**Exp 1**

INPUT :

import hashlib

class MerkleTree:

def \_\_init\_\_(self, data\_blocks):

self.leaves = [(data, self.hash\_data(data)) for data in data\_blocks]

self.intermediate\_hashes = [] # To store intermediate hashes

self.root = self.build\_tree([leaf[1] for leaf in self.leaves])

def hash\_data(self, data):

# Hashes data using SHA-256 and returns the hexadecimal digest

return hashlib.sha256(data.encode(&#39;utf-8&#39;)).hexdigest()

def build\_tree(self, leaves):

if len(leaves) == 1:

return leaves[0]

# If odd number of leaves, duplicate the last leaf

if len(leaves) % 2 != 0:

leaves.append(leaves[-1])

# Compute the parent level

parent\_level = []

for i in range(0, len(leaves), 2):

combined\_hash = leaves[i] + leaves[i + 1]

parent\_hash = self.hash\_data(combined\_hash)

parent\_level.append(parent\_hash)

self.intermediate\_hashes.append((leaves[i], leaves[i + 1], parent\_hash)) # Store intermediate

hashes

# Recursively build the tree until we get the root hash

return self.build\_tree(parent\_level)

def get\_root(self):

return self.root

def get\_leaves(self):

return self.leaves

def get\_intermediate\_hashes(self):

return self.intermediate\_hashes

if \_\_name\_\_ == &quot;\_\_main\_\_&quot;:

data\_blocks = []

print(&quot;Enter data blocks for the Merkle Tree (type &#39;done&#39; to finish):&quot;)

while True:

data = input(&quot;Enter data block: &quot;)

if data.lower() == &#39;done&#39;:

break

data\_blocks.append(data)

if data\_blocks:

merkle\_tree = MerkleTree(data\_blocks)

print(&quot;Hashes of each block:&quot;)

for data, hash\_value in merkle\_tree.get\_leaves():

print(f&quot;Data: {data}, Hash: {hash\_value}&quot;)

print(&quot;\nIntermediate hashes:&quot;)

for leaf1, leaf2, parent\_hash in merkle\_tree.get\_intermediate\_hashes():

print(f&quot;Combined: {leaf1} + {leaf2} -&gt; Hash: {parent\_hash}&quot;)

print(f&quot;\nMerkle Root: {merkle\_tree.get\_root()}&quot;)

else:

print(&quot;No data blocks entered.&quot;)

OUTPUT :

C:\Users\anish\PycharmProjects\pythonProject1\venv\Scripts\python.exe

C:\Users\anish\PycharmProjects\pythonProject1\bc\_merkle.py

Enter data blocks for the Merkle Tree (type &#39;done&#39; to finish):

Enter data block: anisha

Enter data block: dimple

Enter data block: harsh

Enter data block: dhruvi

Enter data block: done

Hashes of each block:

Data: anisha, Hash: 40fc3451e882282df7cc37fdc6d3fd9b4a1d3673c1a05c0350ed4ce31a3ce49b

Data: dimple, Hash: 4cdef747af0872ef28f794224daf75944d0e31c9cf1f1885d2d18b45f41f51bc

Data: harsh, Hash: bd878fa2293de72e9c871463e1c2ef2964acb0ea282540c9d2f52fe32c6d18d1

Data: dhruvi, Hash: d8d6881de3a6bfab0bfd8496c5ae90e7f6c3a8e66f4b36ff9f9137037991f3fc

Intermediate hashes:

Combined: 40fc3451e882282df7cc37fdc6d3fd9b4a1d3673c1a05c0350ed4ce31a3ce49b +

4cdef747af0872ef28f794224daf75944d0e31c9cf1f1885d2d18b45f41f51bc -&gt; Hash:

1654fa61d0d7a11add50bbc3d15be9f4b9d098e181b28138a8afeb24ed32314f

Combined: bd878fa2293de72e9c871463e1c2ef2964acb0ea282540c9d2f52fe32c6d18d1 +

d8d6881de3a6bfab0bfd8496c5ae90e7f6c3a8e66f4b36ff9f9137037991f3fc -&gt; Hash:

13f33e5718920f76cf1e4acdbbbe10fad666d13c7c8fa5989c3ff82f60cd3283

Combined: 1654fa61d0d7a11add50bbc3d15be9f4b9d098e181b28138a8afeb24ed32314f +

13f33e5718920f76cf1e4acdbbbe10fad666d13c7c8fa5989c3ff82f60cd3283 -&gt; Hash:

ca851364fdeb77f51580f608252284ecebfebd8dd9cc8c845ee9dea8839ce367

Merkle Root: ca851364fdeb77f51580f608252284ecebfebd8dd9cc8c845ee9dea8839ce367

Process finished with exit code 0

**Exp 2**

Print your name   
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Greeting {

// State variable to store the name

string public name;

// Constructor to initialize the contract with a name

constructor(string memory \_name) {

name = \_name;

}

// Function to set a new name

function setName(string memory \_name) public {

name = \_name;

