

CORTEZ, LAWRENCE NEIL M.  
NW-301

## Constants

The screenshot displays the Remix IDE interface with the following components:

- Left Sidebar (DEPLOY & RUN TRANSACTIONS):**
  - GAS LIMIT:** Estimated Gas, Custom 3000000.
  - VALUE:** 0 Wei.
  - CONTRACT:** Constants - contracts/Constants.sol.
  - Deploy:** Button to deploy the contract.
  - At Address:** Button to load contract from address.
  - Transactions recorded:** 1.
  - Deployed Contracts:** Section showing the deployed contract 'CONSTANTS AT 0x0A...F771E' with a balance of 0 ETH.
  - Low level interactions:** Section showing 'CALLDATA' with a 'Transaction' button.
- Central Editor:** Contains the Solidity code for the 'Constants' contract:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4 contract Constants {
5     address public constant MY_ADDRESS = 0x777788889999AaAbBbCcCcdDDeeEFFFfCcCc;
6     uint public constant MY_UNIT = 123;
7 }
8
9 contract Var {
10     address public MY_ADDRESS = 0x777788889999AaAbBbCcCcdDDeeEFFFfCcCc;
11 }
```
- Right Sidebar (REMIXAI ASSISTANT):**
  - RemixAI:** Logo and text: 'RemixAI provides you personalized guidance as you build. It can break down concepts, answer questions about blockchain technology and assist you with your smart contracts.'
  - Buttons:** 'How do NFTs work?', 'Show a contract that includes a flash loan', 'Cross-chain messaging protocols?'
  - AI copilot:** Section with 'Explain contract' and 'Listen on all transactions' options.
  - Select Context:** Button to select a context.
  - Select context and ask me anything!:** Text input field.
  - MistralAI:** Button to select MistralAI.
  - Audio Prompt:** Button to use audio prompts.
  - Create new workspace with AI:** Button to create a new workspace.

The bottom status bar shows the system time as 1:31 pm on 25/11/2023.

CORTEZ, LAWRENCE NEIL M.  
NW-301

## Mapping

The screenshot displays the Remix IDE interface with a Solidity contract named `MappingExample` open in the editor. The contract is written in Solidity 0.8.20 and includes a mapping, a non-zero value check, a value setting function, and a value retrieval function.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4
5 contract MappingExample {
6     mapping(address => uint256) public valueMapping;
7
8     modifier nonZeroValue(uint256 _value) {
9         require(_value != 0, "Value cannot be zero");
10     }
11
12
13     modifier valueHasBeenSet() {
14         require(valueMapping[msg.sender] != 0, "No value set for sender");
15     }
16
17
18
19     function setValue(uint256 _value) payable {
20         public
21         nonZeroValue(_value)
22         {
23             valueMapping[msg.sender] = _value;
24         }
25     }
26
27
28     function getValue() public view {
29         return valueMapping[msg.sender];
30     }
31 }
32
```

The left sidebar shows the **DEPLOY & RUN TRANSACTIONS** panel with a custom gas limit of 3,000,000 and a value of 0 Wei. The **Deployed Contracts** section shows the `MappingExample` contract deployed at address `0x5B3...e0dC4`. The **Low level interactions** section shows the `CALLDATA` field.

The right sidebar features the **REMIXAI ASSISTANT** panel, which provides personalized guidance and includes buttons for `How do NFTs work?`, `Show a contract that includes a flash loan`, and `Cross-chain messaging protocols?`.

The bottom status bar indicates the current language is **ENG** and the time is **1:30 pm** on **25/11/2025**.

CORTEZ, LAWRENCE NEIL M.  
NW-301

## ErrorHandling

The screenshot displays the Remix IDE interface, version 1.2.0, with a workspace named 'default\_workspace'. The central editor shows a Solidity contract named 'MappingExample' in 'contracts/Constants.sol'. The contract includes a mapping and two functions: 'setValue' and 'getValue'. The 'setValue' function checks if the value is zero and returns an error message if it is. The 'getValue' function returns the value from the mapping or a default message if the key is not present.

