Jayawant Shikshan Prasarak Mandal

CS4102

Blockchain Technology

Laboratory Manual

Computer Engineering

Author : - Prof. S.H.Deshmukh

© JSPM Group of Institutes, Pune. All Rights Reserved. All the information in this Course Manual is confidential. Participants shall refrain from copying, distributing, misusing or disclosing the content to any third parties any circumstances whatsoever.

Table of Contents

Sr . No.	Topic	Page. No.
A.	Vision, Mission	3
B.	PEOs and POs	4-5
C.	PSOs	6
D.	Course Outcomes and its mapping with POs and PSOs	7
E.	Lab Plan	
F. List of Experiments		
1.	Create your own wallet using Coinbase app and also Metamask for sending cryptocurrency	
2.	Installation of Metamask and testing of networks available on the Ethereum.	
3.	Write a smart contract on a test network, for Bank account of a customer for following operations • Deposit money Withdraw Money	
4.	Create a dApp (de-centralized app) for e-voting system	
5.	Write a program in solidity to create Employee data. Use the following constructs: • Structures • Arrays • Fallback Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.	
6.	Write a survey report on types of Blockchains and its real time use cases	
7.	Mini Project to develop a complete Blockchain based system	
8.	References	

Vision of Department

To create quality computer professionals through an excellent academic environment.

Mission of Department

1. To empower students with the fundamentals of Computer Engineering for being successful professionals.
2. To motivate the students for higher studies, research, and entrepreneurship by imparting quality education.
3. To create social awareness among the students.

Program Educational Objectives: -

PEO I	**Graduate shall have successful professional careers, lead and manage teams.**
PEO II	**Graduate shall exhibit disciplinary skills to resolve real life problems.**
PEO III	**Graduate shall evolve as professionals or researchers and continue to learn emerging technologies.**

Program Outcomes: -
Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO) addressed by the Course:

Program Specific Outcomes (PSOs):
A graduate of the Computer Engineering Program will demonstrate

PSO1: Domain Specialization:

Apply domain knowledge to develop computer-based solutions for Engineering Applications.

PSO2: Problem-Solving Skills:

Find solutions for complex problems by applying problem solving skills and standard practices and strategies in software project development.

PSO3: Professional Career and Entrepreneurship:

Incorporate professional, social, ethical, effective communication, and entrepreneurial practices into their holistic development.

CO	Description
CO1	Explain the basic concepts of Blockchain Technology
CO2	Describe the fundamental concepts in cryptocurrency.
CO3	Implement a private blockchain on Ethereum Network and basics of Smart Contracts
CO4	Apply the Solidity programming concepts for creating smart contracts.
CO5	Explain the Hyperledger blockchain platform.
CO6	Understand and relate the Blockchain concepts to variety of use cases.

CO – PO Mapping

Sub code Subject	CO	PO1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CS 4102 Blockchain Technology	CO1												
	CO2												
	CO3												
	CO4												
	CO5												
	CO6												

CO – PSO Mapping

Sub code Subject	PSO	PSO1	PSO2	PSO3
CS 4102 Blockchain Technology	CO1			
	CO2			
	CO3			
	CO4			
	CO5			
	CO6			

EXPERIMENT NO: 01

Create your own wallet using Coinbase app and also Metamask for sending cryptocurrency.

Theory:

A crypto wallet is a place where you can securely keep your crypto. There are many different types of crypto wallets, but the most popular ones are hosted wallets, non-custodial wallets, and hardware wallets. The most popular and easy-to-set-up crypto wallet is a hosted wallet. When you buy crypto using an app like Coinbase, your crypto is automatically held in a hosted wallet. It's called hosted because a third party keeps your crypto for you, similar to how a bank keeps your money in a checking or savings account.

How to set up a hosted wallet:

Step1 : Select "Create a new wallet"

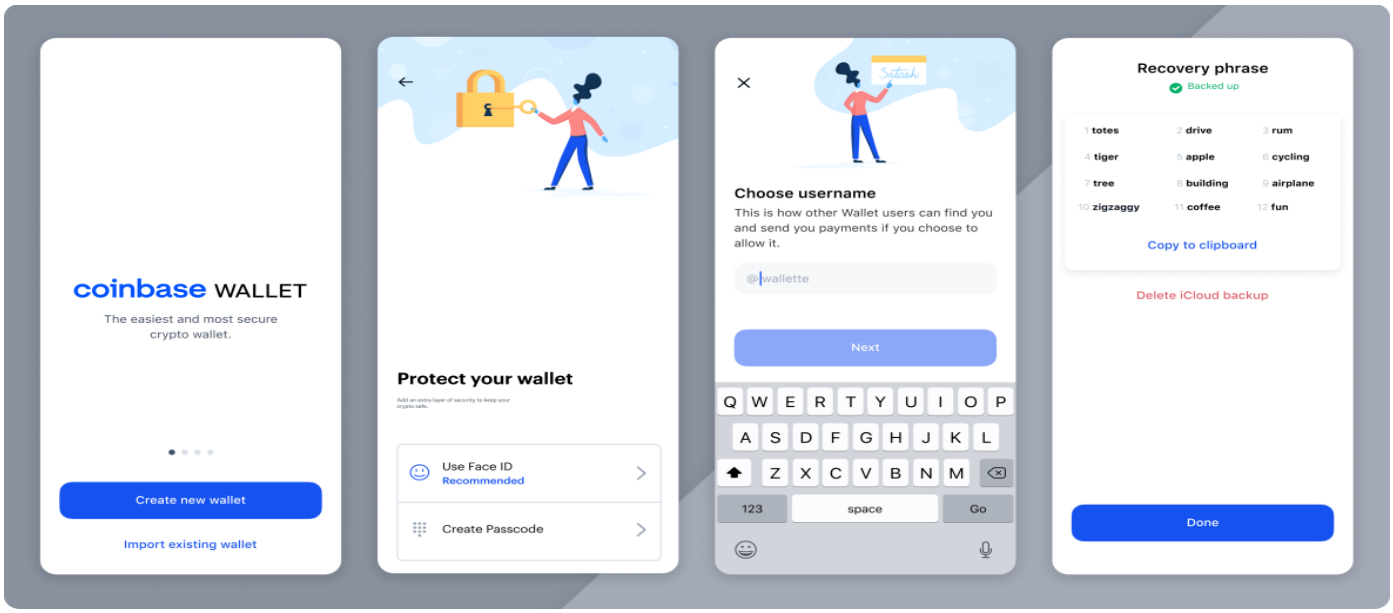
Step 2: Review and accept the Privacy Policy and Term of Service.