}

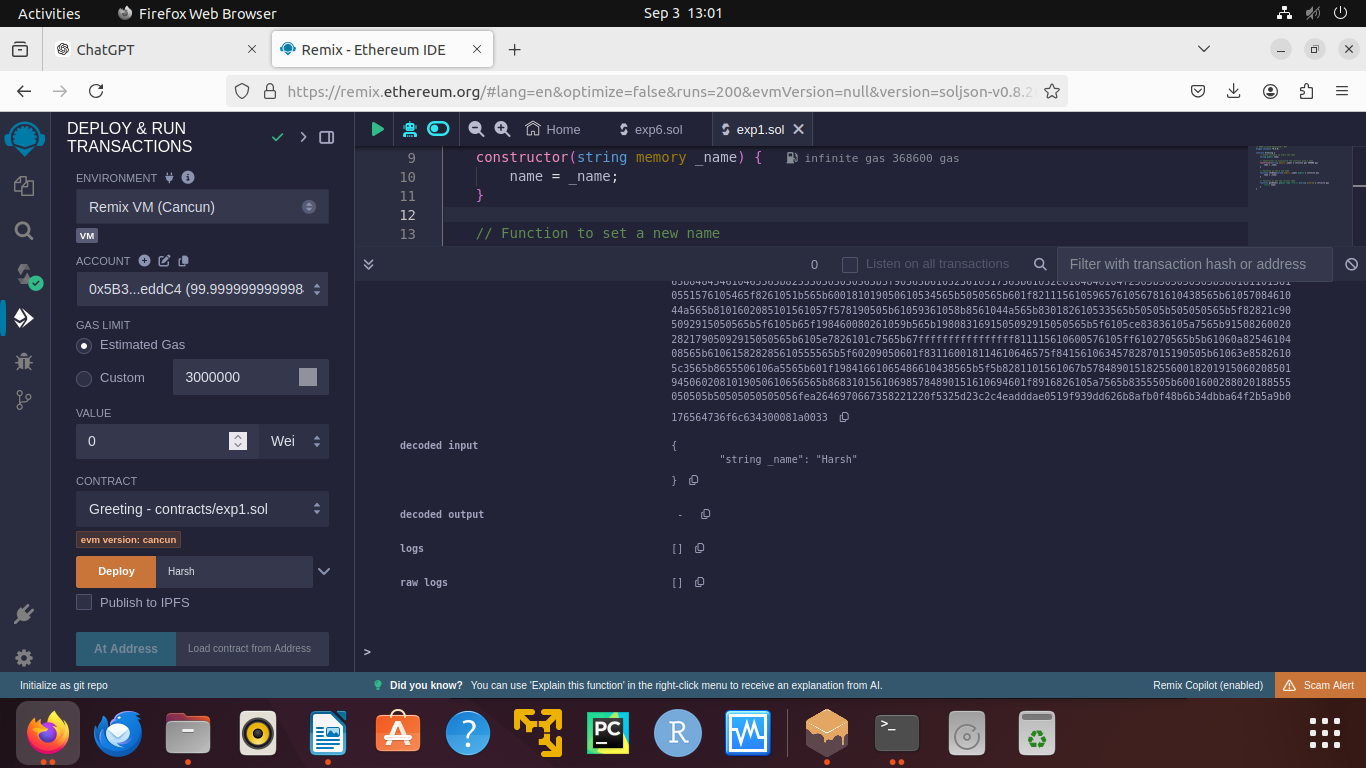
// Function to get the current name

function getName() public view returns (string memory) {

return name;

}

}



Palindrome  
  
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract PalindromeChecker {

// Function to check if a string is a palindrome

function isPalindrome(string memory input) public pure returns (bool) {

bytes memory strBytes = bytes(input);

uint256 length = strBytes.length;

