Practical:1

Aim:

To study the layout and working Remix Integrated Development Environment (IDE) for Ethereum Blockchain.

Software Used

1. IDE remix (0.51.0)
2. Windows (11)
3. Solidity (0.8.17)

Description

1. Explain in general the aim of this experiment.

The aim is to understand the layout and functionality of Remix Integrated Development Environment (IDE) for developing and deploying smart contracts in Solidity on the Ethereum blockchain.

2. Write step-wise description of how you conducted experiment.

Open a web browser and navigate to the official Remix IDE website at remix.ethereum.org.

Familiarize yourself with the file explorer, editor, and terminal/output panel.then create your own file with the extension .sol file.

In the editor, write a basic solidity contract, such as:

//SPDX-License_Identifier:MIT
pragma solidity 0.8.17;

For deploy the contract Go to the "Deploy & Run Transactions" tab.Select the environment Ensure Experiment1 is selected.Enter constructor parameters if needed.Click "Deploy".

Use the "Debugger" tab to step through execution and identify issues.

Save your code and settings.Export the project using the "Files" tab for backup or further development.

3. Code of this experiment with complete explanation in form of comments.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;// This specifies the license for the code, ensuring it's open source and free to use. pragma solidity 0.8.17;
```

4. Snapshot with caption of the of the code and output in the platform used.

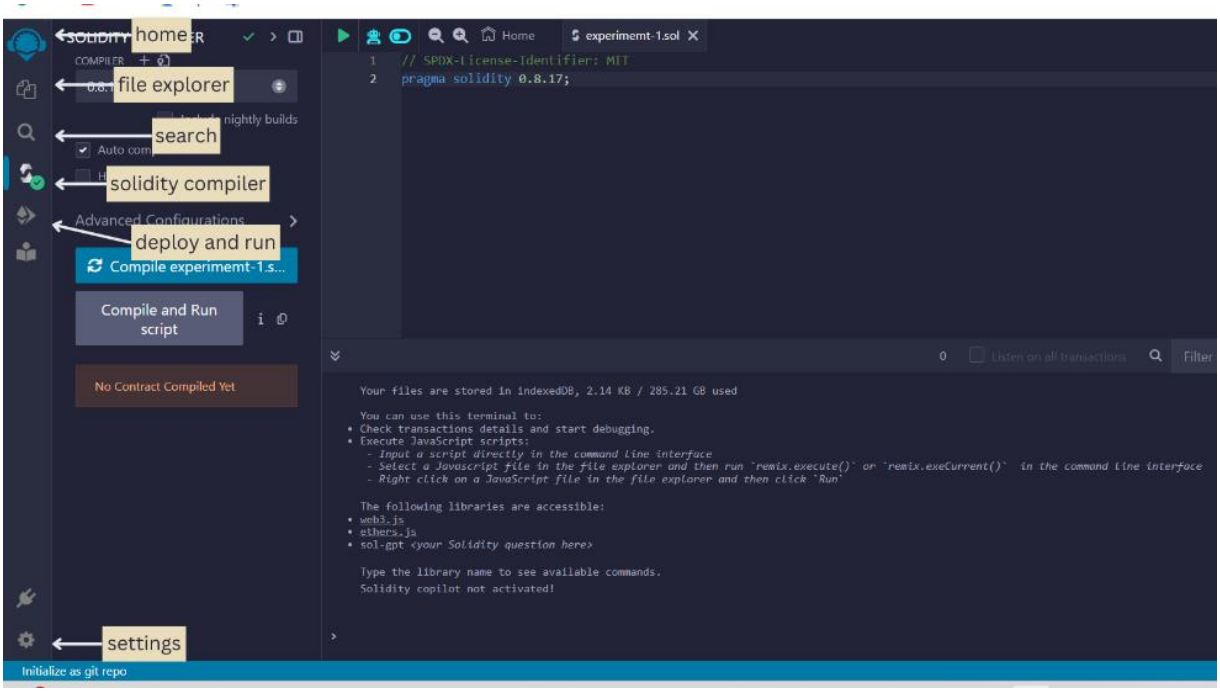

Figure 1.1: File Explore

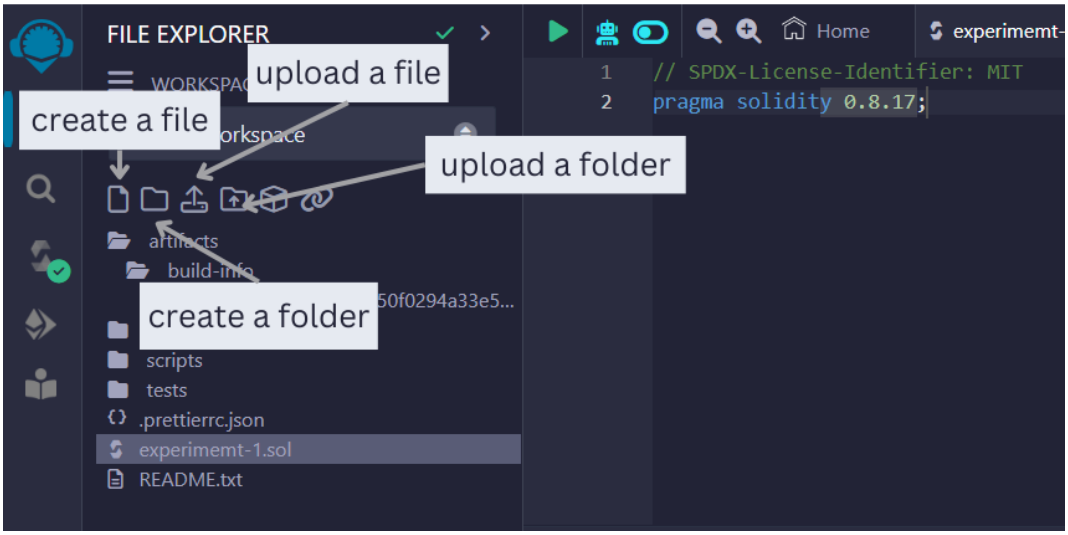

Figure 1.2: How to create a File

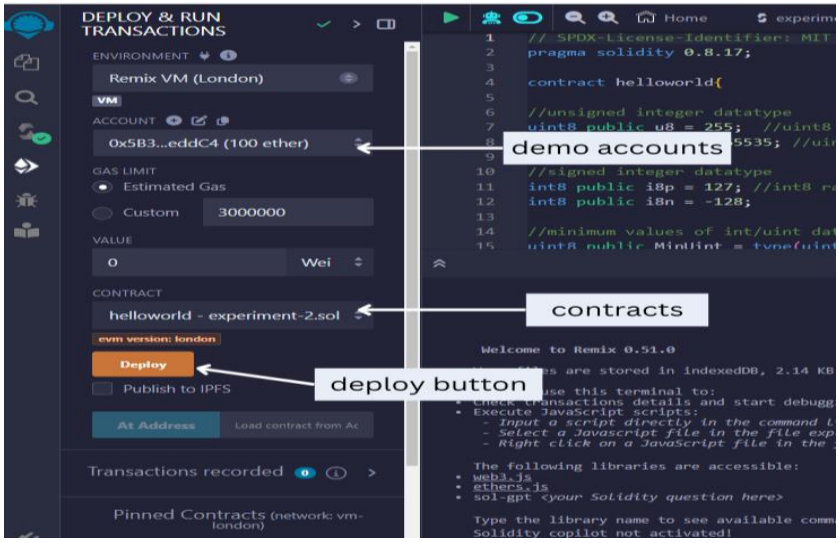

Figure 1.3: Deploy & Run Transactions

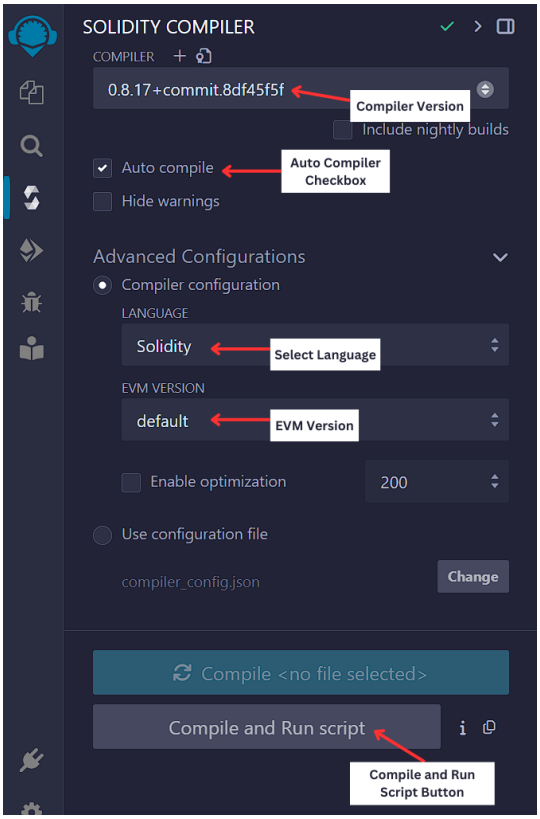

Figure1.4: Solidity Compiler

Conclution: Remix IDE offers a user-friendly interface for developing, deploying, and debugging Ethereum smart contracts. Its intuitive layout and built-in tools make it a popular choice for both beginners and experienced blockchain developers.

Practical:2

AIM

To write, deploy, and execute a basic smart contract that makes use of different data types in Solidity.

Software Used

1. IDE remix (0.51.0)
2. Windows (11)
3. Solidity (0.8.17)

Description

1. Explain in general the aim of this experiment.

·Write a Solidity contract with variables of different data types (uint, int, bool, address, bytes8) and a function to update one.
· Compile and deploy the contract to a blockchain network, then interact with the function to demonstrate data type usage.

2. Write step-wise description of how you conducted experiment.

Open a web browser and navigate to the official Remix IDE website at remix.ethereum.org.

In the editor, write a basic solidity contract, such as:

//SPDX-License_Identifier:MIT
pragma solidity 0.8.17; and write down the code for your given experiment.

For deploy the contract Go to the "Deploy & Run Transactions" tab.Select the environment EnsurePriyanshi2 is selected.Enter constructor parameters if needed.Click "Deploy".

Use the "Debugger" tab to step through execution and identify issues.

Save your code and settings.Export the project using the "Files" tab for backup or further development.

3. Code of this experiment with complete explanation in form of comments.

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.17; // Specifies the Solidity version


contract priyanshi2{
```

```
//unsigned integer datatype
uint8 public priyanshi_u8 = 255;  //uint8 range is 0 to 2^8-1=255
uint16 public priyanshi_u16 = 65535; //uint16 range is 0 to 2^16-1=65535

//signed integer datatype
int8 public priyanshi_i8p = 127; //int8 range is -128 to 127
int8 public priyanshi_i8n = -128;

//minimum values of int/uint datatype
uint8 public priyanshi_MinUint = type(uint8).min;// Minimum value of uint8 (always 0)
int8 public  priyanshi_MinInt = type(int8).min;// Minimum value of int8

//maximum values of int/uint datatype
uint8 public priyanshi_MaxUint = type(uint8).max;// Maximum value of uint8
int8 public  priyanshi_MaxInt = type(int8).max;// Maximum value of int8

//boolean datatype (can be true or false)
bool public  priyanshi_boolvar = true;

//string datatype
string public priyanshi_stringName = "piyu";

//special type of datatype in solidity for store the ethereum address in bits is
//20*8=160 bits and in hexa0decimal it is 160/4=40 hexa-decimal
address public priyanshi_addvar = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;

//bytes datatype
bytes1 priyanshi_bytes1var = 0xff;
```