Step 3: Pick your username. This is how the other wallet users can find you and send you payments.

Step 4: Set your username privacy preferences.

Step 5: Choose your preferred security method, biometrics or a passcode.

Step 6: Backup your recovery phrase. Your recovery phrase is the key to your wallet.



How to import your Metamask Wallet into Coinbase wallet

a. Install the Coinbase Wallet mobile app
b. Open the MetaMask mobile app or browser extension. Enter in your password to access your wallet if prompted.
c. Click on the menu icon on the top left hand corner of the mobile app, or the top right hand corner in the extension, and navigate to Settings. From the list, click on the "Security & Privacy" option, and tap on "Reveal Secret Recovery Phrase". Enter in your password to continue
d. Copy down your Secret Recovery Phrase. This should be 12 words. Keep this safe.
e. Open the Coinbase Wallet app, and select "I already have a wallet". Then tap "Restore with Recovery Phrase"
f. Enter in the Secret Recovery Phrase that you copied down on Step 5. This should be 12 words. Keep this safe, as this is the key to access your wallet and Coinbase cannot recover this phrase for you
g. Select a Wallet username and set your privacy preferences
h. Choose your preferred security method: biometric authentication or a passcode. Keep your passcode safe, as Coinbase cannot recover this for you
i. All done - your MetaMask wallet has been imported

EXPERIMENT NO: 02

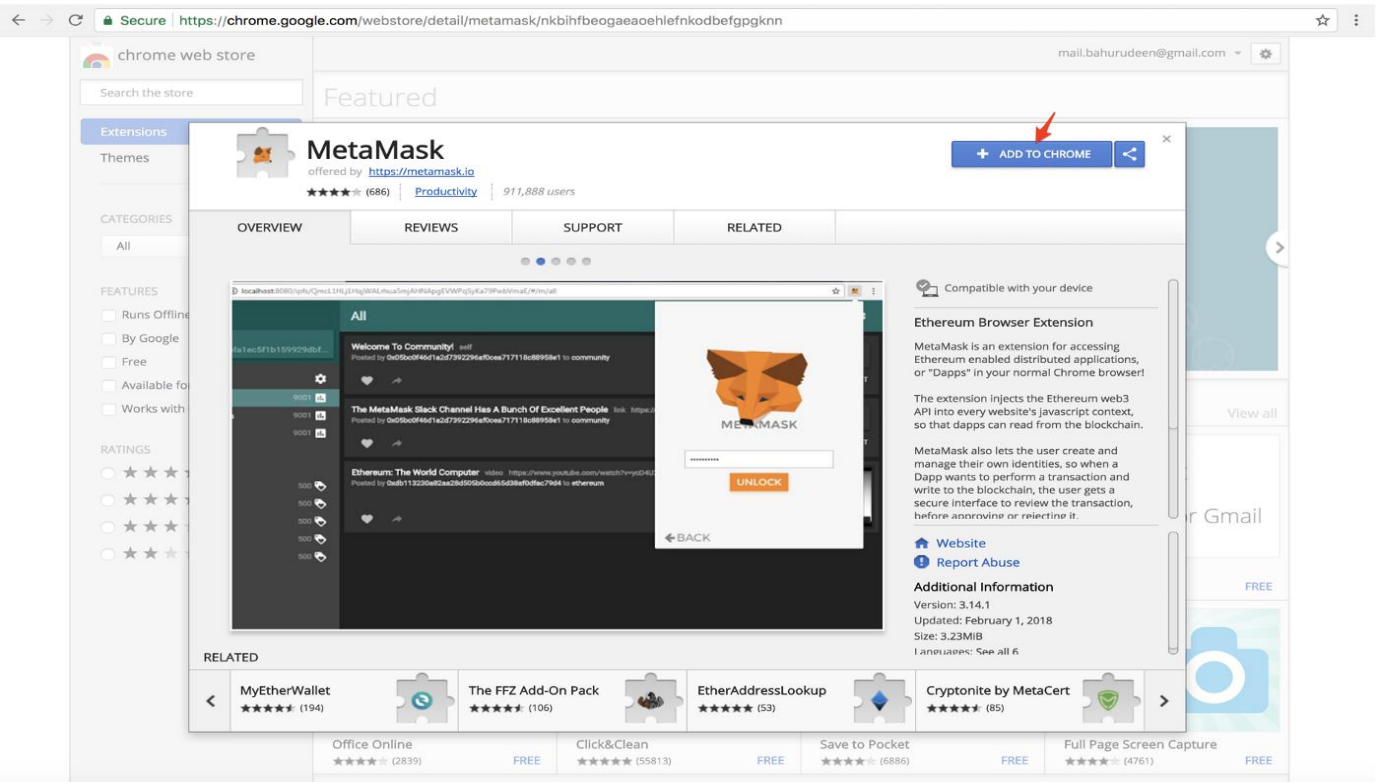
Installation of Metamask and testing of networks available on the Ethereum.

Aim: To Install of Metamask and testing of networks available on the Ethereum.