// Iterate from both ends of the string towards the center

for (uint256 i = 0; i < length / 2; i++) {

if (strBytes[i] != strBytes[length - 1 - i]) {

return false; // Not a palindrome if characters do not match

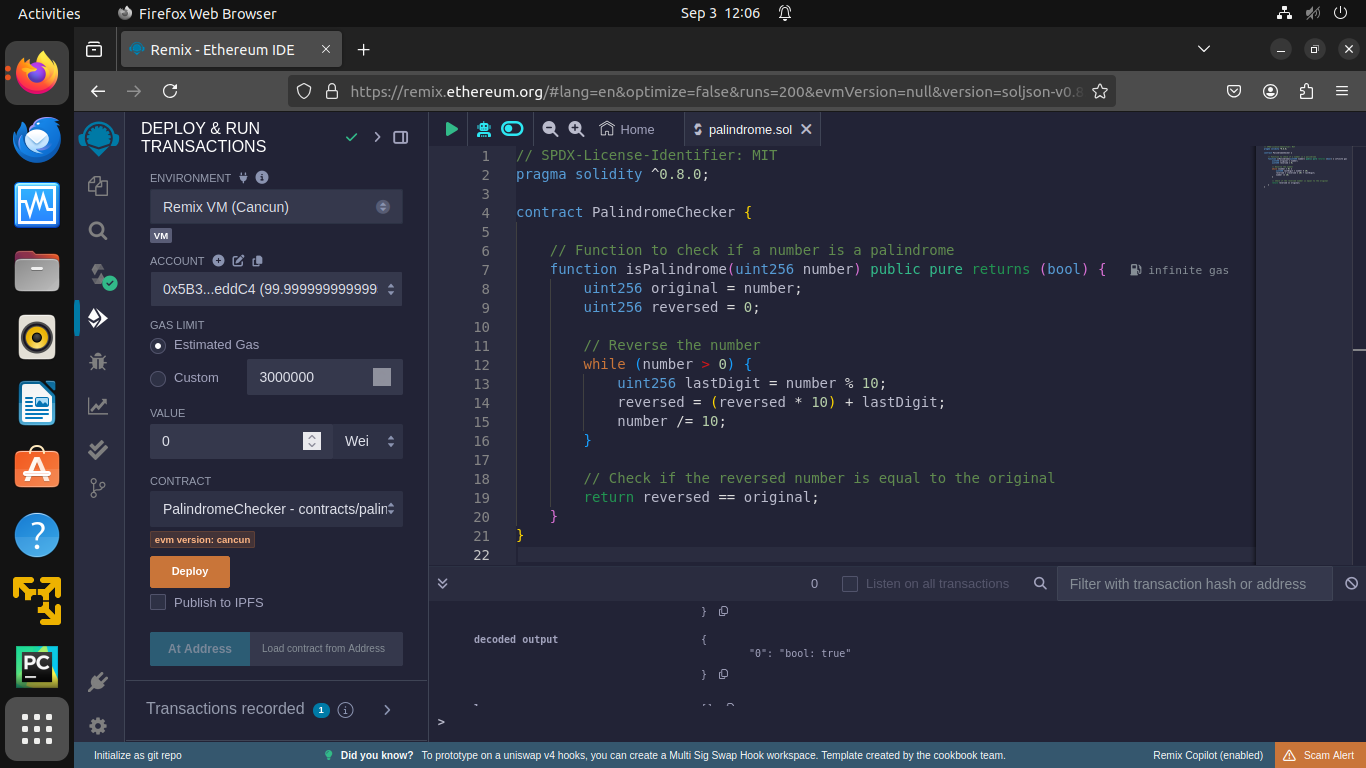
}

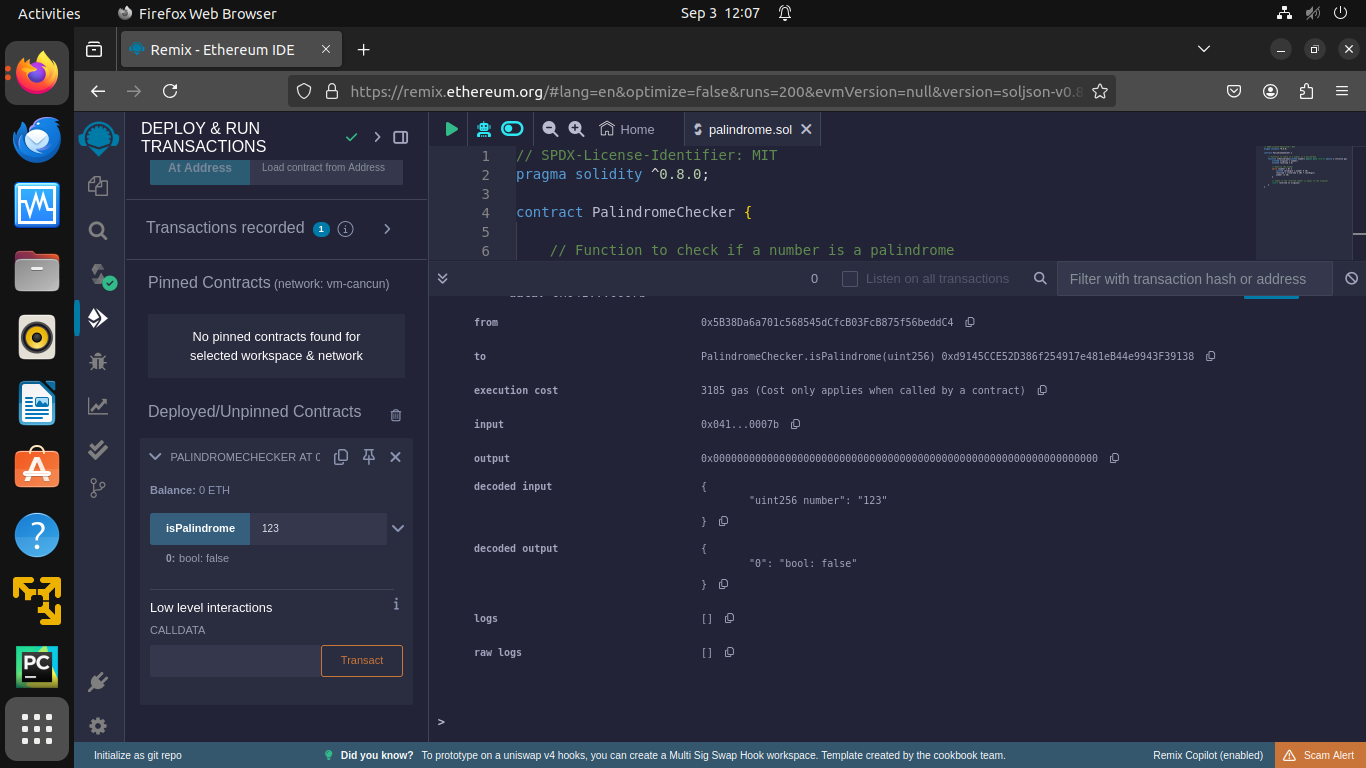
}

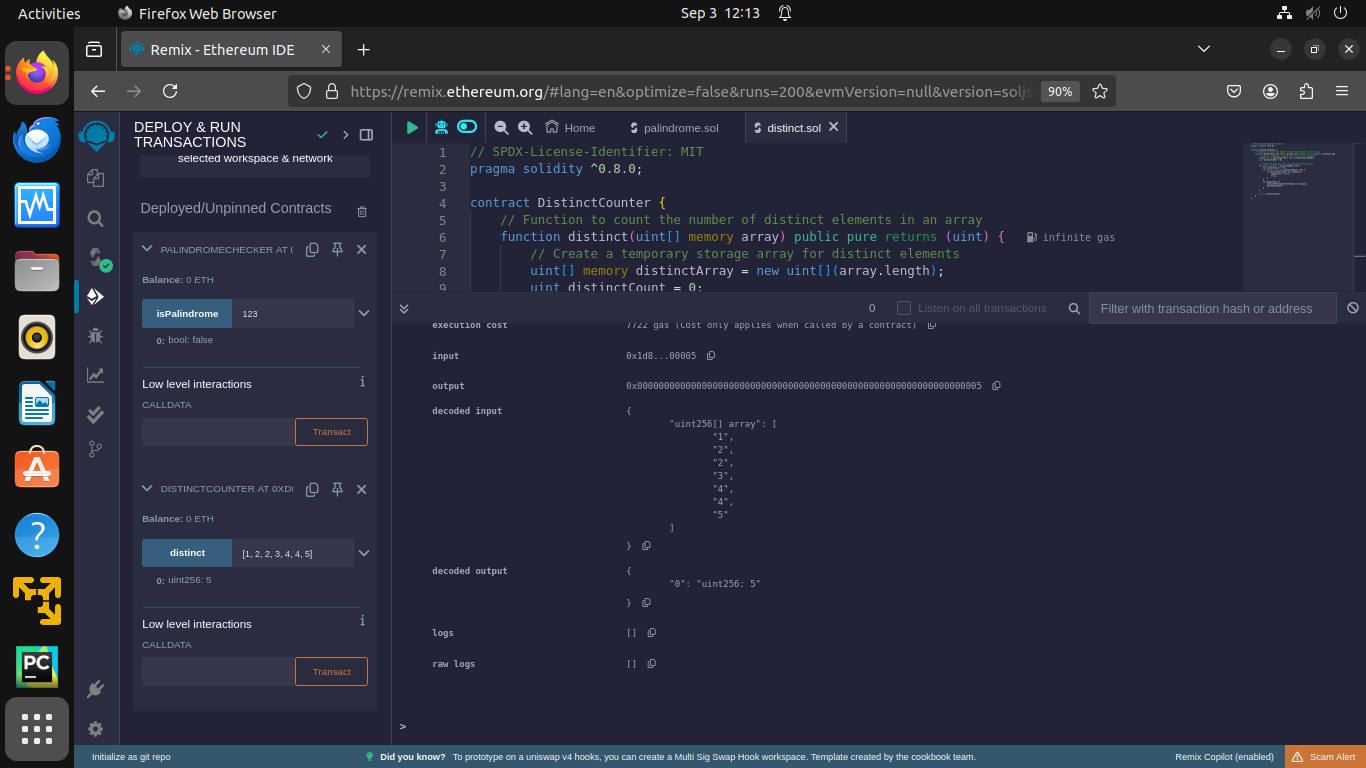
return true; // String is a palindrome

}

}







Concat  
  
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract StringConcatenator {

// Function to concatenate two strings

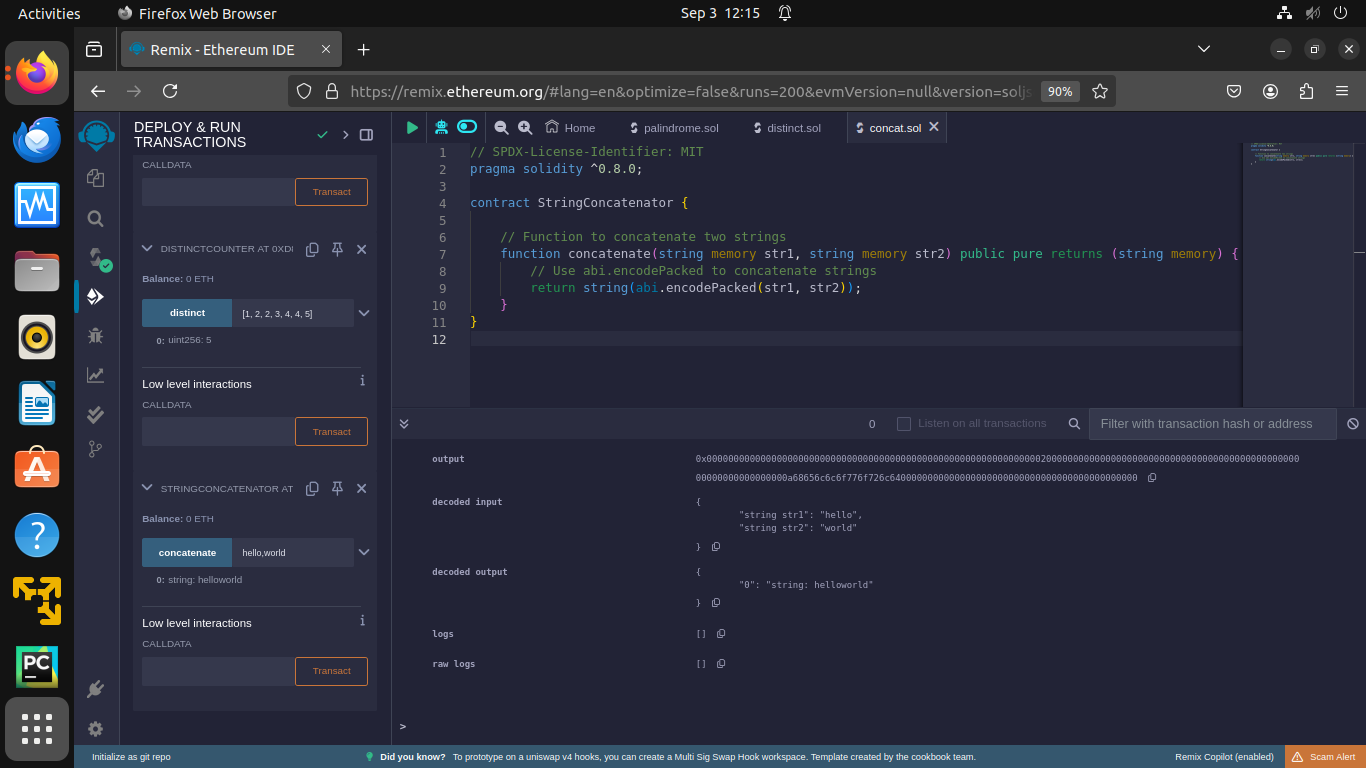
