

2023-COS30049-Computing Technology
Innovation Project

Workshop Guide

Workshop 07

Introduction of Remix IDE and Solidity

Objective: By the end of this workshop, students will have an understanding of retrieving the data from the database and rendering the data in the React pages. In addition, students will know how to use Remix to write, deploy and test solidity files.

Workshop Structure:

1.Data Rendering From Database to Front-End (30 mins):

Data rendering in React refers to the process of displaying data from JavaScript objects or external sources (such as APIs) in a React component's user interface. In React, data rendering typically involves mapping over data and using JSX to generate the appropriate HTML elements for each data item, allowing it to be displayed on the screen.

STEP 1: Send HTTP Request: Send an HTTP request to fetch data.

```
// get data from the database
const handleButtonClick = () => {
  axios.get('http://127.0.0.1:8000/jsonData')
    .then(response => {
      // check the data in the console
      // open the developer mode in the Chrome browser
      console.log(response.data);
    })
    .catch(error => {
      console.error('error here: ', error);
    });
};
```

blog.js:26

```
▼ {name: 'Your name', Uni-year: 2, isStudent: true, hobbies: Array
  (2)} ⓘ
  Uni-year: 2
  ▼ hobbies: Array(2)
    0: "reading"
    1: "swimming"
    length: 2
    ► [[Prototype]]: Array(0)
  isStudent: true
  name: "Your name"
  ► [[Prototype]]: Object
```

>

STEP 2: Define Data State: Use React's useState to define state variables for storing the loaded data.

```
// State variable for storing loaded data
const [data, setData] = useState([]);
```

```

function Blog() {
  // State variable for storing loaded data
  const [data, setData] = useState([]);

  // get data from the database
  const handleClick = () => {
    axios.get('http://127.0.0.1:8000/jsonData')
      .then(response => {
        // check the data in the console
        // open the developer mode in the Chrome browser
        console.log(response.data);
      })
      .catch(error => {
        console.error('error here: ', error);
      });
  };

  return (
    // React Element Here
    <Paper ...
  );
}

export default Blog;

```

STEP 3: Set the Data State: when the AXIOS request gets the response from the API (FastAPI function), we need to set the value to the declared state variable.

```


// Store the fetched data in the state variable
setData(response.data);

```

STEP 4: Set the data inside the React elements (data rendering): We define a Blog element first with the default value. Then we define a button to trigger the API GET request. After receiving the response data, we can replace the data in the Blog element. Here is the sample code:

<https://codesandbox.io/p/sandbox/fastapi-demo-kkd6yv?file=/week7/section1-data%20loading/Blog.js:30,1-31,1>

[CLICK ME](#)