Theory:

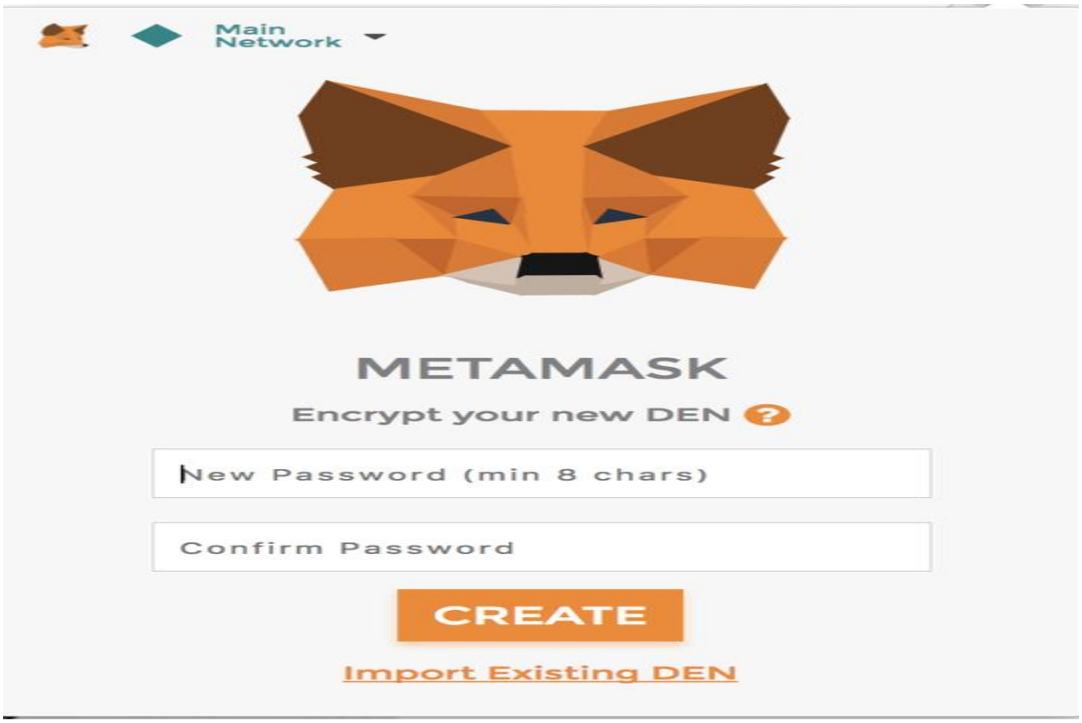
MetaMask is just an Ethereum Browser and Ether wallet. It interacts with **Ethereum Dapps** and **Smart Contracts** without running a full Ethereum node. MetaMask add-on can be installed on Chrome, Firefox, Opera, and the new Brave browser.

First, let's setup a MetaMask on Chrome browser. Open up a new Chrome browser window and navigate to **https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn** and then click on '**ADD TO CHROME**' to install MetaMask plugin.

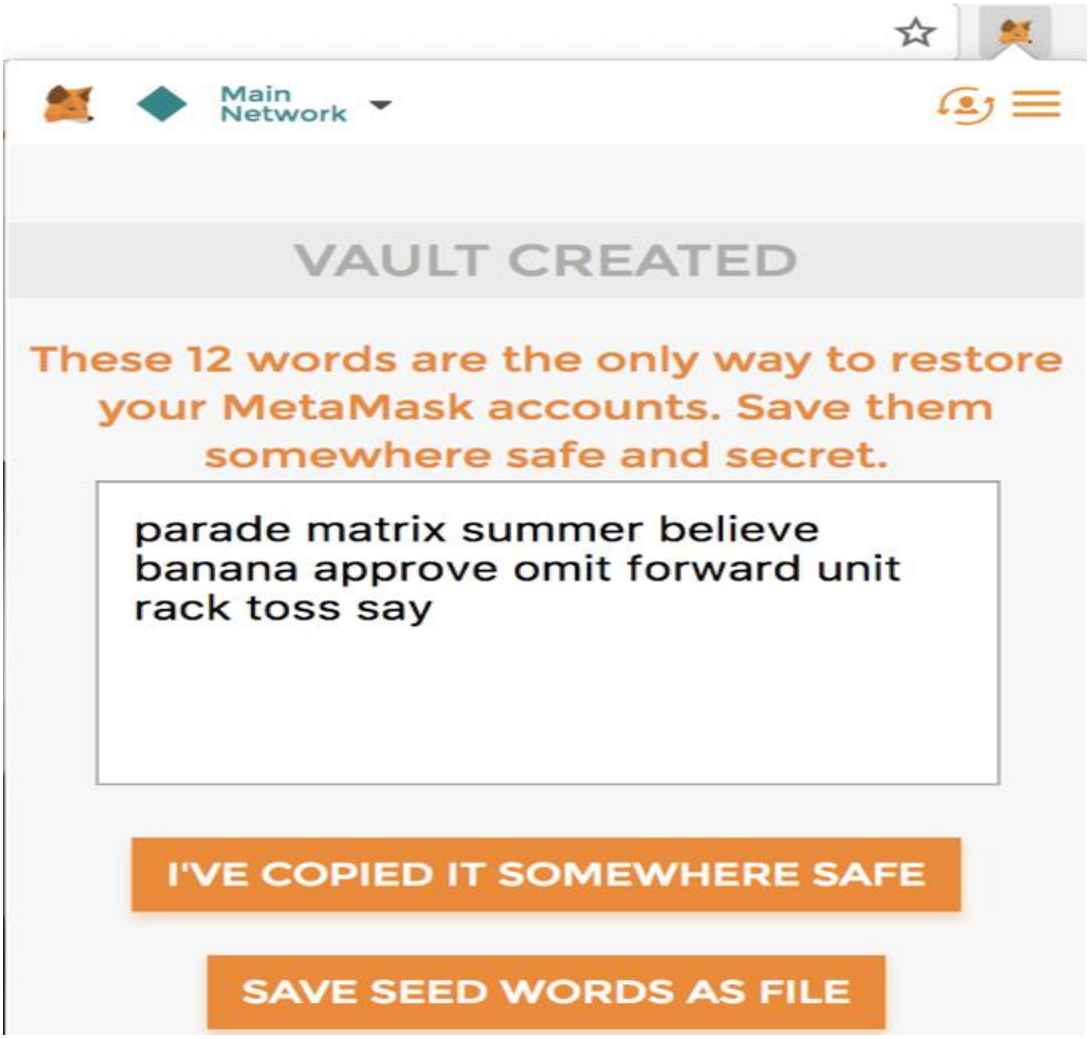


After installing, click on the MetaMask icon on the top right corner of the chrome browser. It will open up MetaMask UI, scroll all the way down and click **Accept** to agree MetaMask's terms of use.

