

CORTEZ, LAWRENCE NEIL M.
NW-301

Constants

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is visible, showing a GAS LIMIT of 3000000 and a CONTRACT named 'Constants - contracts/Constants.sol'. Below it, 'Transactions recorded' and 'Deployed Contracts' sections are shown. In the center, the code editor displays the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Constants {
    address public constant MY_ADDRESS = 0x77778888999AaAbBbCcCcddDdeeEFFFcCc;
    uint public constant MY_UNIT = 123;
}

contract Var {
    address public MY_ADDRESS = 0x77778888999AaAbBbCcCcddDdeeEFFFcCc;
}
```

On the right, the 'REMIXAI ASSISTANT' panel features a teal flower logo and the 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.' Below this are three buttons: 'How do NFTs work?', 'Show a contract that includes a flash loan', and 'Cross-chain messaging protocols?'. At the bottom of the interface, there's a footer bar with various icons and language settings.

CORTEZ, LAWRENCE NEIL M.
NW-301

Mapping

The screenshot shows the Remix IDE interface with the following details:

- Deploy & Run Transactions:** Set to "Custom" with a value of 3000000 Wei.
- Contract:** MappingExample - contracts/Function
- Transactions recorded:** 1
- Deployed Contracts:** MAPPINGEXAMPLE AT 0x583...ed0C4
- Balances:** 0 ETH
- Low level interactions:** CALLDATA
- Code:**

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.20;
4
5
6 contract MappingExample {
7     mapping(address => uint256) public valueMapping;
8
9     modifier nonzeroValue(uint256 _value) {
10         require(_value != 0, "Value cannot be zero");
11        _;
12     }
13
14     modifier valueHasBeenSet() {
15         require(valueMapping[_msg.sender] != 0, "No value set for sender");
16        _;
17     }
18
19     function setValue(uint256 _value) public payable {
20         nonzeroValue(_value);
21
22         valueMapping[_msg.sender] = _value;
23     }
24
25     function getValue() public view returns (uint256) {
26         return valueMapping[_msg.sender];
27     }
28
29 }
```
- REMIXAI ASSISTANT:** Provides personalized guidance about blockchain technology.
- Buttons:** How do NFTs work?, Show a contract that includes a flash loan, Cross-chain messaging protocols?
- Bottom Status:** RemixIDE v1.2.0, Search bar, Browser icons, ENG US, 1:30 pm, 20/12/2023, RemixIDE Credits (remix.ethereum.org)

CORTEZ, LAWRENCE NEIL M.
NW-301

ErrorHandling

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel displays a transaction configuration with 'Custom' selected, '3000000' as the value, and 'Wei' as the unit. Below it, the 'CONTRACT' section shows the deployed contract 'MappingExample' from the file 'contracts/Error.sol'. The code editor on the right contains the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract MappingExample {
    mapping(address => uint256) public valueMapping;

    function setValue(uint256 value) public {
        require(value != 0, "Value cannot be zero");
        valueMapping[msg.sender] = value;
    }

    function getValue() public view returns (uint256) {
        require(valueMapping[msg.sender] != 0, "No value set for sender");
        return valueMapping[msg.sender];
    }
}
```

The RemixAI Assistant panel on the right provides personalized guidance for blockchain technology, including links to 'How do NFTs work?', 'Show a contract that includes a flash loan', and 'Cross-chain messaging protocols?'. It also features an AI copilot for explaining contracts and a context selection dropdown.

CORTEZ, LAWRENCE NEIL M.
NW-301

FunctionModifier

The screenshot shows the Remix IDE interface with the following details:

- Contract Name:** MappingExample
- Compiler:** default_workspace, Version: 0.8.20
- Deployment:** Deployed at address 0x5d... (0x5d8...eddc4)
- Balance:** 0 ETH
- Low-level interactions:** CALLDATA
- Code (Excerpt):**

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

- REMIXAI ASSISTANT:** Provides personalized guidance, including links to NFTs, flash loans, and cross-chain messaging.
- Bottom Bar:** Includes a search bar, system icons (ENG US), and a timestamp (1:29 pm 2/5/2023).

OwnableDemo

The screenshot shows the Remix IDE interface with the following details:

- Contract Name:** OwnableDemo.sol
- Gas Limit:** Estimated Gas (3000000)
- Value:** 0 Wei
- Deployed Contracts:** MYCONTRACT at 0x7EF...BCB (0 ETH balance). Functions listed: anyOneCanCall, onlyOwnerCan..., setOwner, owner.
- Low level interactions:** CALLDATA tab.
- RemixAI Assistant:** Provides personalized guidance, including links to "How do NFTs work?", "Show a contract that includes a flash loan", and "Cross-chain messaging protocols".
- Bottom Status Bar:** Shows network (ENG US), battery, and date (25/11/2025).

The code editor contains the following Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;

contract MyContract {
    address public owner;
    constructor() {
        owner = msg.sender;
    }
    modifier onlyOwner() {
        require(msg.sender == owner, "not owner");
       _;
    }
    function setOwner(address _newOwner) external onlyOwner {
        require(_newOwner != address(0), "invalid address");
        owner = _newOwner;
    }
    function onlyOwnerCanCallThisFunc() external onlyOwner {
        // code
    }
    function anyOneCanCall() external {
        // code
    }
}
```

- **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