Standard license

\$19.00

Full resolution 1920x1080 • JPEG

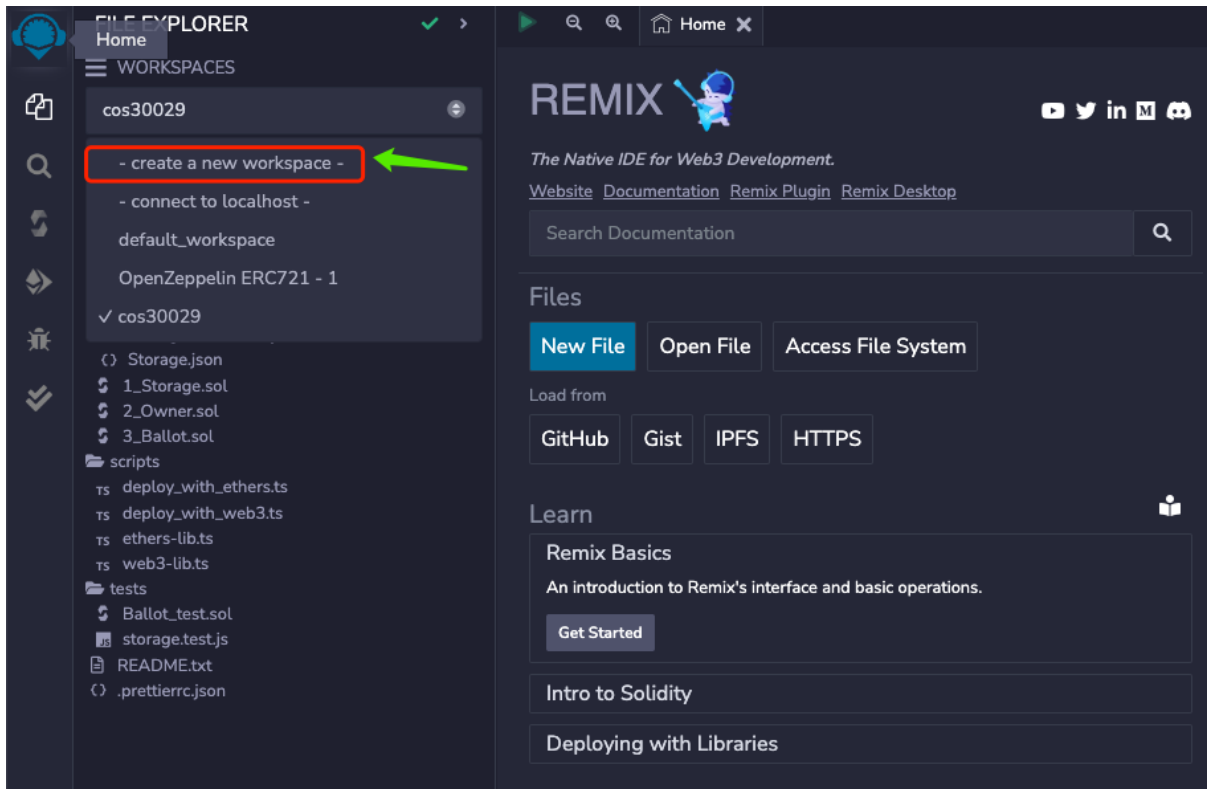
ID: 1030114

2. Remix Usage Overview (20 mins):

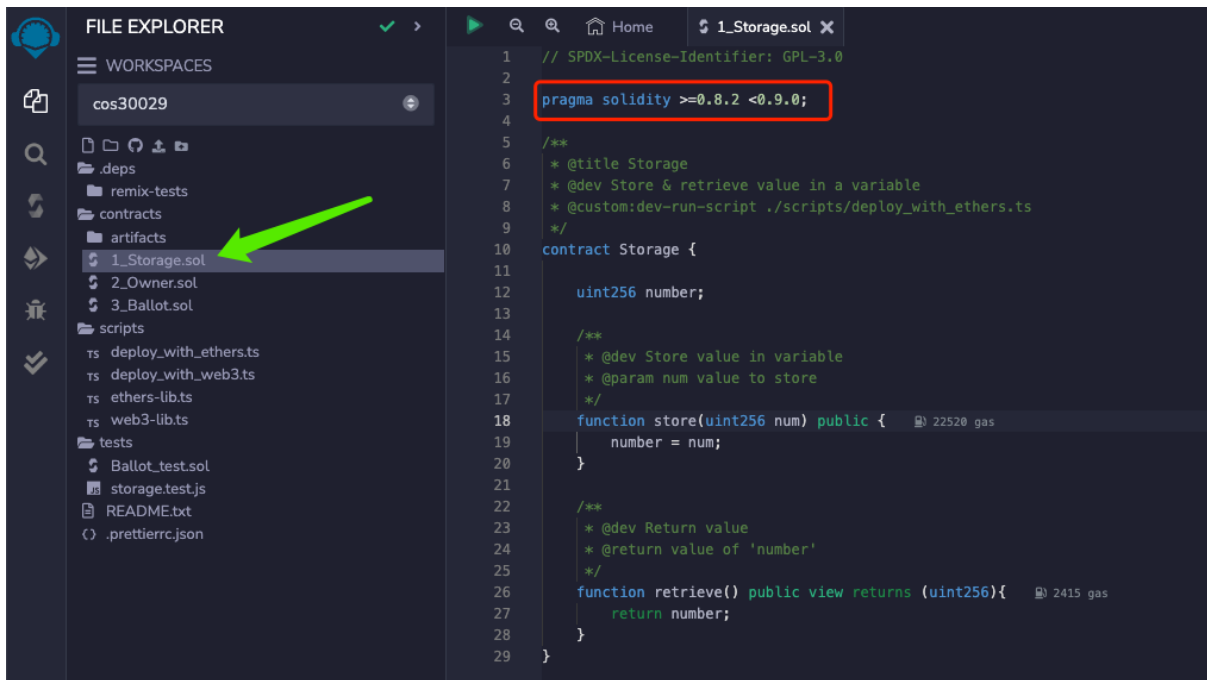
<https://remix.ethereum.org/> is a website and tool within the Ethereum ecosystem used for the development and testing of Ethereum smart contracts.