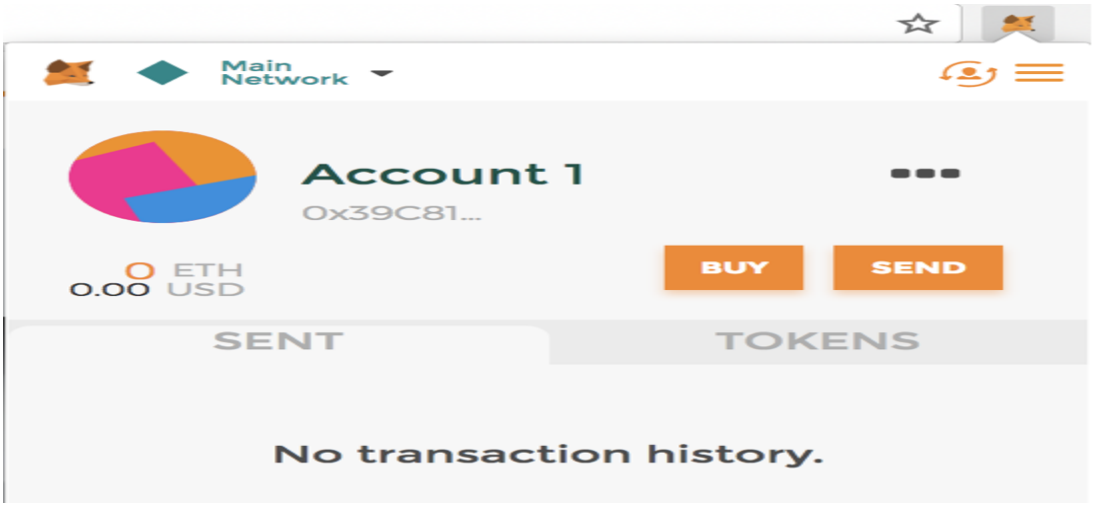
Then enter password, confirm password, and then click **Create** for a new Ethereum account. This account allows to send and receive Ether and also allows to deploy Smart Contracts.



MetaMask will show up 12 words recovery key (**Seed**) as below.



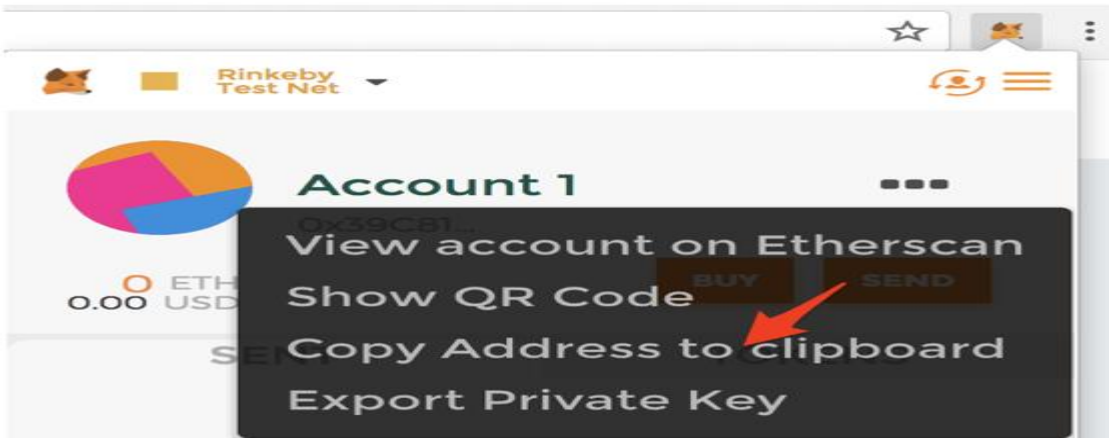
These 12 words are the only way to restore MetaMask accounts if you forget the password. It should be stored in a safe location. Then click on '**I'VE COPIED IT SOMEWHERE SAFE**'. The main screen of MetaMask will open as below.



