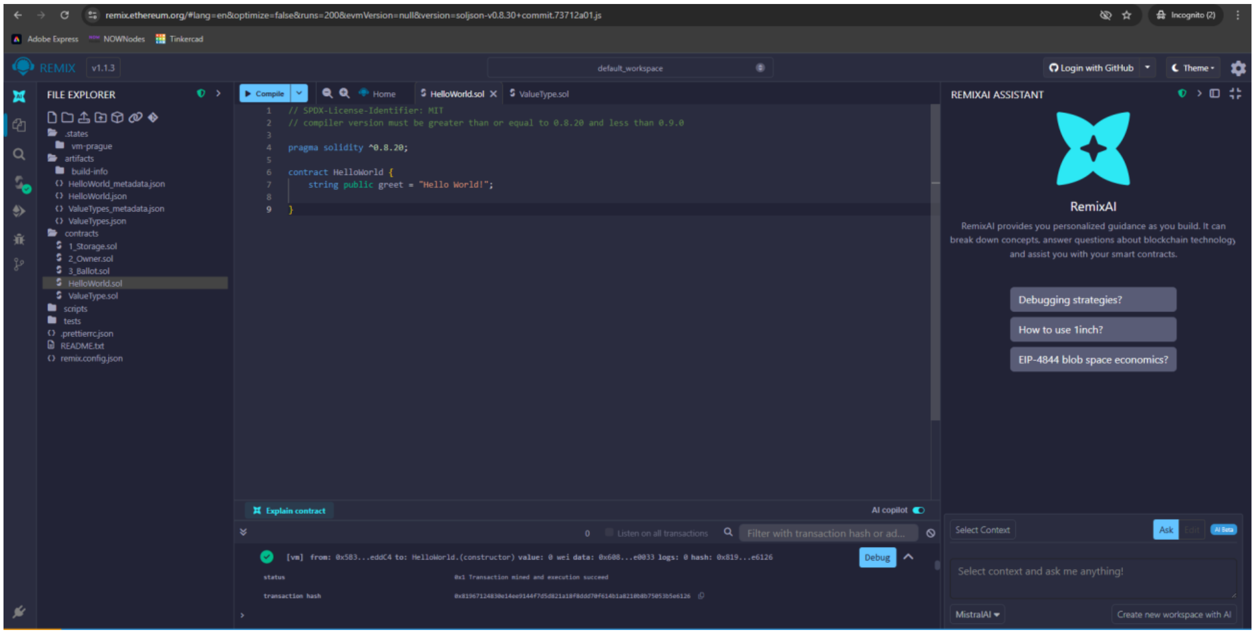
Cortez, Lawrence Neil M.
NW-301

HelloWorld.sol



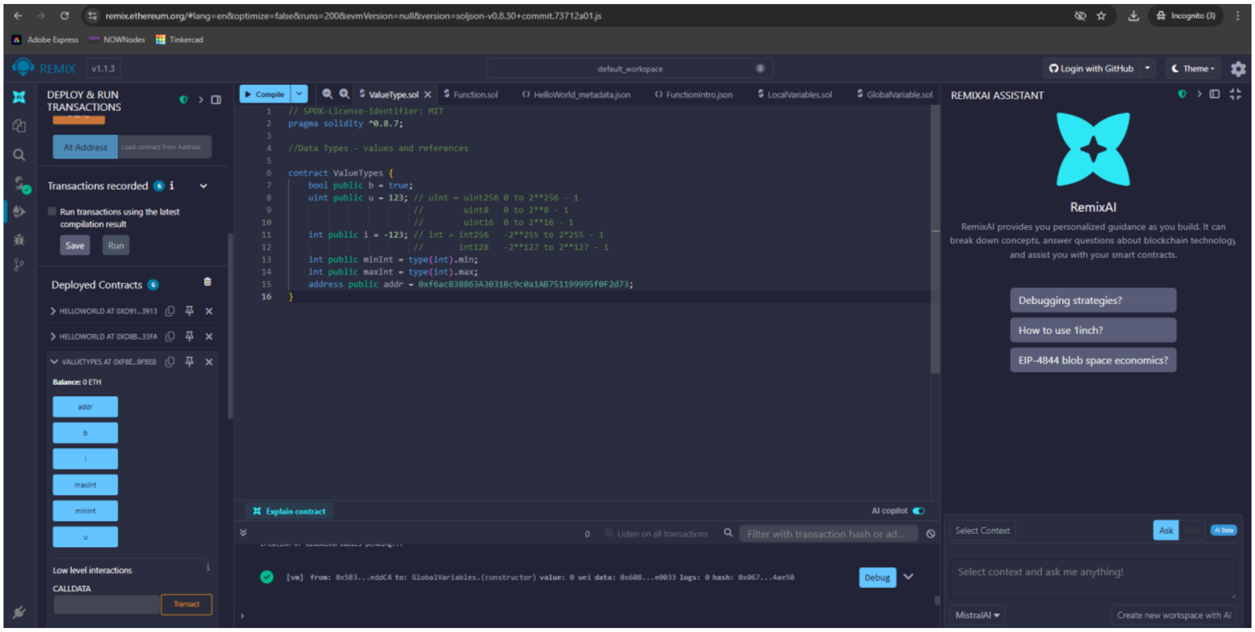
```solidity
// SPDX-License-Identifier: MIT
// compiler version must be greater than or equal to 0.8.20 and less than 0.9.0

pragma solidity ^0.8.20;

contract HelloWorld {
    string public greet = "Hello World!";

}
```

```
[vm] from: 0x5B3...eddC4 to: HelloWorld.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0x819...e6126
status                    0x1 Transaction mined and execution succeed
transaction hash          0x81967124830e14ee9144f7d5d821a18f8ddd70f614b1a8210b8b75053b5e6126
```

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.

Cortez, Lawrence Neil M.
NW-301

ValueType.sol


remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.30+commit.73712a01.js
default_workspace
DEPLOY & RUN TRANSACTIONS
At Address
Transactions recorded
Run transactions using the latest compilation result
Save
Run
Deployed Contracts
HELLOWORLD AT 0XD91...3913
HELLOWORLD AT 0XD6B...33FA
VALUETYPES AT 0XF6E...9FBEB
Balance: 0 ETH
addr
b
i
maxInt
minInt
u
Low level interactions
CALLDATA
Transact
Compile
ValueType.sol
Function.sol
HelloWorld_metadata.json
FunctionIntro.json
LocalVariables.sol
GlobalVariable.sol
Explain contract
AI copilot
Debug
REMIXAI ASSISTANT
RemixAI
Debugging strategies?
How to use 1inch?
EIP-4844 blob space economics?
Ask
Select context and ask me anything!
MistralAI
Create new workspace with AI