- Contract Deployment and Interaction: Developers can use Remix to deploy smart contracts to the Ethereum blockchain and interact with already deployed contracts. This allows for testing the functionality of contracts and how they can be interacted with.
-
- Debugging Tools: Remix provides robust debugging tools to assist developers in identifying and resolving errors and issues within smart contracts.

STEP 1: Open the Remix editor and create your workspace (i.e. COS 30049)

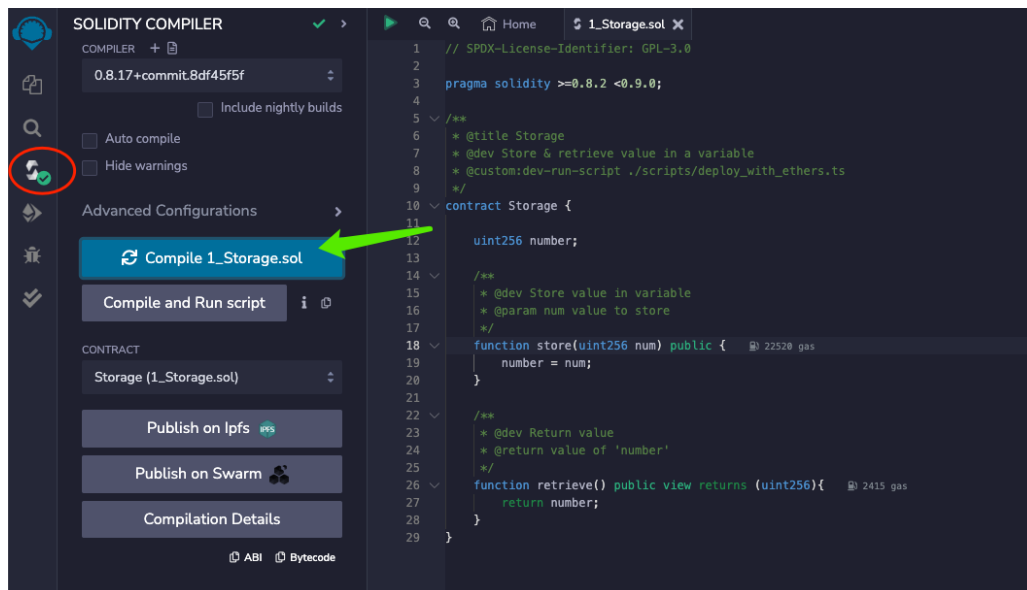


STEP 2: Open the sample contract (Storage.sol)

