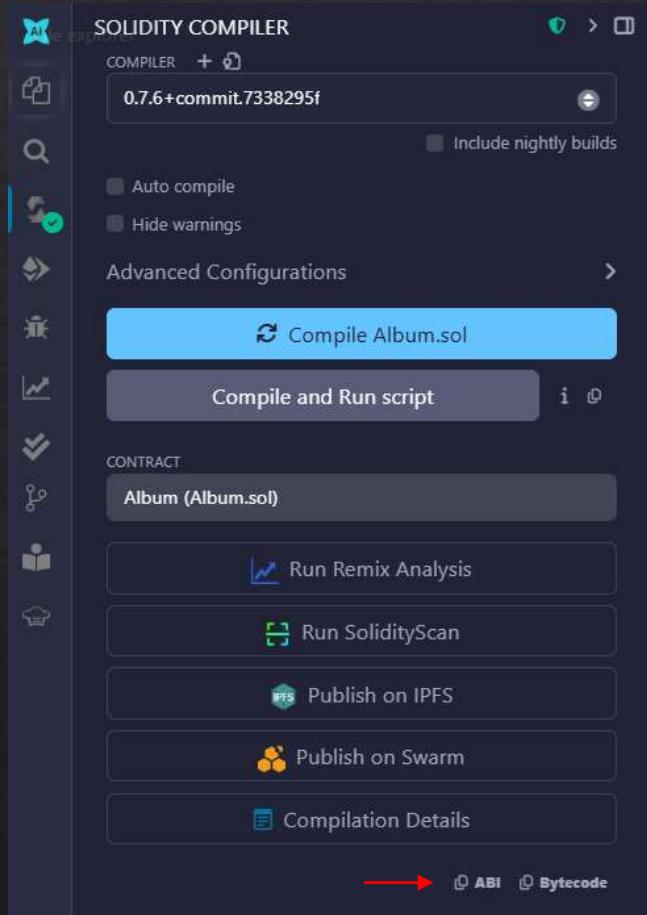
```
// Create a new web3 reference
//This syntax uses Web3's Http. This doesn't support subscriptions
//Web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));

// 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'))
```

index.html



# 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