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

//Data Types - values and references

contract ValueTypes {
    bool public b = true;
    uint public u = 123; // uint = uint256 0 to 2**256 - 1
                         //        uint8   0 to 2**8 - 1
                         //        uint16  0 to 2**16 - 1
    int public i = -123; // int = int256   -2**255 to 2*255 - 1
                         //       int128   -2**127 to 2**127 - 1
    int public minInt = type(int).min;
    int public maxInt = type(int).max;
    address public addr = 0xf6ac838863A3031Bc9c0a1AB751199995f0F2d73;
}
```

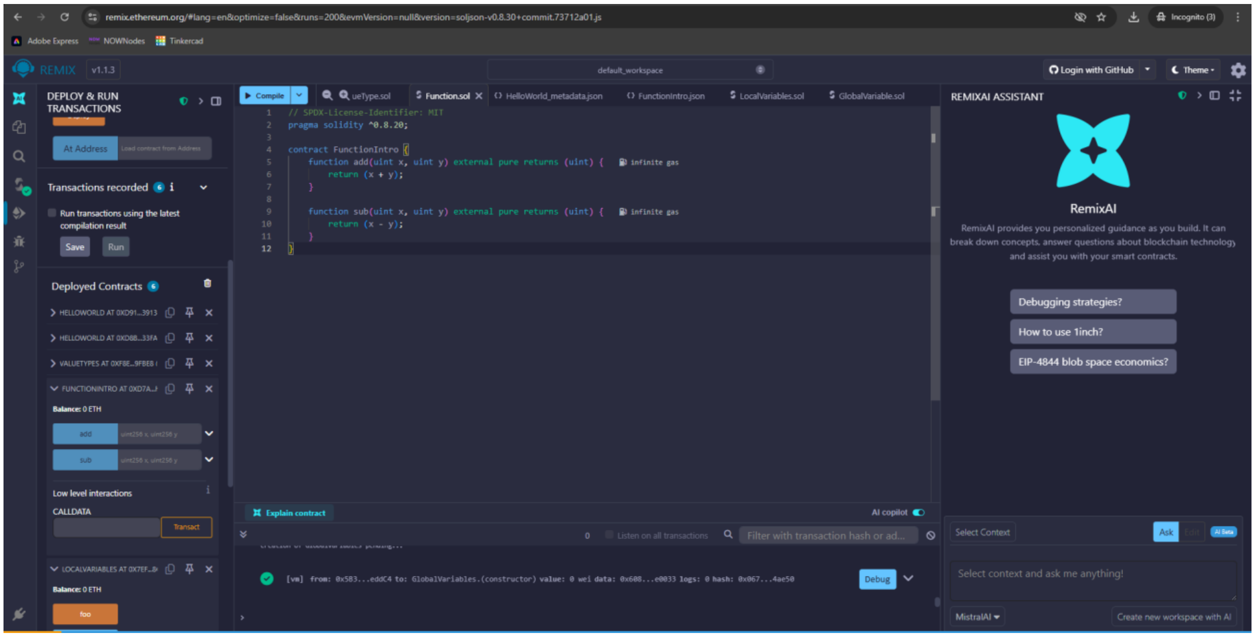
```
[vm] from: 0x5B3...eddC4 to: GlobalVariables.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0x067...4ae50
```

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.

I gained a better understanding of Solidity's handling of various value types by looking over this code. I gained a better understanding of how data is stored and constrained by its type by seeing