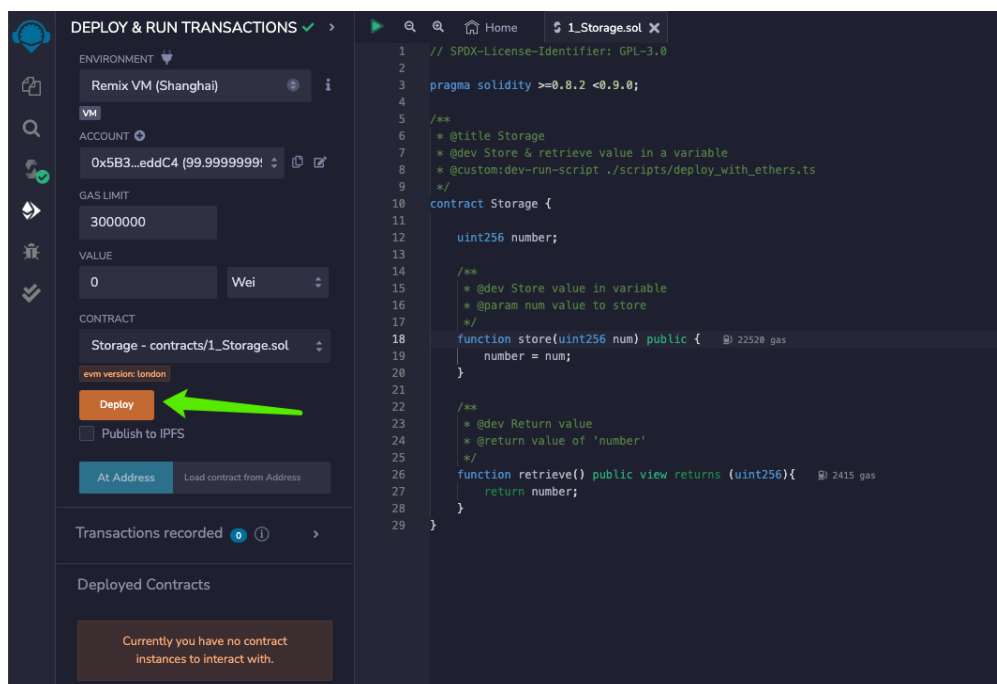


STEP 3: Compile the contract with the suitable solidity version.
You can find the solidity limitation on the top of the smart