bytes2 priyanshi_bytes2var = 0x00ff;

//integer array dynamic and fixed size

uint[] priyanshi_intArrayVar = [1,2,3,4,5];

}

4. Snapshot with caption of the of the code and output in the platform used

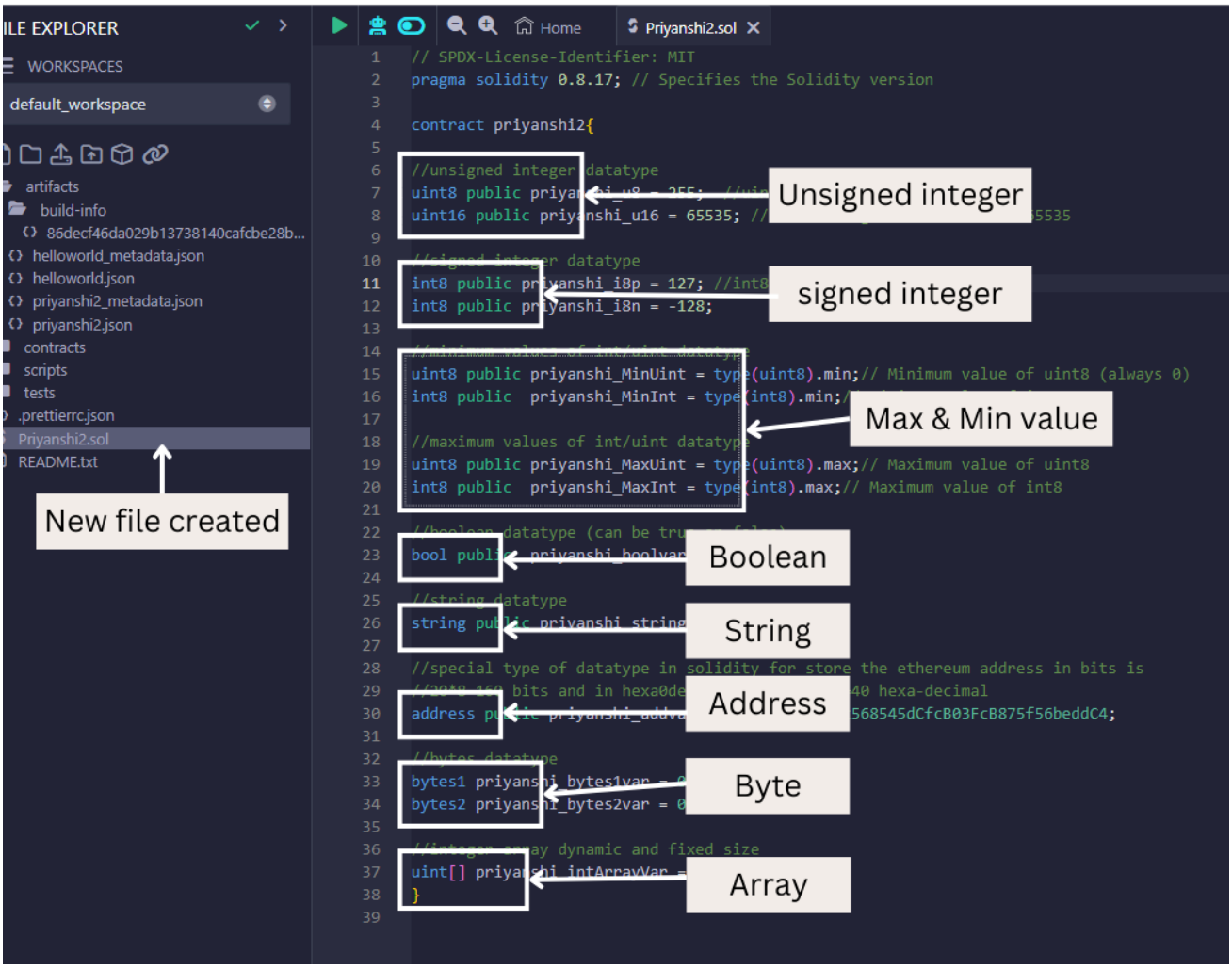

Figure 2.1: Code

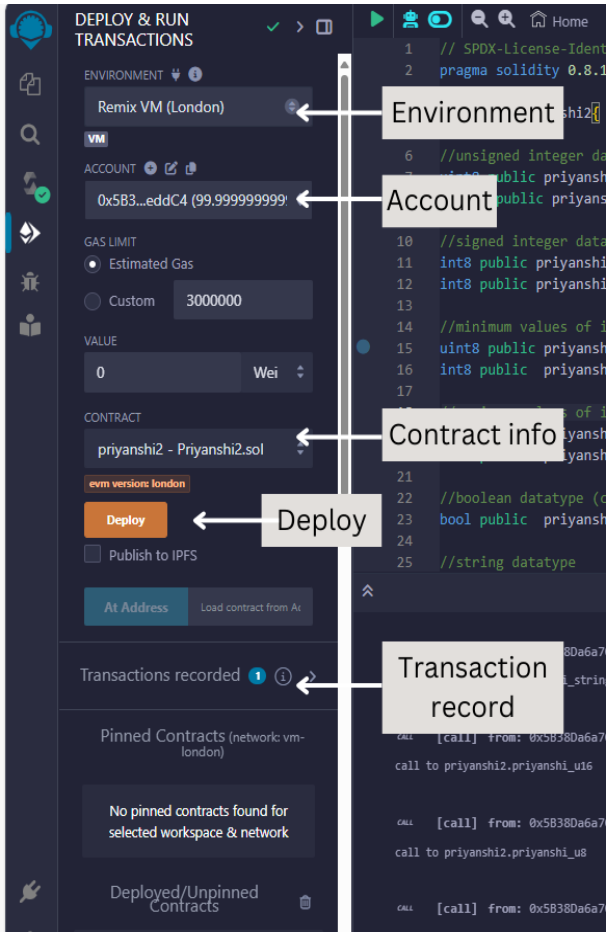

Figure 2.2: Deploy the experiment

Figure 2.3:Variables and their values.

