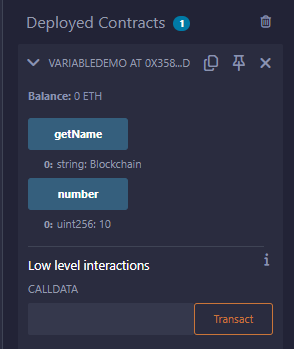
## Working with Variables

Demonstrate usage of state variables, local variables, and visibility modifiers.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract VariableDemo {  
 uint public number = 10;  
 string private name = "Blockchain";  
 bool internal status;  
  
 function getName() public view returns(string memory) {  
 return name;  
 }  
}

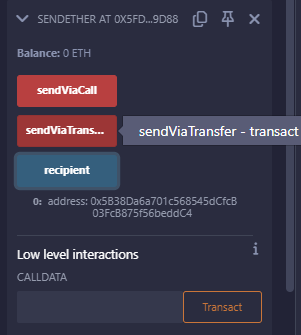


## Sending Money

Create a payable function and transfer Ether using `transfer` and `call`.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract SendEther {  
 address payable public recipient;  
  
 constructor(address payable \_recipient) {  
 recipient = \_recipient;  
 }  
  
 function sendViaTransfer() public payable {  
 recipient.transfer(msg.value);  
 }  
  
 function sendViaCall() public payable {  
 (bool sent, ) = recipient.call{value: msg.value}("");  
 require(sent, "Failed to send Ether");  
 }  
  
 receive() external payable {}  
}

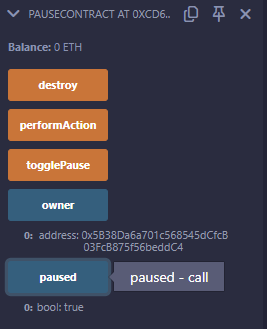


## Starting, Pausing, Stopping, and Deleting Transaction

Use modifiers and `selfdestruct` to manage contract state.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract PauseContract {  
 address public owner;  
 bool public paused;  
  
 constructor() {  
 owner = msg.sender;  
 }  
  
 modifier onlyOwner() {  
 require(msg.sender == owner, "Not owner");  
 \_;  
 }  
  
 modifier whenNotPaused() {  
 require(!paused, "Contract is paused");  
 \_;  
 }  
  
 function togglePause() public onlyOwner {  
 paused = !paused;  
 }  
  
 function destroy() public onlyOwner {  
 selfdestruct(payable(owner));  
 }  
  
 function performAction() public whenNotPaused {  
 // some logic  
 }  
}

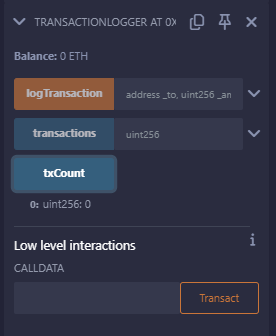


## Mapping and Struct

Define custom struct and mapping to store transaction records.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract TransactionLogger {  
 struct Transaction {  
 address from;  
 address to;  
 uint amount;  
 }  
  
 mapping(uint => Transaction) public transactions;  
 uint public txCount;  
  
 function logTransaction(address \_to, uint \_amount) public {  
 transactions[txCount] = Transaction(msg.sender, \_to, \_amount);  
 txCount++;  
 }  
}



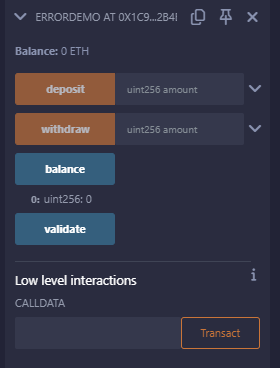
## Error Handling

Use `require`, `assert`, and `revert` for error control.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract ErrorDemo {  
 uint public balance;  
  
 function deposit(uint amount) public {  
 require(amount > 0, "Amount must be > 0");  
 balance += amount;  
 }  
  
 function validate() public view {  
 assert(balance >= 0);  
 }  
  
 function withdraw(uint amount) public {  
 if (amount > balance) {  
 revert("Insufficient balance");  
 }  
 balance -= amount;  
 }

}

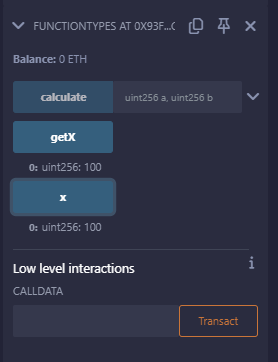


## View/Pure, Receive Function and Fallback Function

Demonstrate different function types and handling direct Ether sends.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract FunctionTypes {  
 uint public x = 100;  
  
 function getX() public view returns (uint) {  
 return x;  
 }  
  
 function calculate(uint a, uint b) public pure returns (uint) {  
 return a + b;  
 }  
  
 receive() external payable {}  
  
 fallback() external payable {}  
}



## Inheritance, Modifier, and Importing

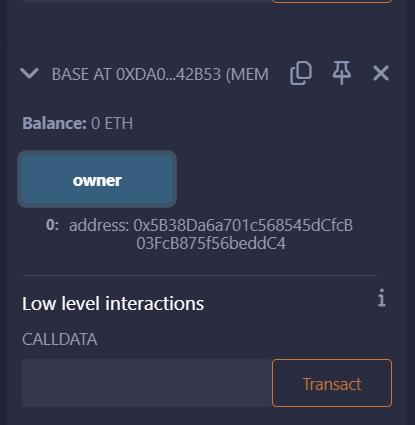
Use inheritance and import statement to structure contracts.

Sample Solidity Code:

// Base.sol  
pragma solidity ^0.8.0;  
  
contract Base {  
 address public owner;  
  
 constructor() {  
 owner = msg.sender;  
 }  
  
 modifier onlyOwner {  
 require(msg.sender == owner, "Not owner");  
 \_;  
 }  
}

//Derived.sol

pragma solidity ^0.8.0;  
import "./Base.sol";  
  
contract Derived is Base {  
 string public data;  
  
 function setData(string memory \_data) public onlyOwner {  
 data = \_data;  
 }  
}



## Events and Return Variables

Emit events and return multiple values from functions.

Sample Solidity Code:

pragma solidity ^0.8.0;  
  
contract EventExample {  
 event Transfer(address indexed from, address indexed to, uint amount);  
  
 function send(address to, uint amount) public returns (bool success, uint timestamp) {  
 emit Transfer(msg.sender, to, amount);  
 return (true, block.timestamp);  
 }  
}