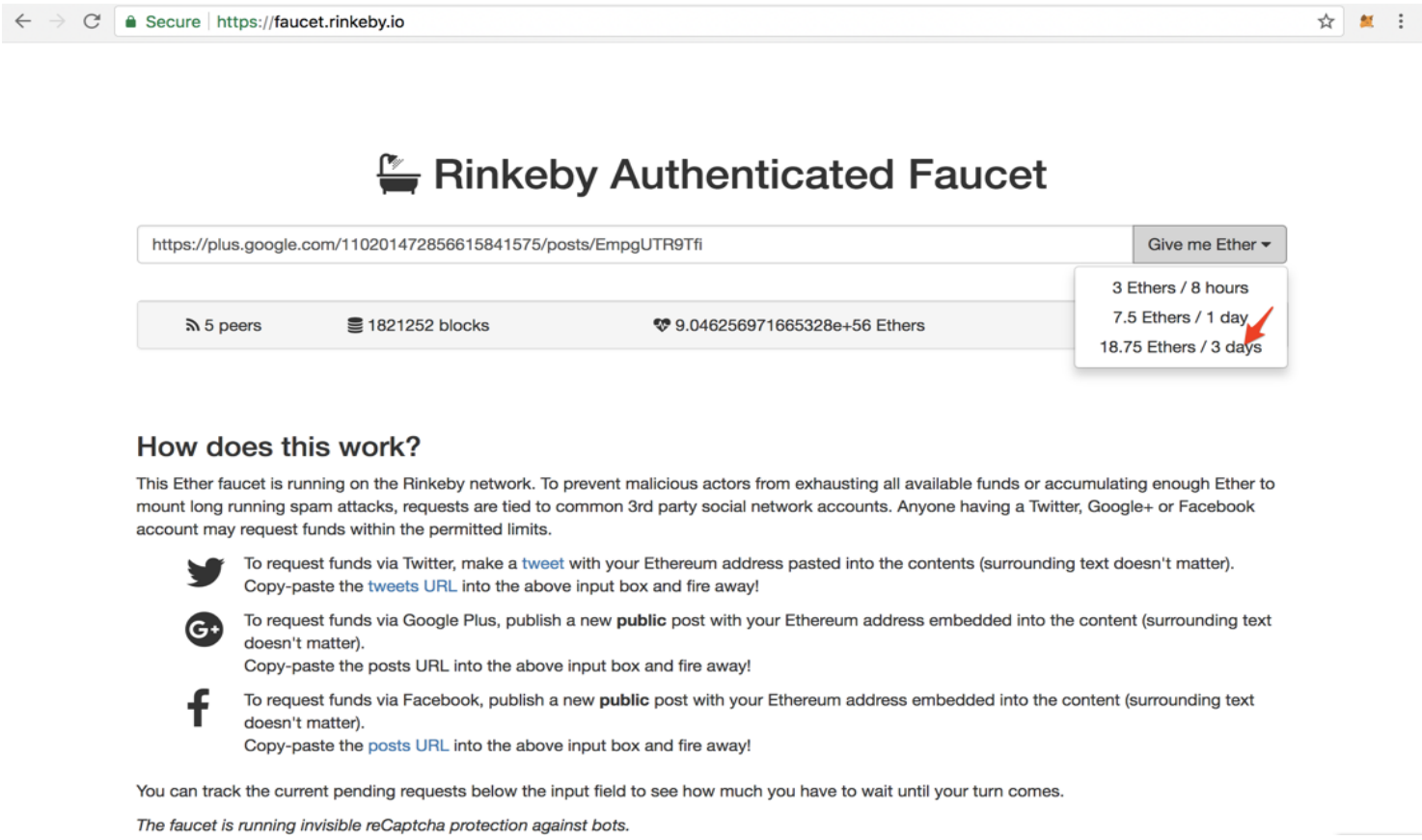
. In the upper left corner, click "**Main Network**" Then select "**Rinkeby Test Network**"

The Rinkeby Test Network is used to test the code and get free Ether to test Smart Contracts.

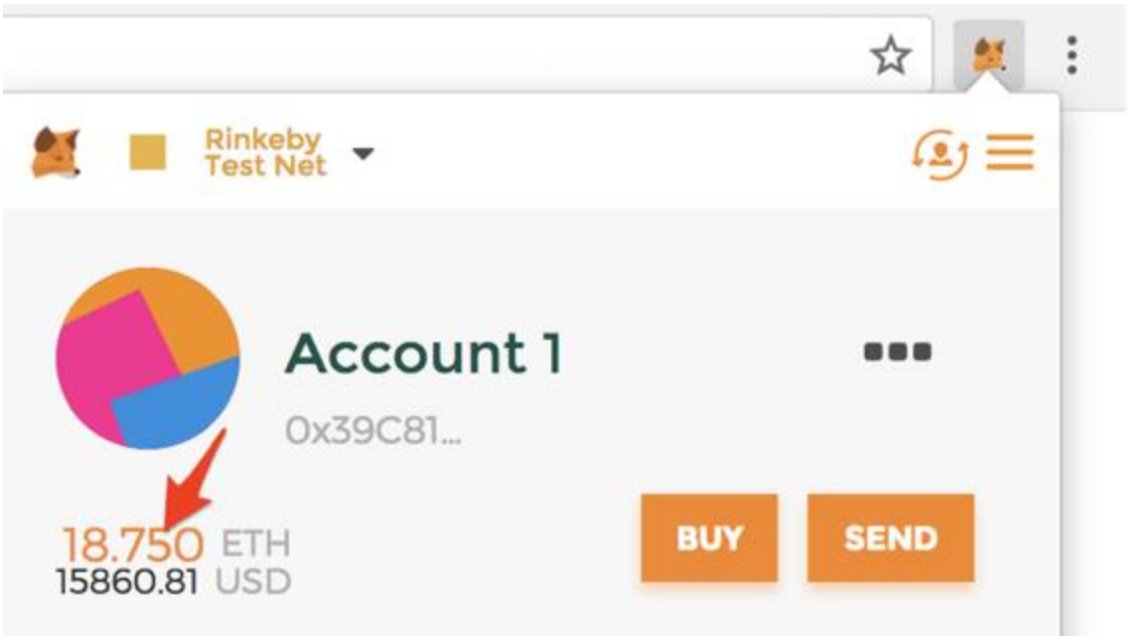
First we need to publish a new public post with Ethereum address. Open up MetaMask window, click on the 3 dots and then click on "**Copy Address to clipboard**" to get the **Account1** address.