variables like bool, uint, int, and address declared with real-world examples. I also realized how crucial it is to select the appropriate size when working with numbers in smart contracts after seeing the breakdown of ranges for uint and int. I was able to see how Solidity allows you to directly access the boundaries of a data type by looking at type(int).min and type(int).max. All things considered, this straightforward contract strengthened my grasp of the fundamental data types that operate on the Ethereum blockchain.

Cortez, Lawrence Neil M.
NW-301

Functions.sol



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

contract FunctionIntro {
    function add(uint x, uint y) external pure returns (uint) {
        return (x + y);
    }

    function sub(uint x, uint y) external pure returns (uint) {
        return (x - y);
    }
}
```

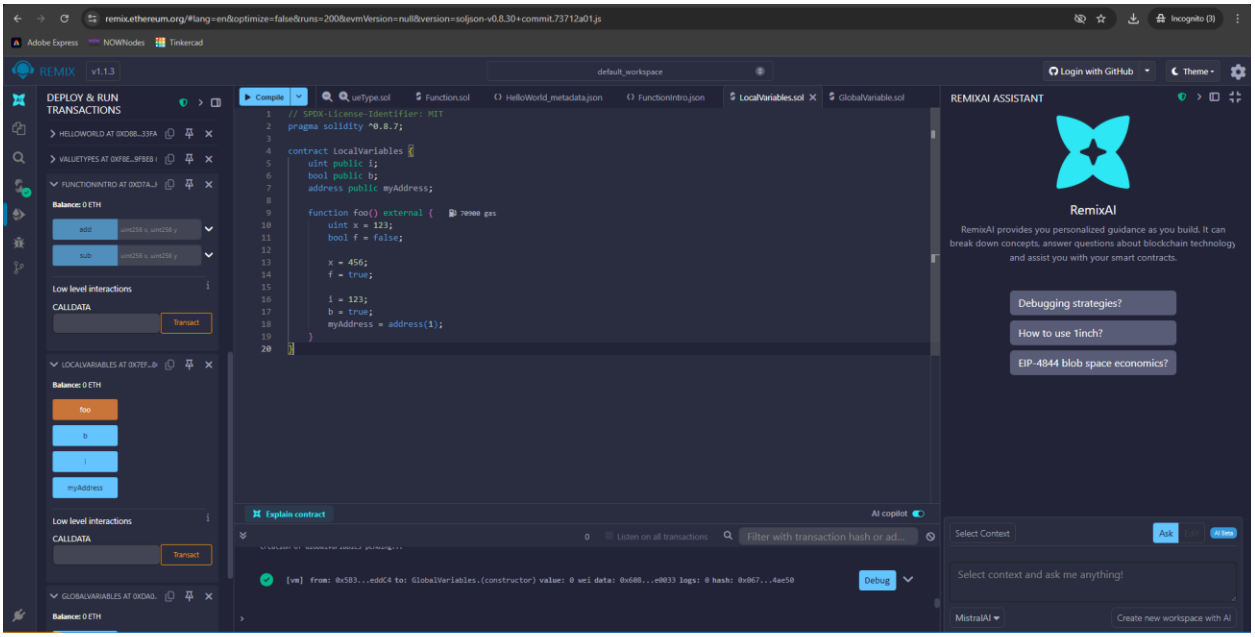
```
[vm] from: 0x5B3...eddC4 to: GlobalVariables.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0x067...4ae50
```