function concatenate(string memory str1, string memory str2) public pure returns (string memory) {

// Use abi.encodePacked to concatenate the strings

return string(abi.encodePacked(str1, str2));

}

}



Exp 3  
  
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract TransactionHandler {

// Event to log transactions

event Sent(address indexed from, address indexed to, uint256 amount);

// Function to receive Ether. This function is called when Ether is sent to the contract.

receive() external payable {}

// Function to withdraw Ether from the contract

function withdraw(uint256 amount, address payable recipient) public {

require(amount <= address(this).balance, "Insufficient balance in the contract");

require(amount > 0, "Amount must be greater than zero");

// Transfer the specified amount of Ether to the recipient

(bool success, ) = recipient.call{value: amount}("");

require(success, "Failed to send Ether");

// Emit an event to log the transaction

emit Sent(address(this), recipient, amount);

}

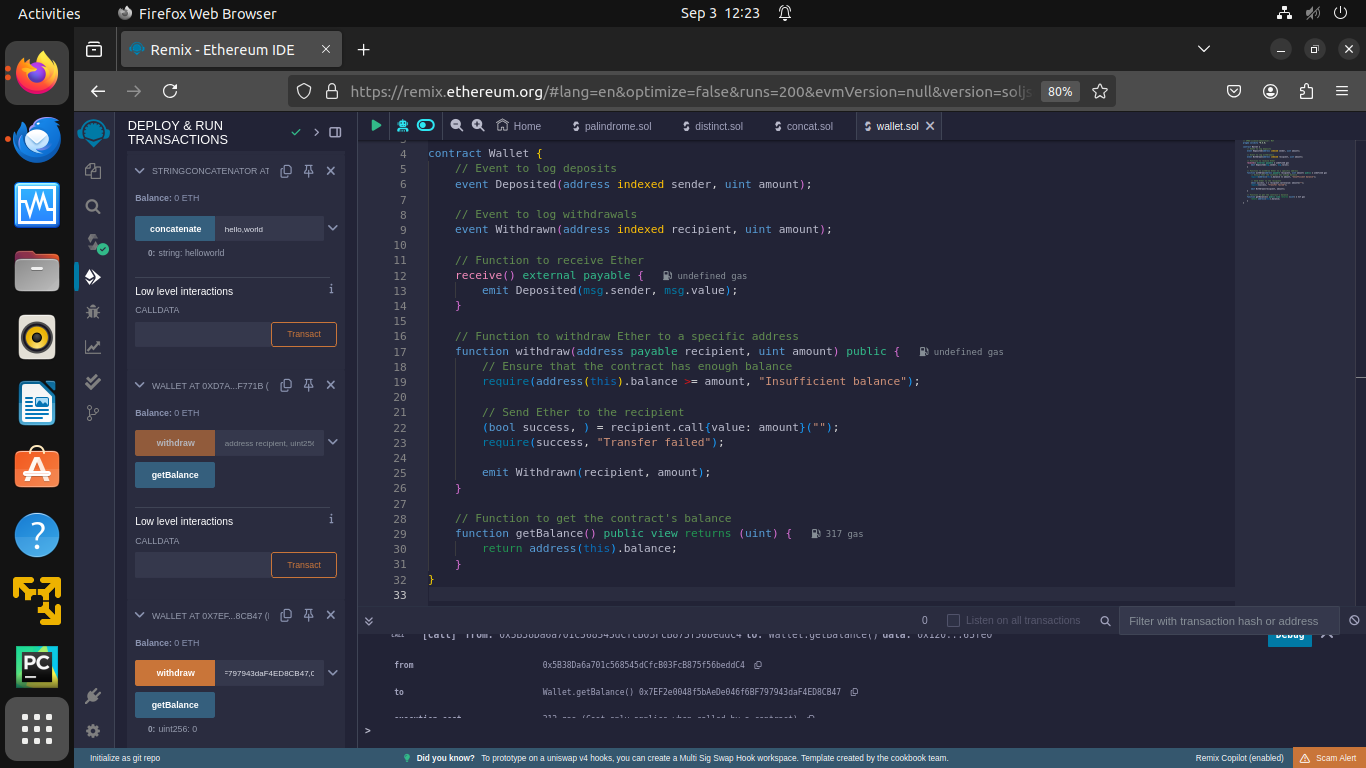
// Function to check the contract's balance