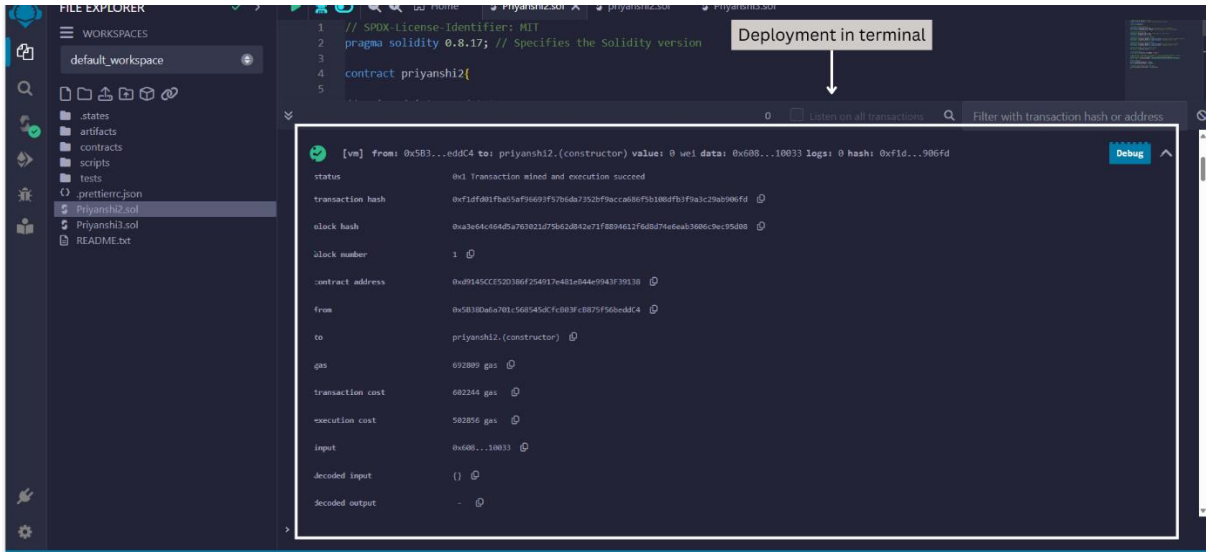

Figure 2.4:Deployment In terminal

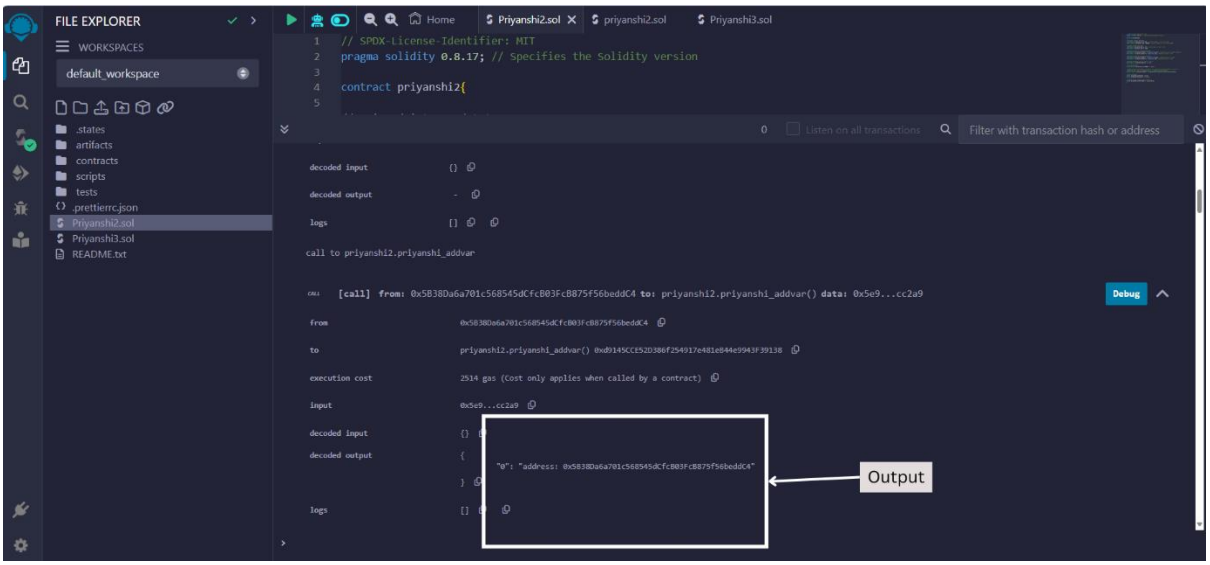

Figure 2.5:Output in terminal

Conclusion:

Writing, Deploying, and Running a Solidity Smart Contract with Various Data Types By creating a Solidity smart contract that utilizes different data types, you can gain practical experience with the language's core functionalities.

Practical:3

AIM

To write, deploy, and execute a smart contract that consists of state variables, local variables, constructor and public/external function in Solidity.

Software Used

1. IDE remix (0.51.0)
2. Windows (11)
3. Solidity (0.8.17)

Description

1. Explain in general the aim of this experiment.

The aim of this experiment is to demonstrate the basic concepts of writing, deploying, and executing a Solidity smart contract that utilizes state variables, a constructor, and public/external functions.

2. Write step-wise description of how you conducted experiment.

Open a web browser and navigate to the official Remix IDE website at remix.ethereum.org.

create a new Solidity file by clicking on the "+" icon in the file explorer section. Name the file Priyanshi3.sol

In the editor, write a basic solidity contract, such as:
//SPDX-License_Identifier:MIT
pragma solidity 0.8.17; and write down the code for your given experiment.

For deploy the contract Go to the "Deploy & Run Transactions" tab.Select the environment EnsurePriyanshi2 is selected.Enter constructor parameters if needed.Click "Deploy".

Use the "Debugger" tab to step through execution and identify issues.

Save your code and settings.Export the project using the "Files" tab for backup or further development.

3. Code of this experiment with complete explanation in form of comments.

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity 0.8.17;// Specifies the version of Solidity to be used
contract Priyanshi3 {
    string public name;// Public variable to store the name
    uint public age;// Public variable to store the age
    bool public student;// Public variable to indicate if the person is a student
    // Constructor function that is executed once when the contract is deployed
    constructor ()
    {
        name = "Priyanshi";// Initialize the name variable with "Priyanshi"
        age = 20;// Initialize the age variable with 20
        student = true;// Initialize the student variable with true
    }
    // External function to set the details of the person
    function setdetails_external (string memory _name, uint _age, bool _student) external {

        name = _name;// Update the name variable with the provided name
        age = _age;// Update the age variable with the provided age
        student = _student;// Update the student variable with the provided boolean value
    }
    // Public function to set the details of the person
    function setdetails_public (string memory _name, uint _age, bool _student) public {