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.30+commit.73712a01.js
REMIX v1.1.3
default_workspace
Login with GitHub
Theme
DEPLOY & RUN TRANSACTIONS
At Address
Load contract from Address
Transactions recorded 6
Run transactions using the latest compilation result
Save
Run
Deployed Contracts 6
HELLOWORLD AT 0XD91...3913
HELLOWORLD AT 0XD8B...33FA
VALUETYPES AT 0XF8E...9FBE8
FUNCTIONINTRO AT 0XD7A...
Balance: 0 ETH
add
uint256 x, uint256 y
sub
uint256 x, uint256 y
Low level interactions
CALLDATA
Transact
LOCALVARIABLES AT 0X7EF...
Balance: 0 ETH
foo
Compile
ueType.sol
Function.sol
HelloWorld_metadata.json
FunctionIntro.json
LocalVariables.sol
GlobalVariable.sol
infinite gas
infinite gas
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
Debug
REMIXAI ASSISTANT
RemixAI
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.
Debugging strategies?
How to use 1inch?
EIP-4844 blob space economics?
Select Context
Ask
Select context and ask me anything!
MistralAI
Create new workspace with AI

Working with this code gave me a clearer understanding of how basic Solidity functions operate. The simple add and sub functions helped me see how inputs and return values are handled, and why using keywords like external and pure matters. Since these functions don’t change or read any blockchain data, marking them as pure makes the contract more efficient. Overall, this small example showed me that even very simple operations can teach important concepts about how smart contracts are structured and how they behave on the Ethereum network.

Cortez, Lawrence Neil M.
NW-301

LocalVariable



remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.30+commit.73712a01.js
REMIX v1.1.3
default_workspace
Login with GitHub
Theme
DEPLOY & RUN TRANSACTIONS
Compile
ueType.sol
Function.sol
HelloWorld_metadata.json
FunctionIntro.json
LocalVariables.sol
GlobalVariable.sol
REMIXAI ASSISTANT
RemixAI
Debugging strategies?
How to use 1inch?
EIP-4844 blob space economics?
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
Debug
Select Context
Ask
Select context and ask me anything!
MistralAI
Create new workspace with AI