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4 contract MappingExample {
5     mapping(address => uint256) public valueMapping;
6
7     function setValue(uint256 _value) public {
8         require(_value != 0, "Value cannot be zero");
9         valueMapping[msg.sender] = _value;
10    }
11
12    function getValue() public view returns (uint256) {
13        require(valueMapping[msg.sender] != 0, "No value set for sender");
14        return valueMapping[msg.sender];
15    }
16 }
17
```

On the left sidebar, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing a custom deployment with a gas limit of 300,000. Below this, the 'MappingExample - contracts/Errors.sol' contract is selected, and the 'Deploy' button is visible. The 'Transactions recorded' section shows one transaction, and the 'Deployed Contracts' section lists the 'MappingExample' contract at address 0x000...e003.

At the bottom, the 'AI copilot' panel is open, displaying a message: '[msg] from: 0x583...e0dCA to: MappingExample.constructor value: 0 wei data: 0x000...e003 logs: 0 hash: 0x104...E9036'. The 'Debug' button is visible next to the message.

On the right, the 'REMIXAI ASSISTANT' panel is active, featuring the RemixAI logo and a description: 'RemixAI provides you personalized guidance as you build. It can break down concepts, answer questions about blockchain technology, and assist you with your smart contracts.' Below this, there are three buttons: 'How do NFIs work?', 'Show a contract that includes a flash loan', and 'Cross-chain messaging protocols?'. At the bottom of the assistant panel, there is a 'Select Context' dropdown, a 'Ask' button, and a 'Create new workspace with AI' button.

CORTEZ, LAWRENCE NEIL M.  
NW-301

## FunctionModifier

The screenshot displays the Remix IDE interface, version 1.2.0, with a workspace named 'default\_workspace'. The central editor shows a Solidity contract named 'MappingExample' with the following code:

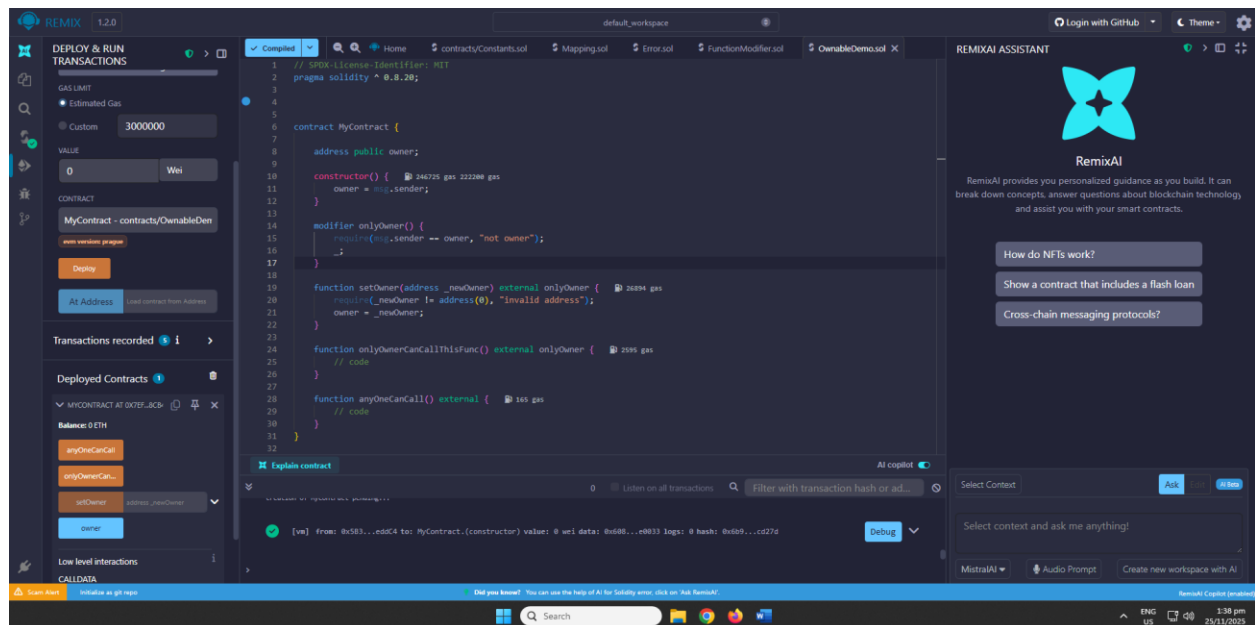
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4
5 contract MappingExample {
6     mapping(address => uint256) public valueMapping;
7
8     modifier nonZeroValue(uint256 _value) {
9         require(_value != 0, "Value cannot be zero");
10    }
11
12
13     modifier valueHasBeenSet() {
14         require(valueMapping[msg.sender] != 0, "No value set for sender");
15    }
16
17
18
19
20     function setValue(uint256 _value) @ 22658 gas
21     public
22     nonZeroValue(_value)
23     {
24         valueMapping[msg.sender] = _value;
25     }
26
27
28     function getValue() @ 4784 gas
29     public
30     view
31     valueHasBeenSet
32     returns (uint256)
```