        name = _name;// Update the name variable with the provided name
```

```
        age = _age; // Update the age variable with the provided age
        student = _student;// Update the student variable with the provided boolean value
    }
}
```

4. Snapshot with caption of the of the code and output in the platform used.

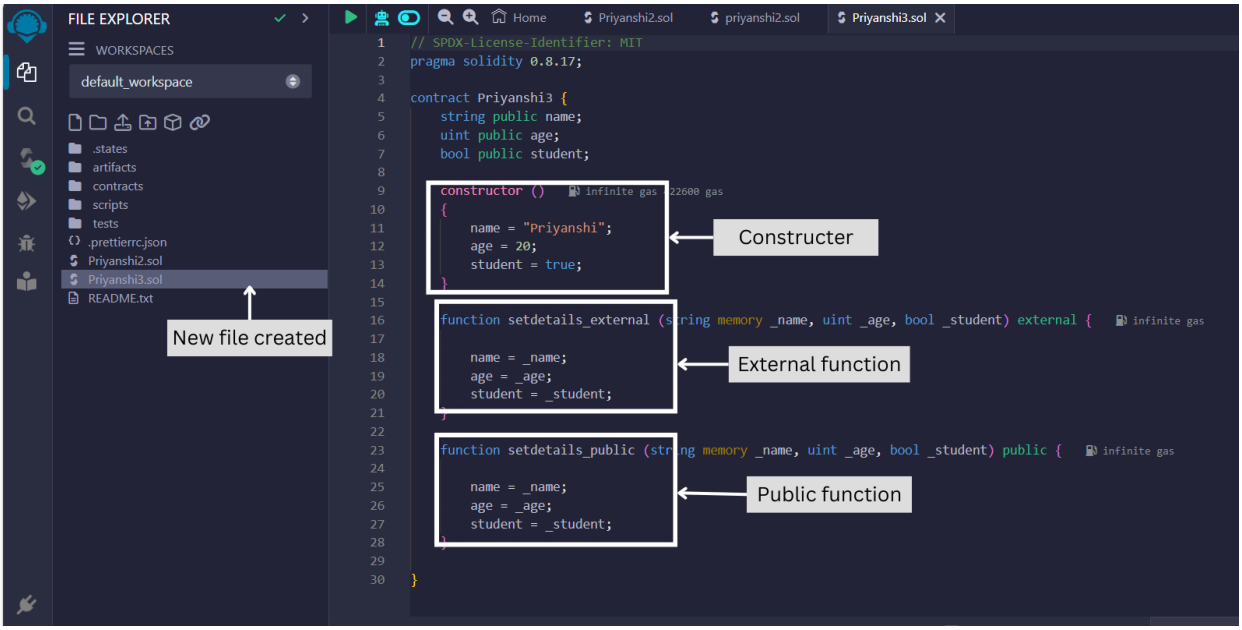

Figure 2.1: Code.

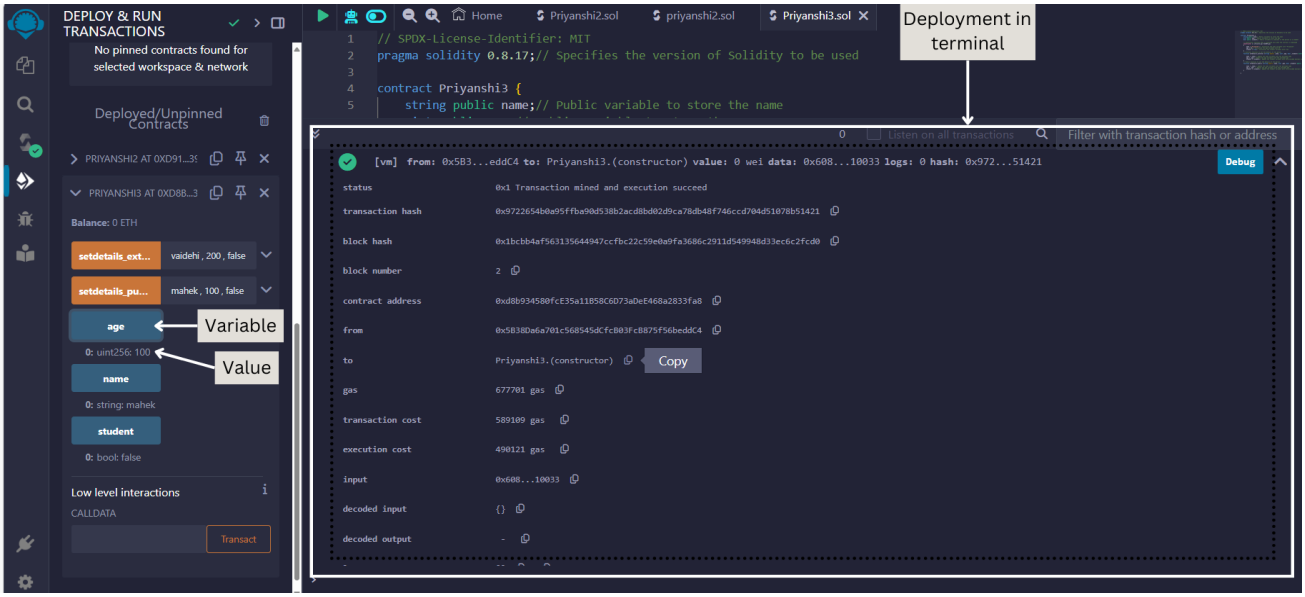

Figure 2.3: Age

Figure 2.4: Name

Figure 2.5: Student

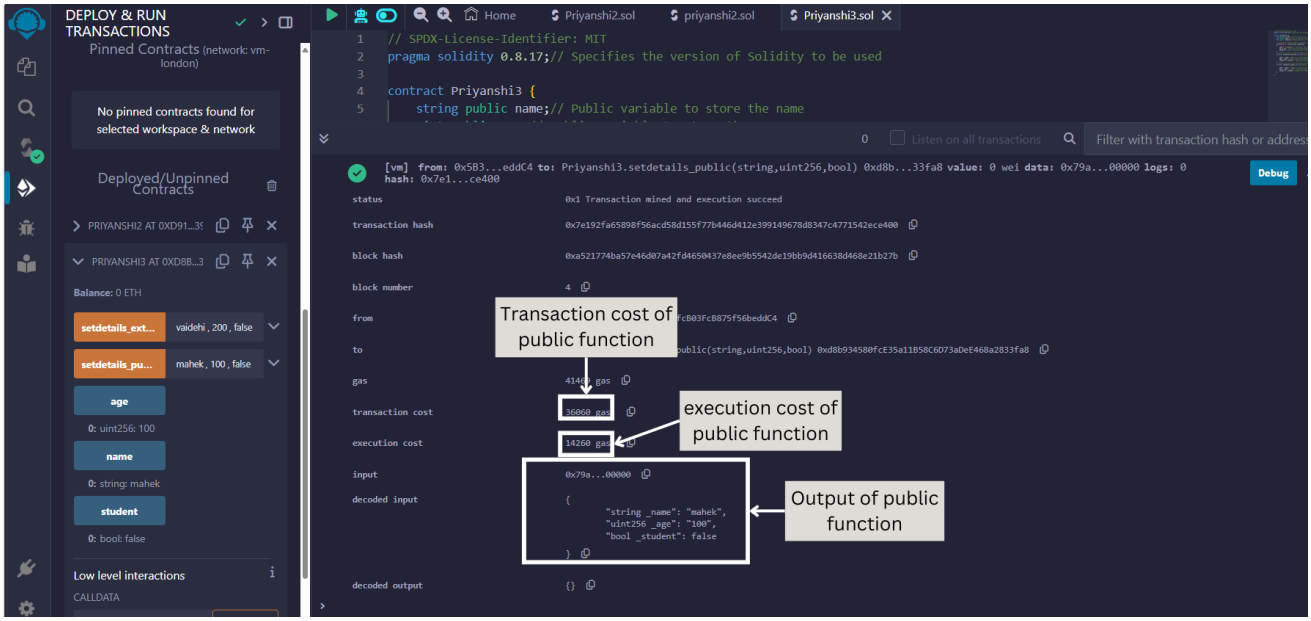

Figure 2.6:Output of a public function

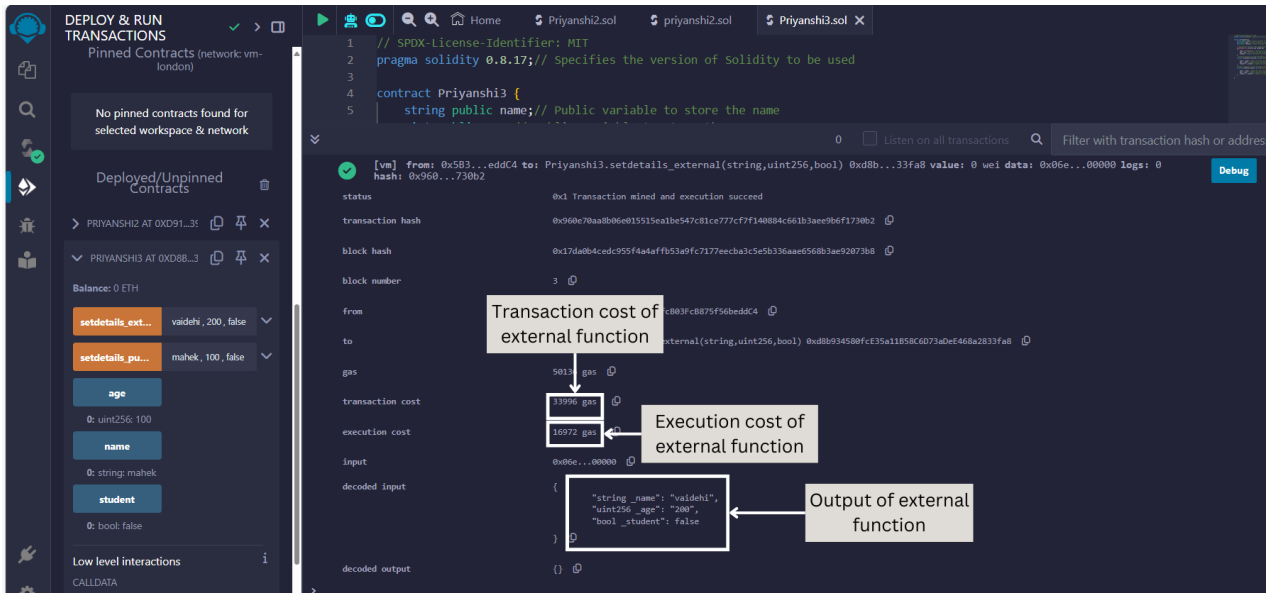

Figure 2.7:Output of a External function

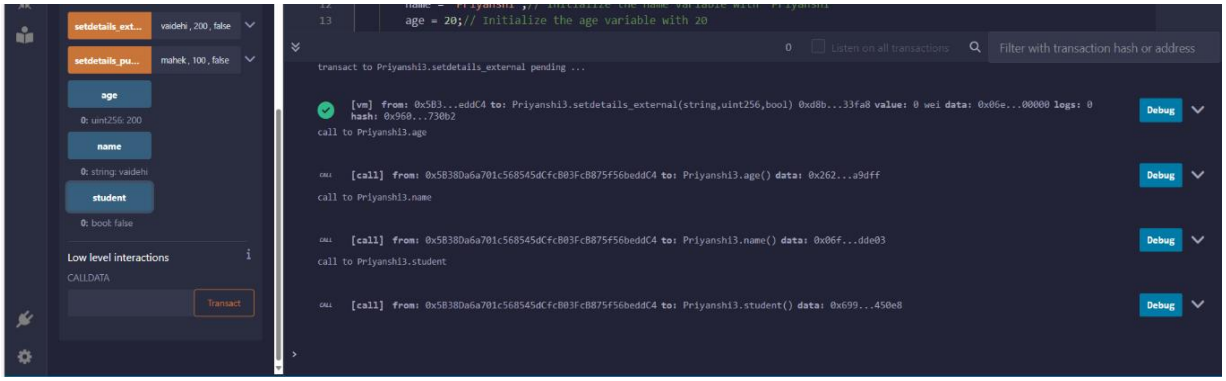

Figure 2.8: External function.

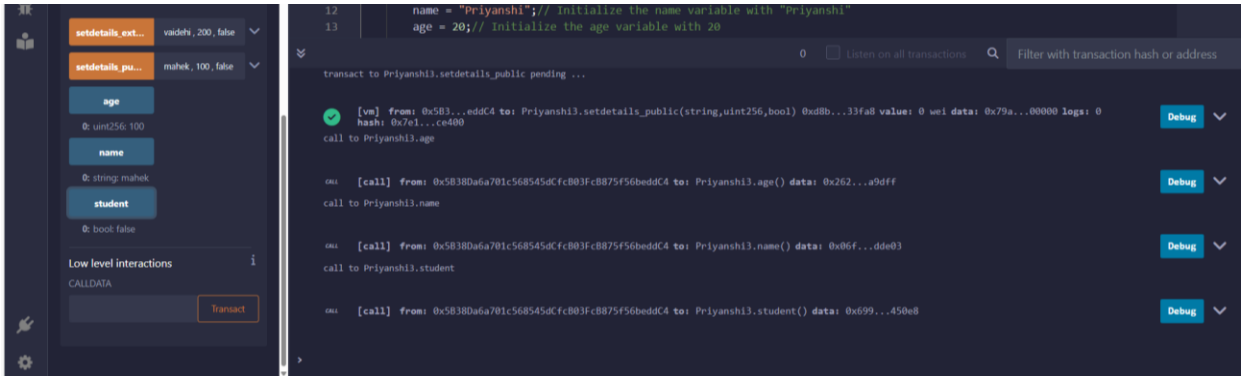

Figure 2.9: Public function.

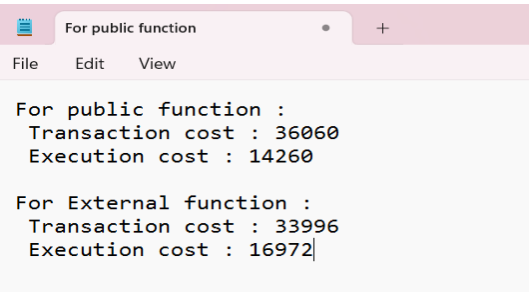

Figure 2.10: Cost comparison between public function and external function.

Conclusion

The experiment with the Priyanshi3 smart contract successfully demonstrated key concepts of Solidity programming, including the initialization of state variables through a constructor, the implementation of public and external functions for modifying these variables, and the accessibility of public variables from outside the contract. Upon deployment, the state variables name, age, and student were correctly initialized and subsequently updated through the defined functions, showcasing the differences in function visibility.

Practical:4