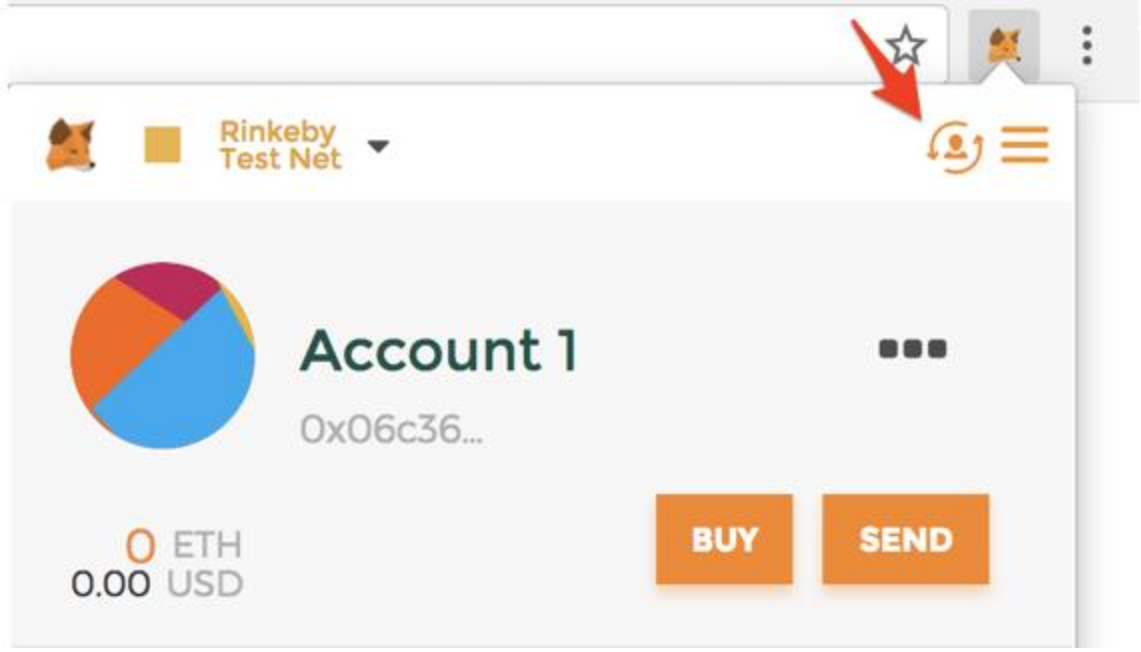
Open up a new browser window and navigate to **https://faucet.rinkeby.io/** then paste the newly created post's url on text area, and then select '**18.75 Ethers / 3 days**'. It takes some time for processing.

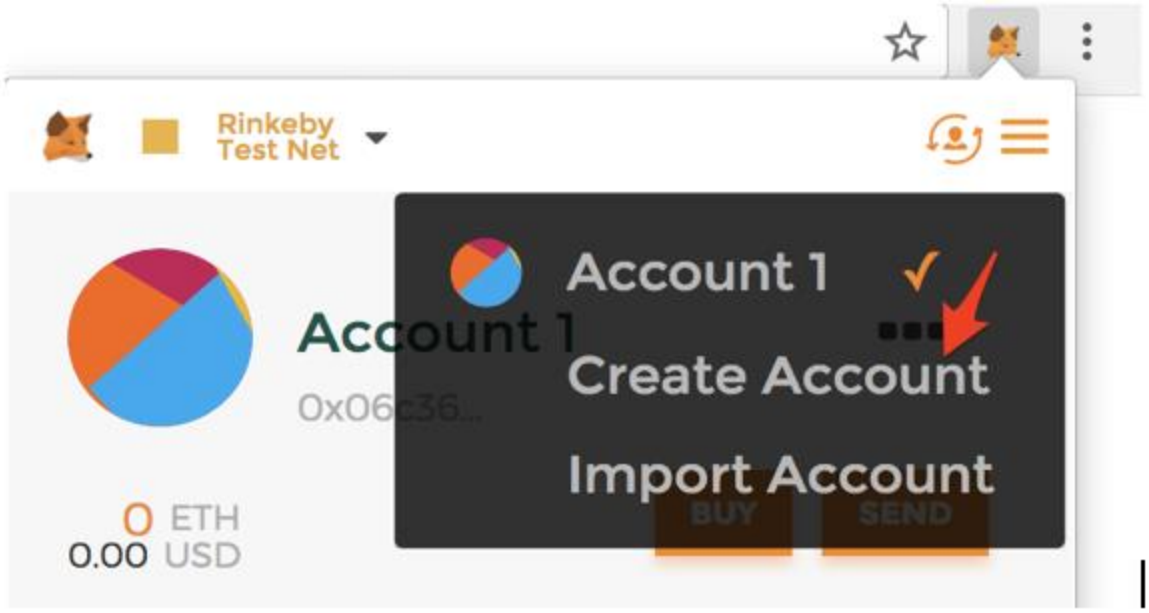


Open up MetaMask window and make sure that **18.750** ETH is added to **Account1**.



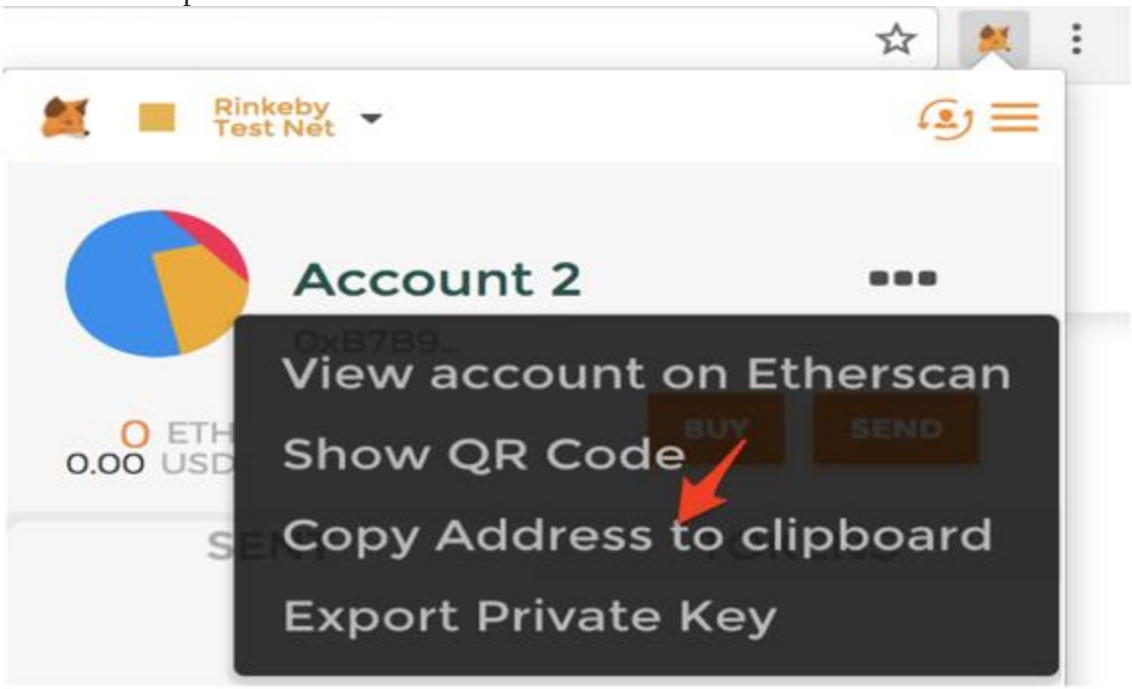
Let’s create an another account, click on **Account** menu and then click **Create Account** for another new account.