The left sidebar contains the 'DEPLOY & RUN TRANSACTIONS' panel, showing a custom deployment with a value of 3000000 Wei. It also displays a list of deployed contracts, including 'MappingExample' at address 0x091... and a transaction record for 'setValue'.

The right sidebar features the 'REMIXAI ASSISTANT' panel, which includes a logo, a description of the assistant's capabilities, and buttons for 'How do NFTs work?', 'Show a contract that includes a flash loan', and 'Cross-chain messaging protocols?'. Below this is a 'Select Context' dropdown and a 'Select context and ask me anything!' input field.

The bottom status bar shows the current transaction details: '[vm] from: 0x583...ed8c4 to: MappingExample.(constructor) value: 0 wei data: 0x088...e0b33 logs: 0 hash: 0x055...8cab'. The system tray at the bottom indicates the time as 1:29 pm on 25/11/2025.

## OwnableDemo



- **Function Modifiers & Ownable**

*When should you use a modifier like onlyOwner instead of inline checks, and what risks arise if ownership isn't managed properly?*

**You should use a modifier like onlyOwner when the same access control logic is applied to multiple functions. Modifiers improve readability, reusability, and maintainability compared to repeating inline require checks in each function. If ownership isn't managed properly—such as allowing an invalid address to become owner, or failing to restrict critical functions—unauthorized users could gain control, leading to loss of funds, contract misuse, or security vulnerabilities.**

- **Error Handling**

*How do you choose between require, revert, and assert, and why might custom errors be better than error strings?*

I choose between require, revert, and assert based on the type of condition I am checking. I use require for validating inputs or external conditions, because it stops execution and refunds remaining gas. Revert is used when I want to explicitly terminate execution in more complex scenarios. Assert is reserved for internal invariants that should never fail; failing an assert consumes all remaining gas. Custom errors are often better than string messages because they are more gas-efficient—they encode the error type instead of storing a string in memory, which saves cost on deployment and frequent function calls.

- **Constants & Variables**

*When should a value be constant, immutable, or mutable, and how does that choice affect gas cost and flexibility?*

I use constant for values that will never change, as they are stored in bytecode and are very gas-efficient. I use immutable for values that are set once during contract deployment and cannot change afterward, offering a balance between flexibility and gas savings. Regular mutable variables are used when the value needs to change over time, but they are more expensive in terms of gas because they are stored in contract storage. Choosing the right type ensures a balance between efficiency, flexibility, and contract functionality.

GITHUB LINK:

<https://github.com/L-Cortez/BLOCKTECH/tree/main/Prelim%20Lab%20Activity%202%20Solidity%20Basics%20II>

CORTEZ, LAWRENCE NEIL M.  
NW-301