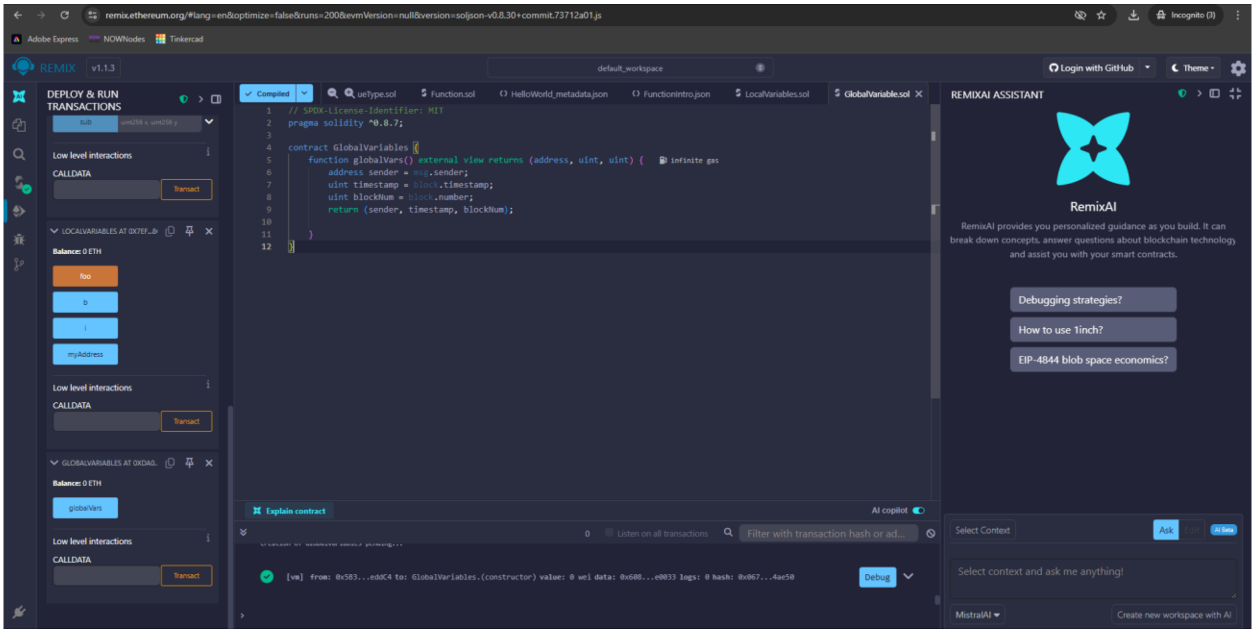
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract LocalVariables {
    uint public i;
    bool public b;
    address public myAddress;

    function foo() external {    70900 gas
        uint x = 123;
        bool f = false;

        x = 456;
        f = true;

        i = 123;
        b = true;
        myAddress = address(1);
    }
}
```

```
[vm] from: 0x5B3...eddC4 to: GlobalVariables.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0x067...4ae50
```

I was better able to comprehend the distinction between Solidity's local and state variables after looking at this contract. The function foo() demonstrates how local variables, such as x and f, are only present inside the function and can be altered without influencing the data that is stored on the blockchain. On the other hand, when the state variables i, b, and myAddress are updated, they are actually saved to the contract's storage. The idea was much simpler to understand after witnessing this division in action. It helped me understand how Solidity handles temporary versus permanent data and how choosing the right one can impact smart contract behavior and gas prices.

Cortez, Lawrence Neil M.
NW-301

GlobalVariable



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract GlobalVariables {
    function globalVars() external view returns (address, uint, uint) {
        address sender = msg.sender;
        uint timestamp = block.timestamp;
        uint blockNum = block.number;
        return (sender, timestamp, blockNum);

    }
}
```