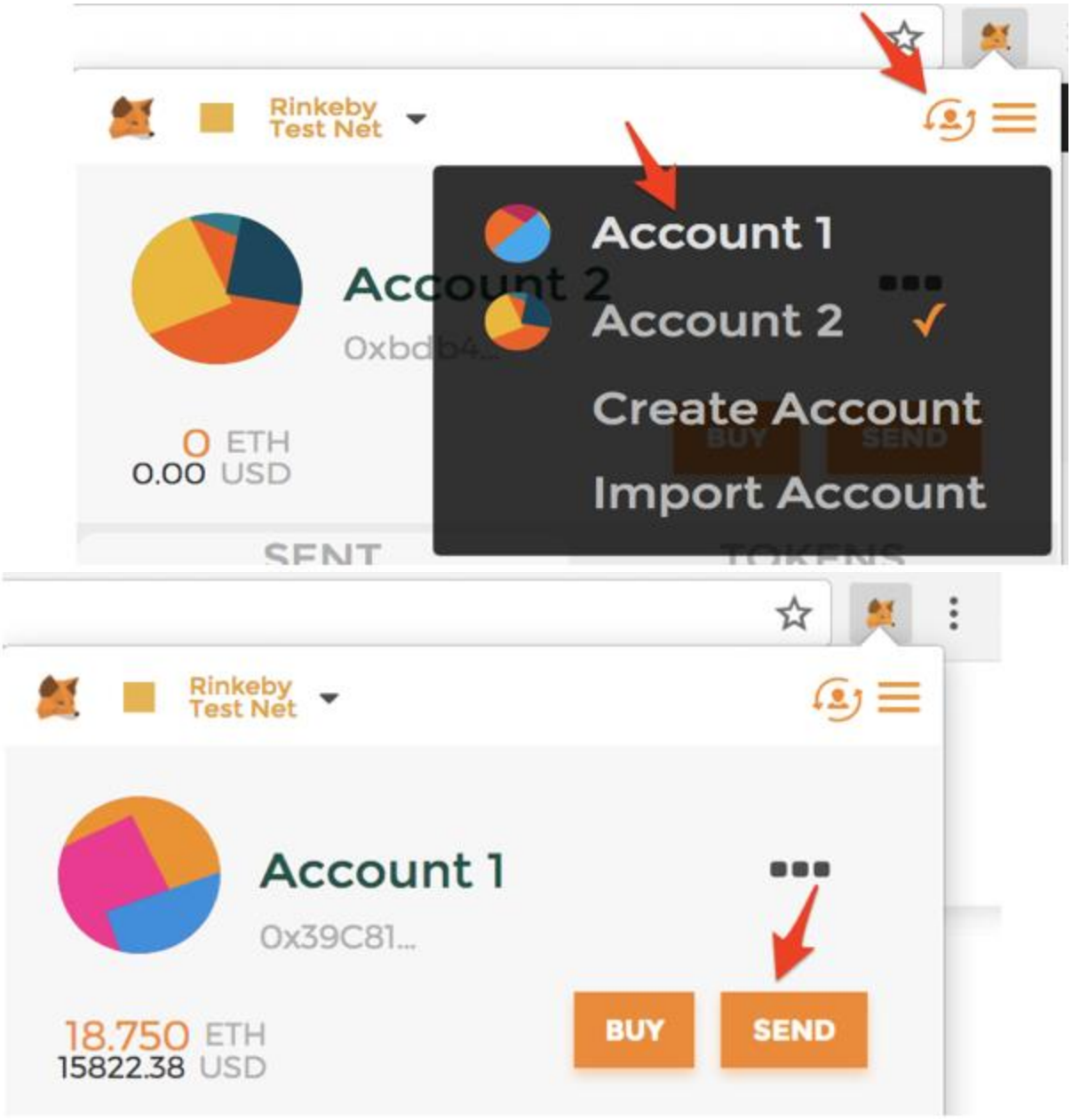


Initially **Account2** has 0 ETH available balance. Let's send **0.1** ETH to **Account2** from **Account1**.