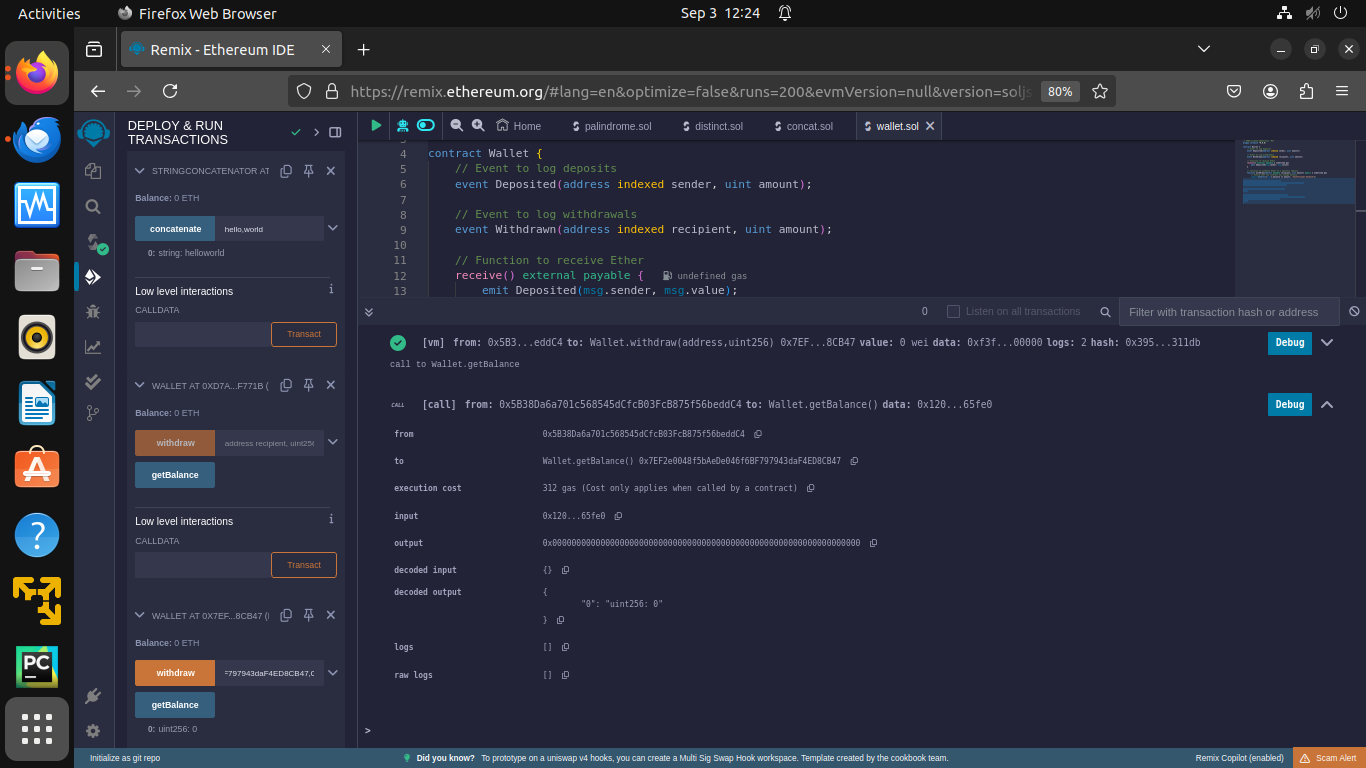
function getBalance() public view returns (uint256) {

return address(this).balance;