Aim: To write, deploy, and execute a smart contract that performs simple mathematical operations of three operands, and makes use of view functions in Solidity.

Software Used

1. IDE remix (0.51.0)
2. Windows (11)
3. Solidity (0.8.17)

Description

1. Explain in general the aim of this experiment.

The primary goal of this experiment is to introduce you to the fundamentals of smart contract development using Solidity. By creating a simple contract that performs basic mathematical operations.

2. Write step-wise description of how you conducted experiment.

Open a web browser and navigate to the official Remix IDE website at remix.ethereum.org.

create a new Solidity file by clicking on the "+" icon in the file explorer section. Name the file Priyanshi3.sol

In the editor, write a basic solidity contract, such as:
//SPDX-License_Identifier:MIT
pragma solidity 0.8.17; and write down the code for your given experiment.

For deploy the contract Go to the "Deploy & Run Transactions" tab.Select the environment EnsurePriyanshi2 is selected.Enter constructor parameters if needed.Click "Deploy".

Use the "Debugger" tab to step through execution and identify issues.

Save your code and settings.Export the project using the "Files" tab for backup or further development.

3. Code of this experiment with complete explanation in form of comments.

// SPDX-License-Identifier: MIT //Specifies the license under which the code is released.

pragma solidity 0.8.17; // Specifies the version of Solidity to be used