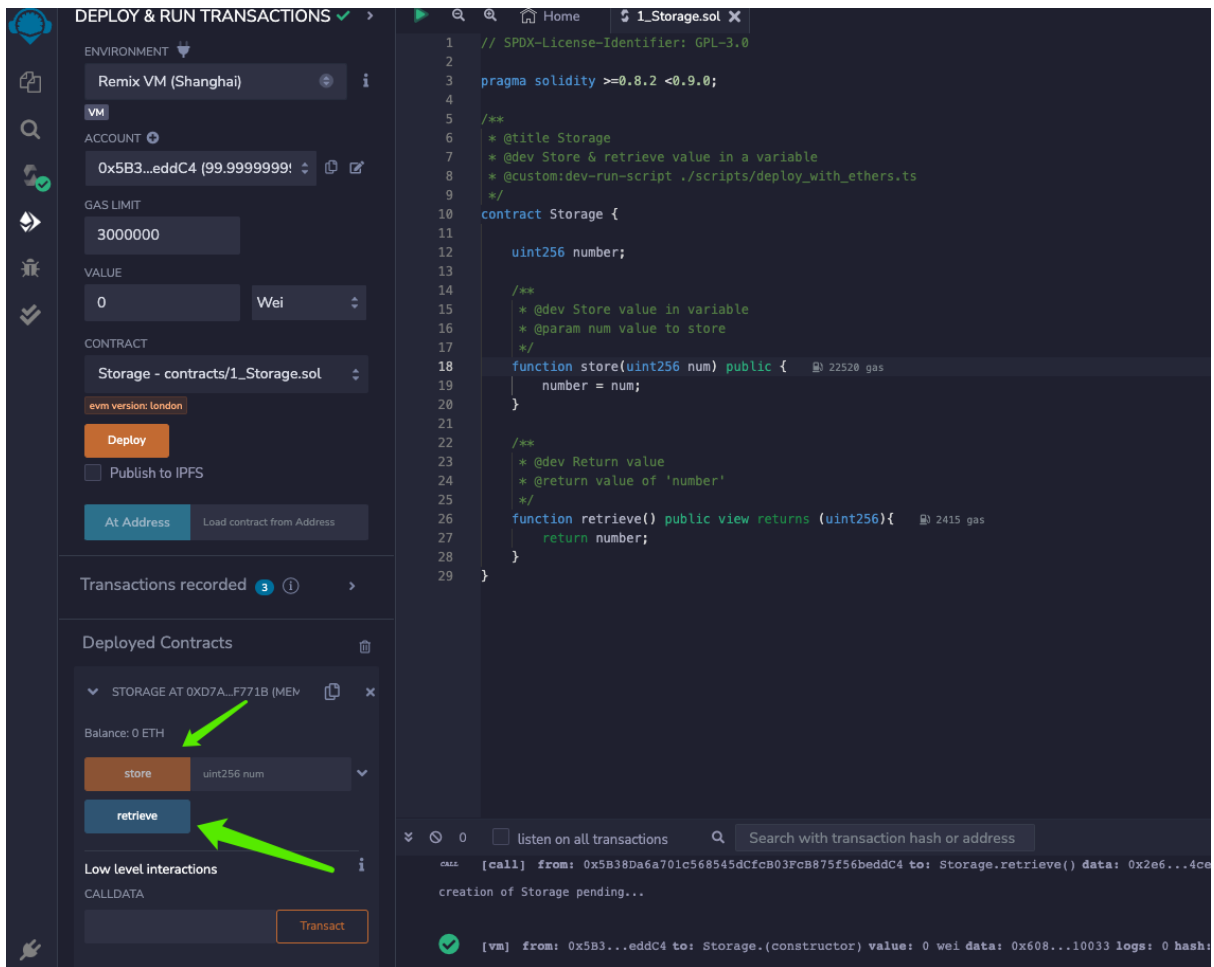
contract. Once compiling successfully, there will be a green tick on the left side.



STEP 4: Deploy the contract in the Remix editor.



STEP 5: Test functions and try to store and retrieve the value.



3. Value types in solidity (25 mins)

TYPE 1: Boolean Value

Boolean type is a binary variable that can take values of true or false.

```
// Boolean operators
bool public _bool1 = !_bool; // logical NOT
bool public _bool2 = _bool && _bool1; // logical AND
bool public _bool3 = _bool || _bool1; // logical OR
bool public _bool4 = _bool == _bool1; // equality
bool public _bool5 = _bool != _bool1; // inequality
```

TYPE 2: Integers Value

int: This is a signed integer type that can store both positive and negative numbers.

uint: This is an unsigned integer type that can only store non-negative whole numbers.

```
// Integer
int public _int = -1; // integers including negative numbers
uint public _uint = 1; // non-negative numbers
uint256 public _number = 20220330; // 256-bit positive integers
```

In addition, integer value can perform the arithmetic operations

```
// Integer operations
uint256 public _number1 = _number + 5; // +, -, *, /
uint256 public _number2 = 4**2; // Exponent
uint256 public _number3 = 9 % 4; // Modulo (Modulus)
bool public _numberbool = _number2 > _number3; // Great than
```

TYPE 3: Address Value

In Solidity, the address is a value type used to represent Ethereum addresses. They are typically 20 bytes in size and are usually represented in hexadecimal format.