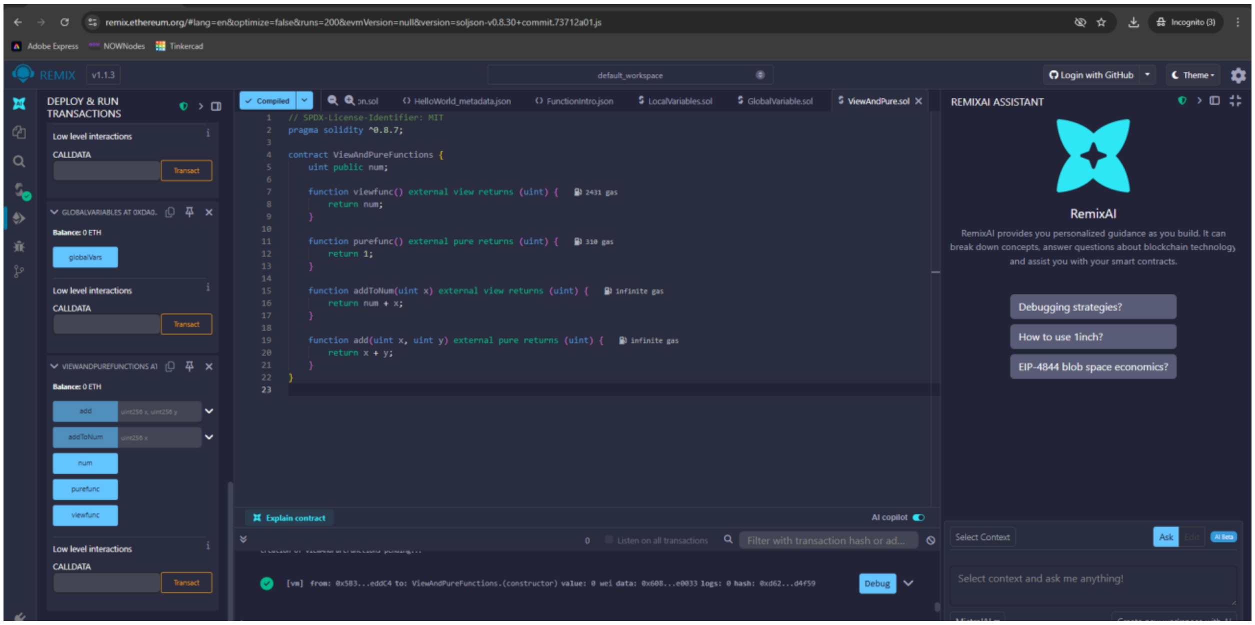
```
[vm] from: 0x5B3...eddC4 to: GlobalVariables.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0x067...4ae50
```

remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.30+commit.73712a01.js
REMIX v1.1.3
default_workspace
DEPLOY & RUN TRANSACTIONS
Compiled
ueType.sol
Function.sol
HelloWorld_metadata.json
FunctionIntro.json
LocalVariables.sol
GlobalVariable.sol
infinite gas
Low level interactions
CALLDATA
Transact
LOCALVARIABLES AT 0X7EF...
Balance: 0 ETH
foo
b
i
myAddress
GLOBALVARIABLES AT 0XDA0...
globalVars
Explain contract
AI copilot
Listen on all transactions
Filter with transaction hash or ad...
Debug
REMIXAI ASSISTANT
RemixAI
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.
Debugging strategies?
How to use 1inch?
EIP-4844 blob space economics?
Select Context
Ask
Select context and ask me anything!
MistralAI
Create new workspace with AI
Login with GitHub
Theme

Studying this contract helped me understand how Solidity gives access to global variables that show information about the blockchain environment. By using msg.sender, block.timestamp, and block.number, I realized how a smart contract can automatically know who is interacting with it, when the transaction occurs, and which block it is part of. Seeing these values returned from the globalVars() function made the concept clearer for me. It showed me how smart contracts can respond to real-time blockchain data, and it helped me see how powerful and connected these built-in global variables are.

Cortez, Lawrence Neil M.
NW-301

View and Pure



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract ViewAndPureFunctions {
    uint public num;

    function viewfunc() external view returns (uint) {
        return num;
    }

    function purefunc() external pure returns (uint) {
        return 1;
    }

    function addToNum(uint x) external view returns (uint) {
        return num + x;
    }

    function add(uint x, uint y) external pure returns (uint) {
        return x + y;
    }
}
```

```
[vm] from: 0x5B3...eddC4 to: ViewAndPureFunctions.(constructor) value: 0 wei data: 0x608...e0033 logs: 0 hash: 0xd62...d4f59
```

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.

Working on this contract helped me understand the difference between **view** and **pure** functions in Solidity. By fixing the errors and seeing how each function interacts with state variables, I learned why view functions can read state and why pure functions must stay independent of it. Correcting the syntax also reminded me how important proper naming and structure are in Solidity. Overall, this exercise strengthened my understanding of how smart contracts handle calculations and data access.

Cortez, Lawrence Neil M.
NW-301

IDENTIFY IF VIEW OR PURE:

```
function add(uint a, b) public ? returns (uint) {

        return a + b;

}
```

- ✓ **PURE**

```
function getBalance(address account) public ? returns (uint) {

        return account.balance;

}
```

- ✓ **VIEW**

```
function multiply(uint a, uint b) internal ? returns (uint) {

        return a * b;

}
```

- ✓ **PURE**