```
contract Priyanshi4 { //New smart contract

 // state variables

 int256 Priyanshi_O1; //Indecates the first variable that stores the first operand.

 int256 Priyanshi_O2;//Indecates the second variable that stores the second operand.

 int256 Priyanshi_O3; //Indecates the third variable that stores the third operand.

 constructor() { //This is a constructor function that initializes the state variables when the contract is deployed.

 Priyanshi_O1 = 12; // Set the first operand to 12

 Priyanshi_O2 = 34; // Set the first operand to 34

 Priyanshi_O3 = 56; // Set the first operand to 56

 }

 // This function returns the sum of operand1, operand2, and operand3.

// Function to add the three operands and return the result.

// This is a view function, meaning it doesn't modify the state of the contract.

 function add_function () view public returns (int)

 {

 int total; //Local variable to store the sum

 total = Priyanshi_O1 + Priyanshi_O2 + Priyanshi_O3; // total is equal to operand1 + operand2 + operand3

 return total; // return the value of total

 }

 // This function returns the result of subtracting operand2 and operand3 from operand1.

 function sub_function() view public returns (int)

 {
```

```
    int total; // local variable

    total = Priyanshi_O1 - Priyanshi_O2 - Priyanshi_O3; // total is equal to operand1 – operand2 – operand3

    return total; // return the value of total

    }

    // This function returns the product of operand1, operand2, and operand3.

    function mul_function () view public returns (int)

    {

    int total; // local variable

    total = Priyanshi_O1 * Priyanshi_O2 * Priyanshi_O3; // total is equal to operand1 * operand2 * operand3

    return total; // return the value of total

    }

    // This function returns the result of dividing operand1 by operand2, and then dividing the result by operand3.

    function div_function () view public returns (int)

    {

    int total = 0; // local variable

    total = (Priyanshi_O1 / Priyanshi_O2) / Priyanshi_O3; // total is equal to (operand1 / operand2) / operand3

    return total; // return the value of total

    }

    // This function sets the values of operand1, operand2, and operand3.

    function set_operands (int x, int y, int z) public

    {
```

```
Priyanshi_O1 = x; // set operand1 equal to a

Priyanshi_O2 = y; // set operand2 equal to b

Priyanshi_O3= z; // set operand3 equal to c

}

// This function returns the values of operand1, operand2, and operand3.

function get_operands () view public returns (int, int, int)

{

return (Priyanshi_O1, Priyanshi_O2, Priyanshi_O3); // return the values of operand1, operand2, and operand3

