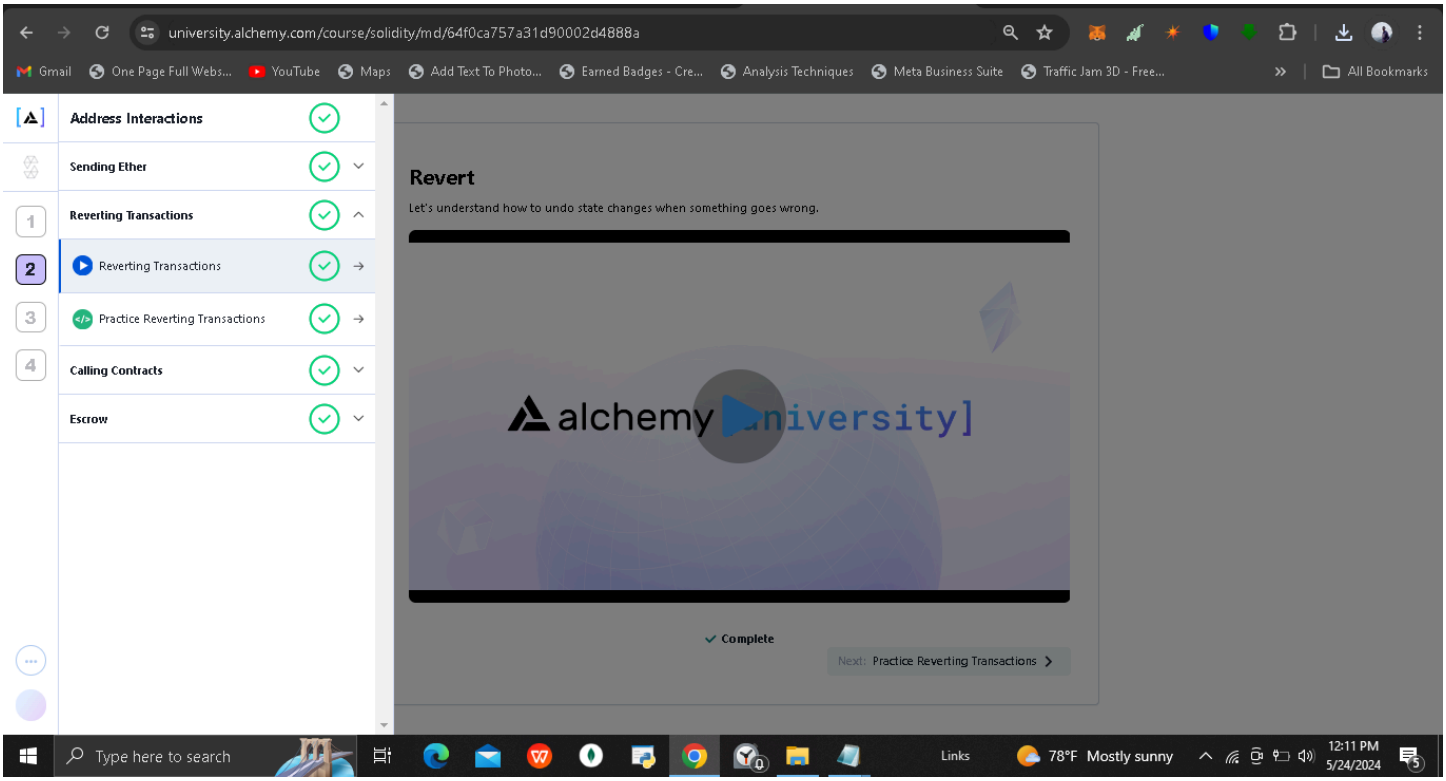
Lesson 2

Revert

- Constructor Revert
- Only Owner
- Owner Modifier



- Constructor Revert

Contract has been enhanced with additional functionality for withdrawing funds. Here's a breakdown:

1. **State Variables**:
   - owner: An address variable to store the owner of the contract.
2. **Error Definitions**:
   - DepositTooLow: An error to be thrown when the deposit during contract deployment is less than 1 ether.
   - Forbidden: An error to be thrown when someone other than the owner tries to withdraw funds.
3. **Constructor**:
   - The constructor is marked as payable, indicating it can receive Ether upon deployment.

- It checks if the value sent with the deployment transaction (msg.value) is less than 1 ether. If it is, the constructor reverts the transaction, throwing the DepositTooLow error.
- If the deposit is sufficient, the owner is set to the address of the deployer (msg.sender).

4. **withdraw() Function**:
   - This function allows the owner to withdraw the entire balance of the contract.
   - It first checks if the caller is the owner. If not, it reverts the transaction, throwing the Forbidden error.
   - If the caller is the owner, it attempts to transfer the entire balance of the contract to the owner's address.
   - It uses a low-level call to transfer Ether, and then checks if the call was successful using require.

Overall, this contract enforces that only the owner can withdraw funds and ensures that the contract is initialized with a minimum deposit of 1 ether during deployment. It's a solid foundation, but always remember to thoroughly test your contracts to ensure their functionality and security.

• Only Owner

contract looks well-structured and includes features for handling deposits, ownership, and withdrawals. Here's a summary of its key components:

1. **State Variables**:
   - owner: An address variable to store the owner of the contract.
2. **Error Definitions**:
   - DepositTooLow: An error to be thrown when the deposit during contract deployment is less than 1 ether.
   - Forbidden: An error to be thrown when someone other than the owner tries to withdraw funds.
3. **Constructor**:
   - The constructor is marked as payable, allowing it to receive Ether upon deployment.
   - It checks if the value sent with the deployment transaction (msg.value) is less than 1 ether. If it is, the constructor reverts the transaction, throwing the DepositTooLow error.
   - If the deposit is sufficient, the owner is set to the address of the deployer (msg.sender).
4. **withdraw() Function**:
   - This function allows the owner to withdraw the entire balance of the contract.
   - It checks if the caller is the owner. If not, it reverts the transaction, throwing the Forbidden error.
   - If the caller is the owner, it attempts to transfer the entire balance of the contract to the owner's address using a low-level call.
   - It checks if the call was successful using require.

Overall, your contract is structured logically and follows best practices. However, it's essential to thoroughly test it to ensure its functionality and security, especially in scenarios where reentrancy or other vulnerabilities might exist.

- ## Owner Modifier

Contract looks well-structured and implements access control using a modifier only Owner to restrict certain functions to be callable only by the contract owner. Here's a breakdown of your contract:

1. **State Variables**:
   - configA, configB, configC: These variables hold configuration values of type uint.
   - owner: This variable stores the address of the contract owner.
2. **Error Definition**:
   - Forbidden: An error to be thrown when a non-owner tries to access restricted functions.
3. **Constructor**:
   - The constructor initializes the owner variable with the address of the contract deployer (msg.sender).
4. **Functions**:
   - setA, setB, setC: These functions allow the contract owner to set the values of configA, configB, and configC respectively. Each function is marked as public and can only be called by the owner due to the onlyOwner modifier.
5. **Modifier**:
   - onlyOwner: This modifier restricts access to functions by allowing them to be called only by the contract owner. It checks if the sender of the transaction (msg.sender) is equal to the owner address. If not, it reverts the transaction, throwing the Forbidden error. Otherwise, it proceeds with the function execution.

Your contract effectively enforces access control, ensuring that only the contract owner can modify the configuration variables. This pattern is commonly used to secure critical functions and data within a smart contract. However, as always, remember to thoroughly test your contract to ensure its functionality and security.