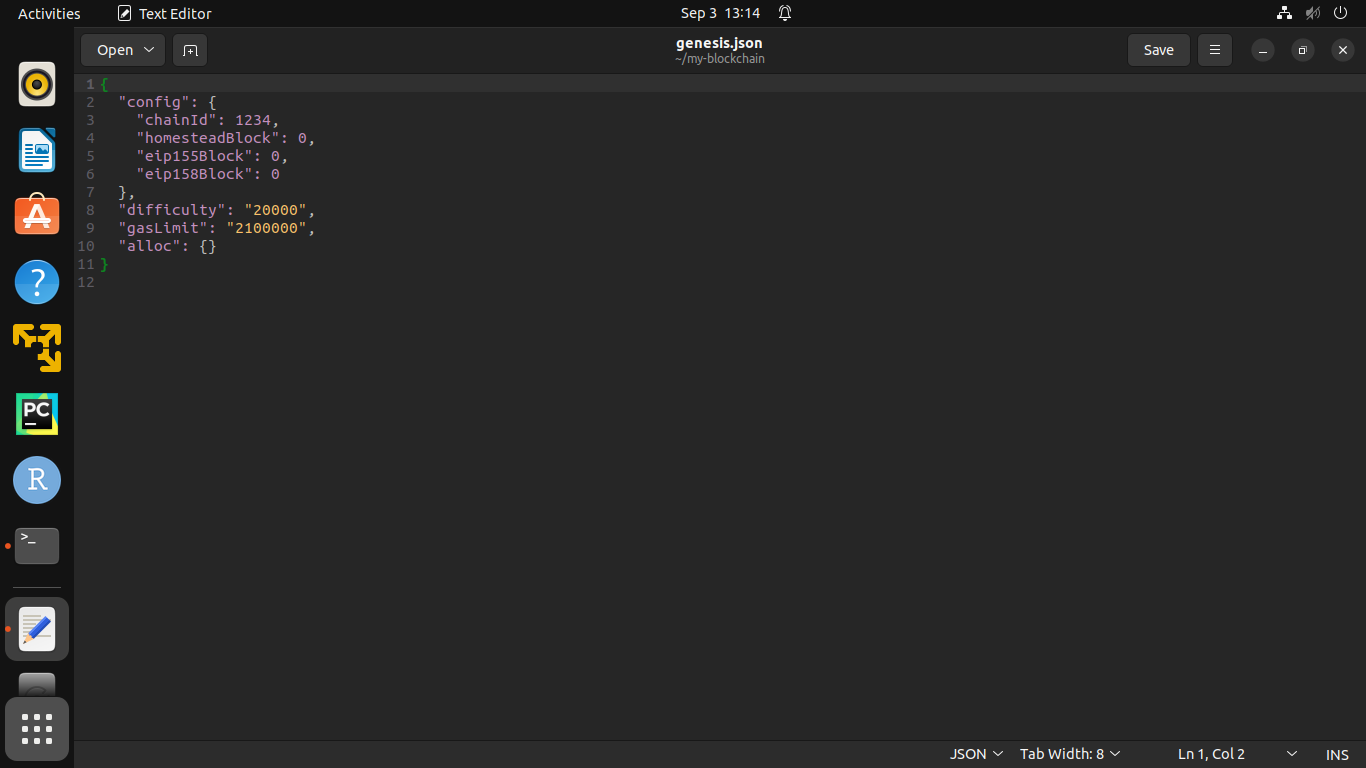
}

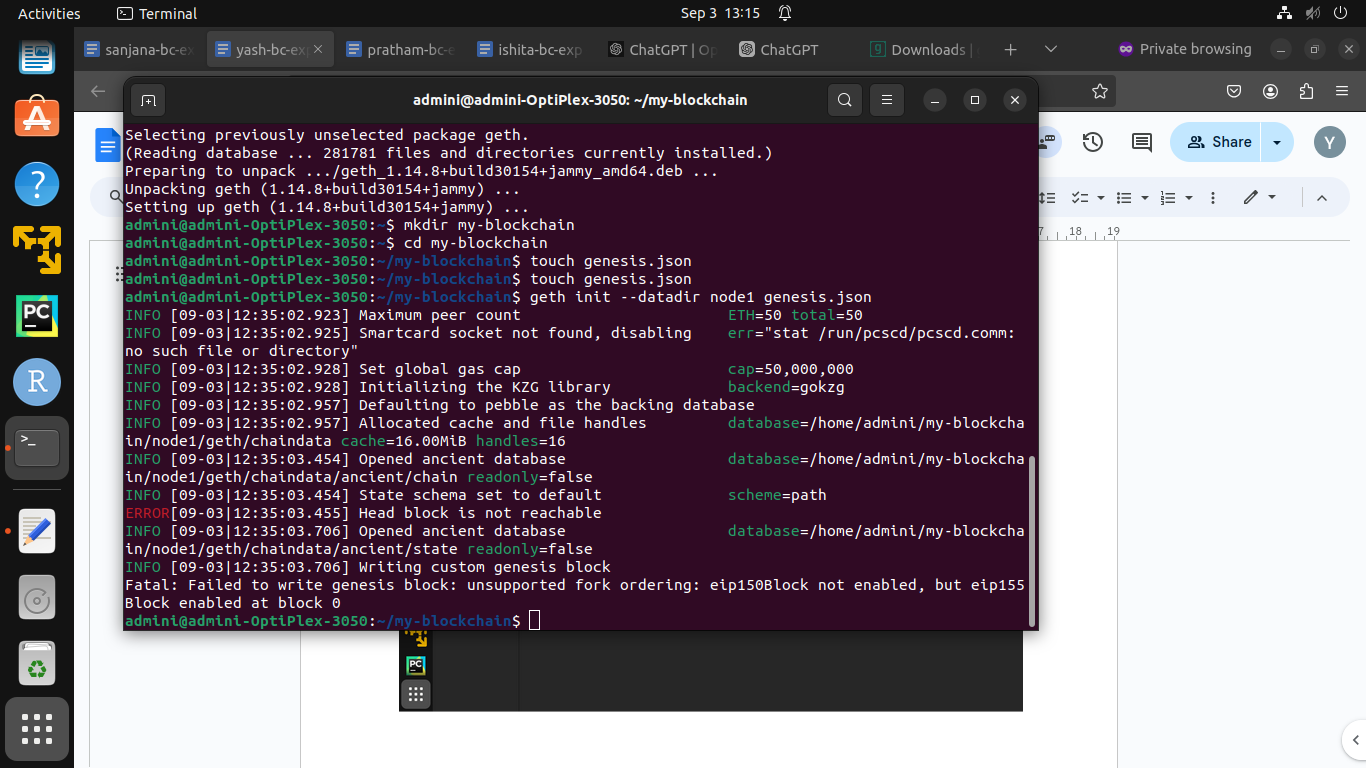
}

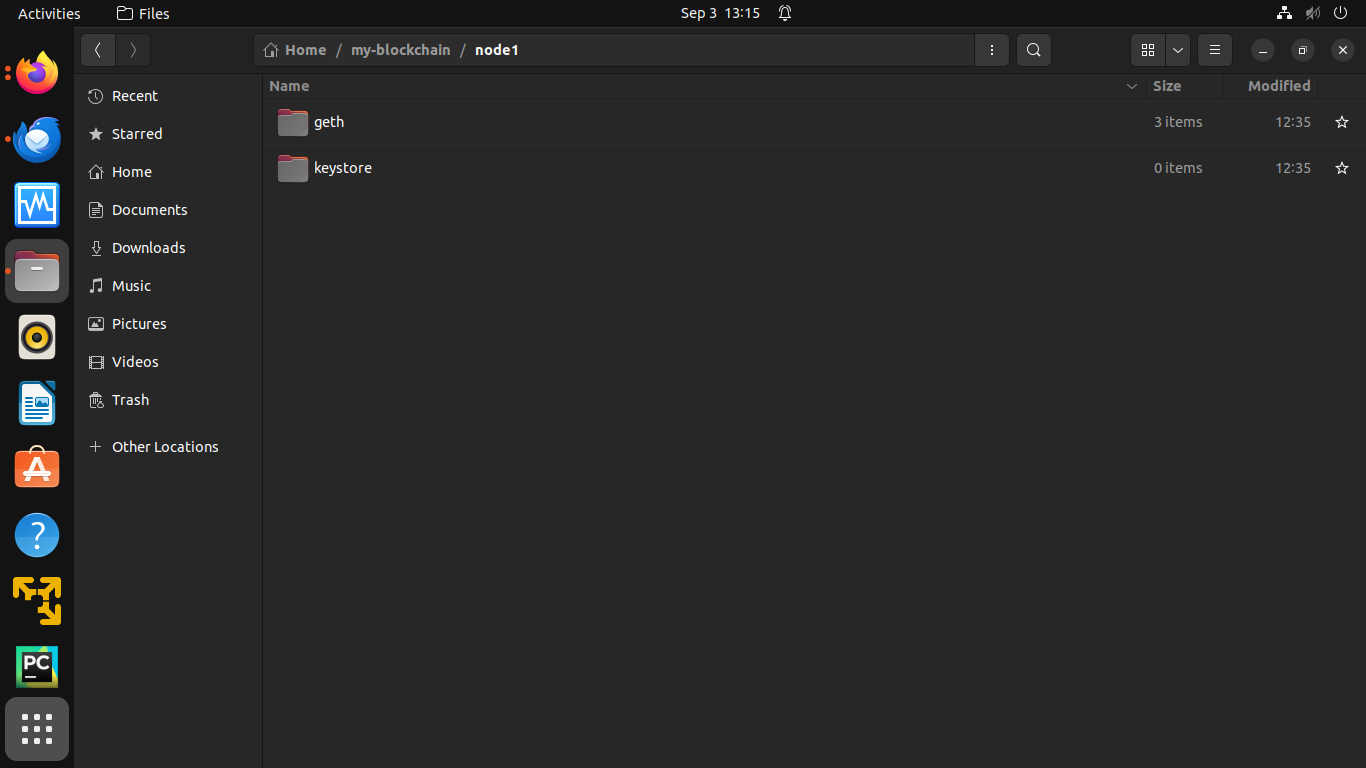




Exp 5







Exp 6

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleCalculator {

// State variables to hold results

int public result;

// Function to perform addition

function add(int a, int b) public {

result = a + b;

}

// Function to perform subtraction

function subtract(int a, int b) public {

result = a - b;

}

// Function to perform multiplication

function multiply(int a, int b) public {

result = a \* b;

}

// Function to perform division

function divide(int a, int b) public {

require(b != 0, "Division by zero is not allowed");

result = a / b;

}

}