}

}
```

4. Snapshot with caption of the of the code and output in the platform used.

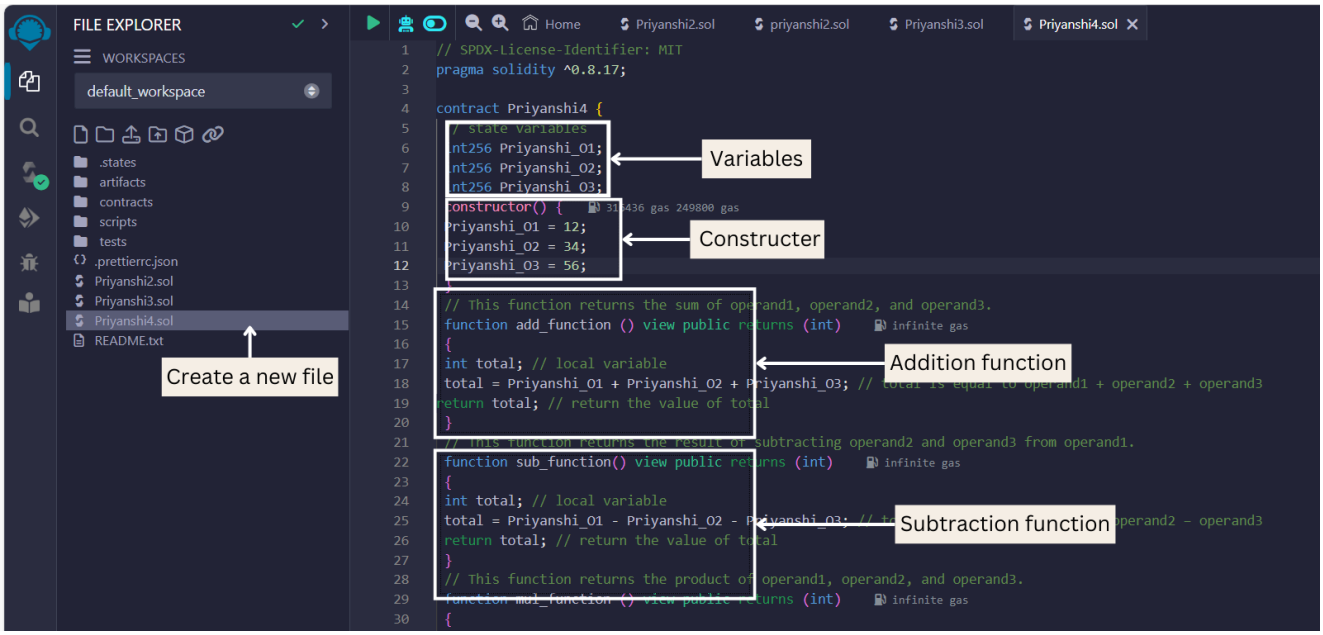

Figure 4.1 : Create A file

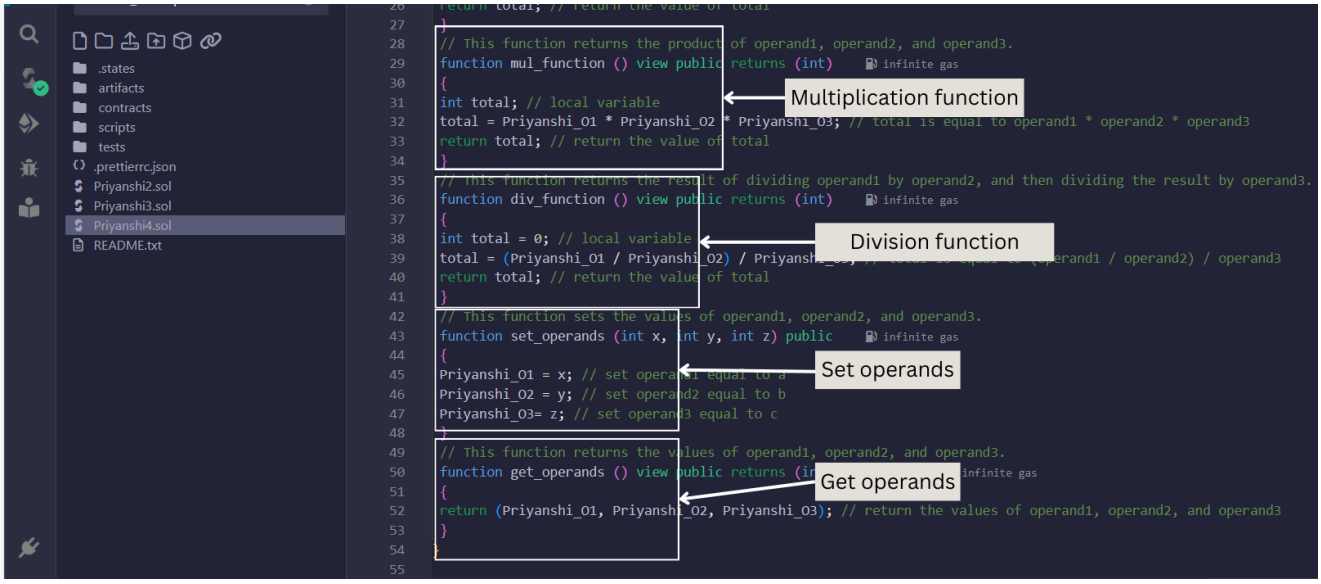

Figure 4.2 : Define functions

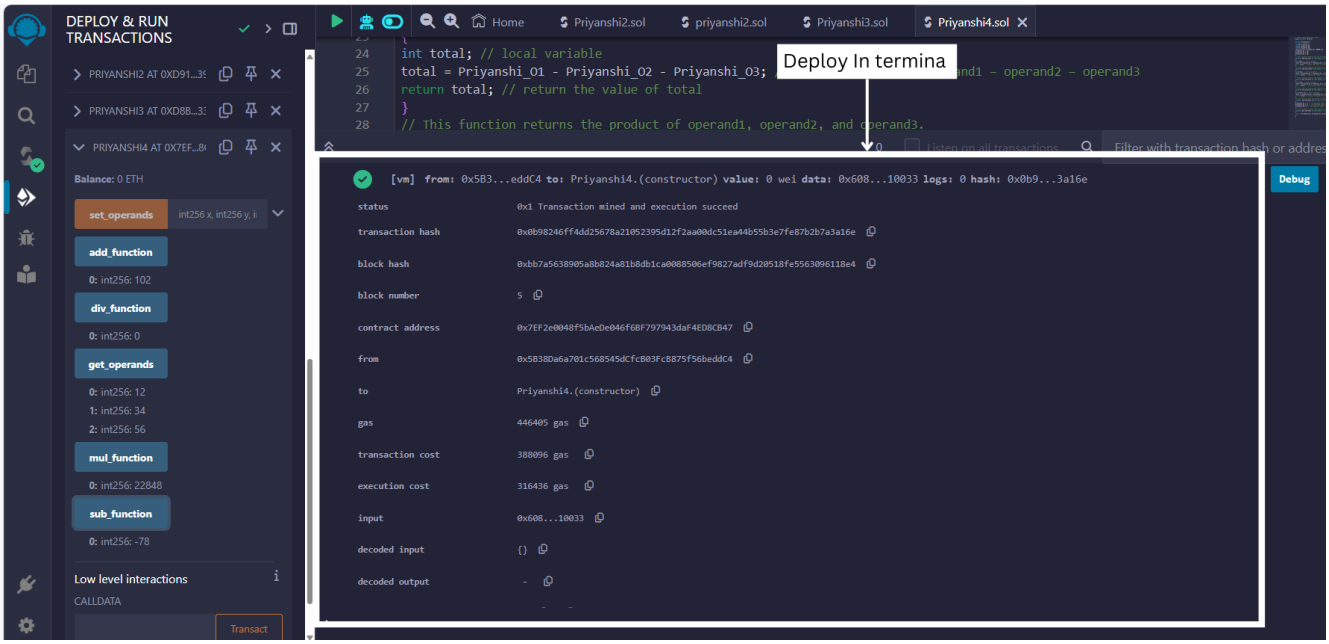

Figure 4.3 :Deploy in terminal

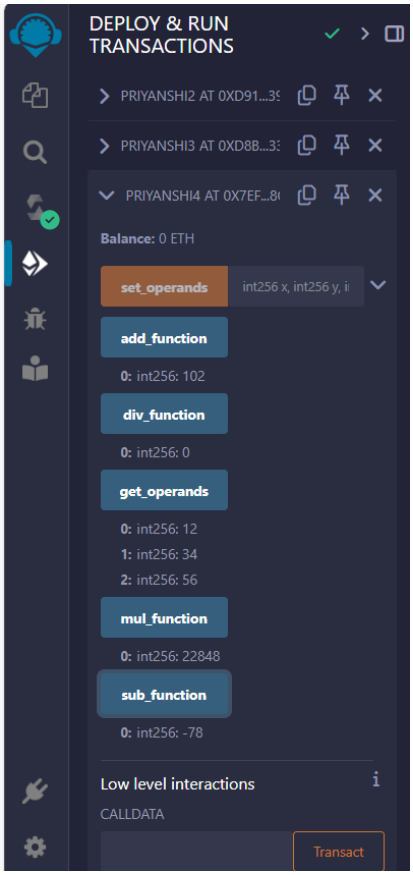

Figure 4.4 :Initialized values