- address: Holds a 20 byte value (size of an Ethereum address).
- address payable: Same as address, but with the additional members transfer and send to allow ETH transfers.

```
// Address
address public _address = 0x7A58c0Be72BE218B41C608b7Fe7C5bB630736C71;
address payable public _address1 = payable(_address); // payable address
(can transfer fund and check balance)
// Members of address
uint256 public balance = _address1.balance; // balance of address
```

TYPE 4: Enum Type

In Solidity, the enum type, short for "enumerable," is used to create a user-defined data type for a set of constant values.

```
// Let uint 0, 1, 2 represent Buy, Hold, Sell
enum ActionSet { Buy, Hold, Sell }
// Create an enum variable called action
ActionSet action = ActionSet.Buy;

// Enum can be converted into uint
function enumToUint() external view returns(uint){
    return uint(action);
}
```

Here is the sample code:

<https://codesandbox.io/p/sandbox/fastapi-demo-kkd6yv?file=/week7/section2-solidityDataTypes/valueType.sol:15,57>

4. Function Type (20 mins)

Here is the general format of the function in solidity

function <function name>(<parameter types>)
[internal|external] [pure|view|payable] [returns (<return types>)]

Visibility of the function

In Solidity, function visibility or function access specifies who can call and interact with a particular function within a smart contract

- public: Functions declared as public can be called from within the contract, from other contracts, and externally by anyone.
- internal: Functions declared as internal can only be called from within the current contract and derived contracts.
- private: Functions declared as private can only be called from within the current contract.
- external: Functions declared as external can only be called externally, typically from outside the contract.

```
function privateFunction() private pure returns (string memory) {  
    return "Private function";  
}  
  
function internalFunction() internal pure returns (string memory) {  
    return "Internal function";  
}  
  
function externalFunction() external pure returns (string memory) {  
    return "External function";  
}
```

Pure and View in solidity

These are additional modifiers used in combination with other visibility modifiers.

- Functions marked as **view** do not modify the state.
- Functions marked as **pure** do not read or modify the state.

They are often used with public and internal functions to specify that a function is read-only and doesn't change the state.

```
function getNumber(uint256 tmp) external pure returns(uint256 new_numberTest){
    new_numberTest = tmp + 1;
}

function viewFunction() external view returns (uint256) {
    return numberTest;
}
```

The sample code is here:

<https://codesandbox.io/p/sandbox/fastapi-demo-kkd6yv?file=/week7/section3-functionType/test.sol:44,2>

5. Mapping, Constructor and Modifier (20 mins)

What is Mapping:

In Solidity, a mapping is a data structure that is used to associate values (referred to as the "values" or "mapped values") with specific keys.

The format of declaring the mapping is mapping(_KeyType => _ValueType), where _KeyType and _ValueType are the variable types of Key and Value respectively. For example:

```

// Declare a mapping from addresses to integers
mapping(address => uint256) public balances;

// Function to set a balance for an address
function setBalance(address _address, uint256 _balance) public {
    balances[_address] = _balance;
}

// Function to get the balance for an address
function getBalance(address _address) public view returns (uint256) {
    return balances[_address];
}

```

What is Constructor:

In Solidity, a constructor is a special type of function that is automatically executed when a smart contract is deployed to the Ethereum blockchain. The constructor is used to initialize the state variables and perform any setup or configuration that is required for the contract's operation. Here is the basic structure and location of the constructor:

```

pragma solidity ^0.8.0;

contract Sample {
    // State variables

    // Constructor
    constructor() {
        // Initialization logic here
    }

    // Other functions
}

```

What is Modifier:

In Solidity, a modifier is a special keyword used to modify the behavior of functions. It allows you to apply some logic before or after the execution of a function, enabling code reuse and organization, thus enhancing the maintainability and readability of smart contracts. Typically, a modifier is used to perform condition checks before a function is executed to ensure that the function can only be called under specific conditions.

```
// Modifier definition
modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can call this function");
    _; // The '_' here signifies continuing with the function body
}
```

Here is the code sample:

<https://codesandbox.io/p/sandbox/fastapi-demo-kkd6yv?file=/week7/sec5/test.sol:26,2>

6. Reflection (5 mins)

Give students the opportunity to share what they learned, found interesting, or had difficulty understanding. Offer additional resources for them to learn more about FastAPI Python Framework

Online Code Sample:

<https://codesandbox.io/p/sandbox/fastapi-demo-kkd6yv?file=/week7/section1-data%20loading/App.js>