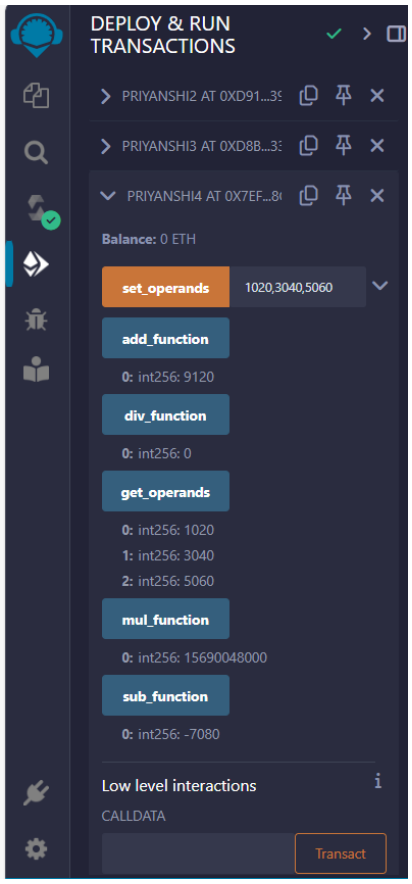

Figure 4.5 :Set values

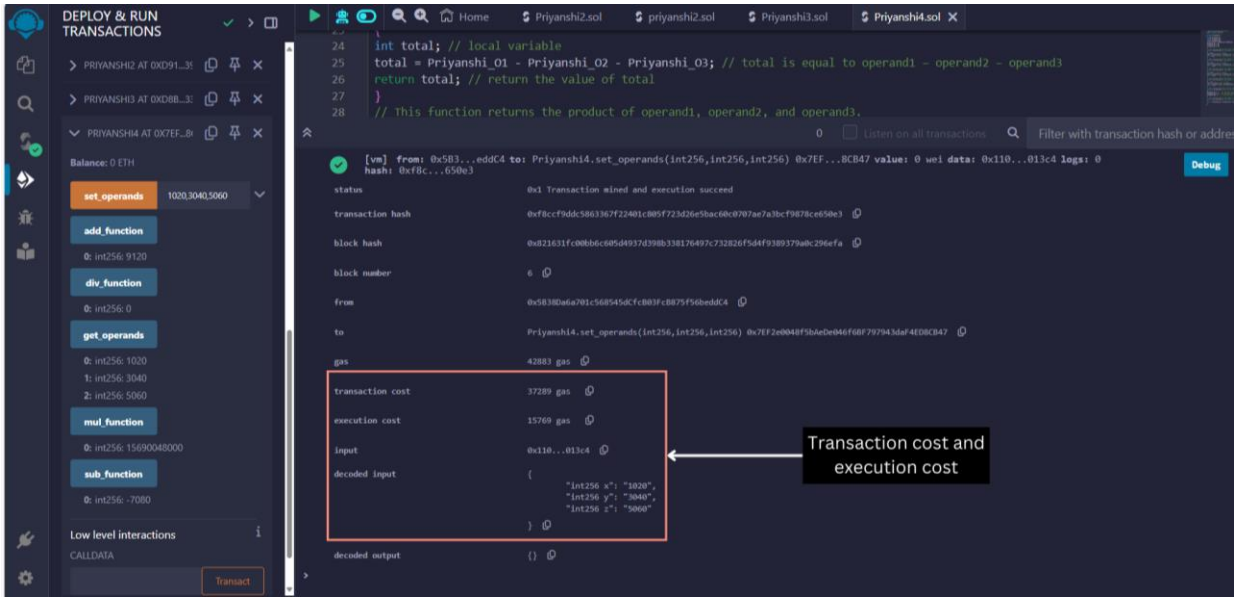

Figure 4.6 :Transaction cost :37289 , Execution cost :15769

Conclusion:

In this experiment, we successfully wrote, deployed, and executed a smart contract in Solidity using Remix IDE to perform basic mathematical operations (addition, subtraction, multiplication, and division) on three operands.