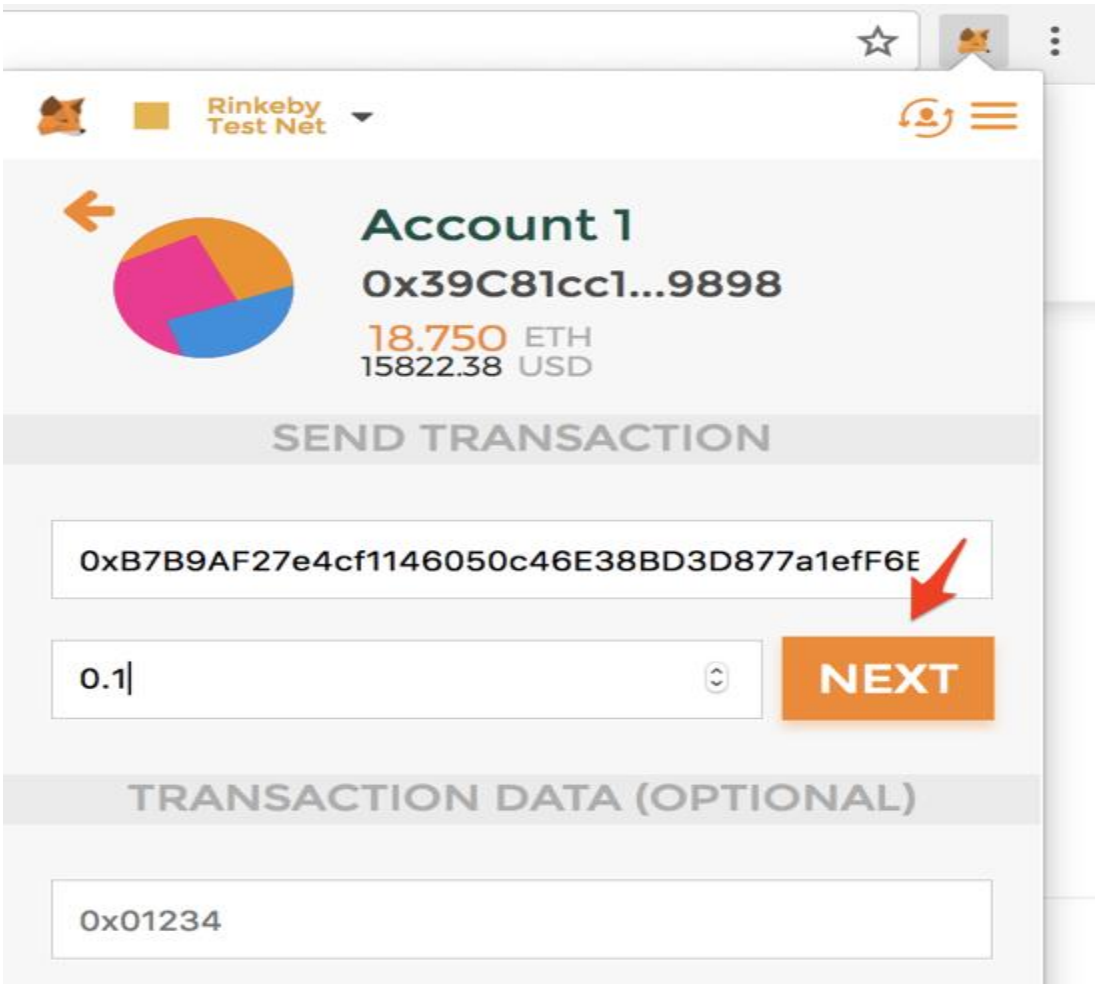
Copy **Account2** address to clipboard make sure **Account2** is selected.



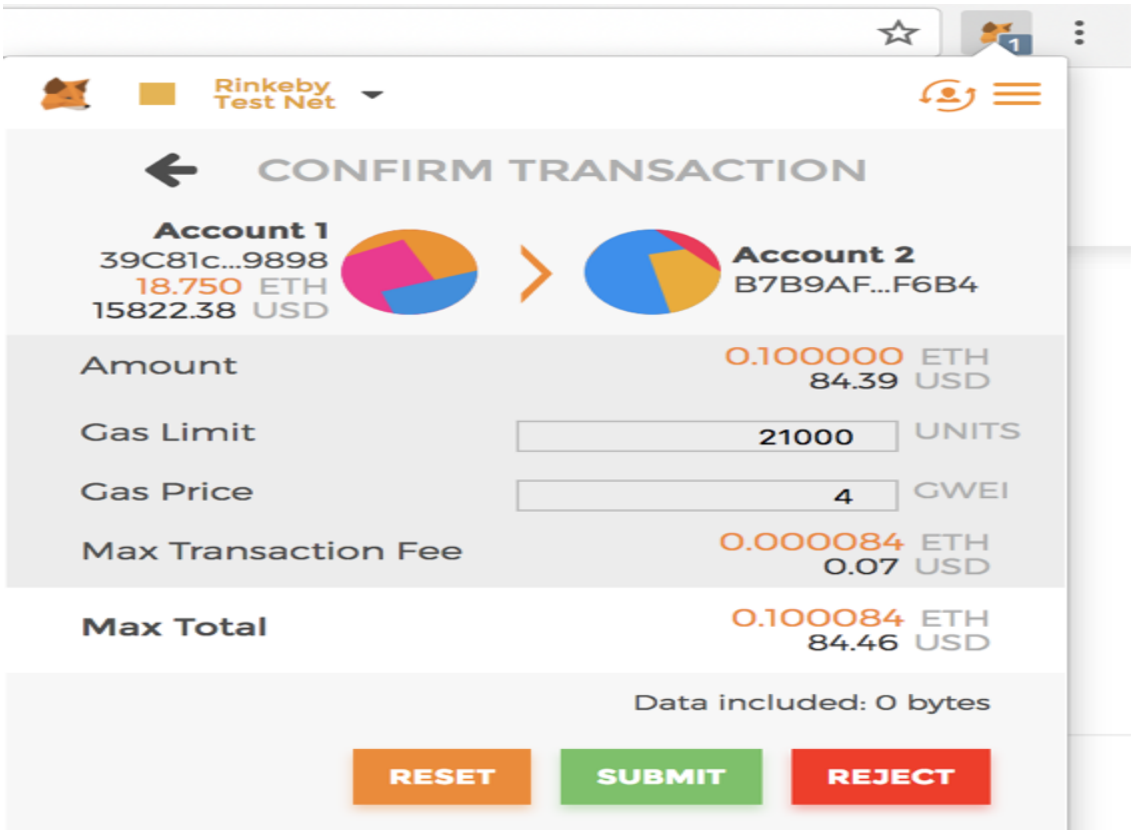
Choose **Account1** from account menu, and then click **Send**.



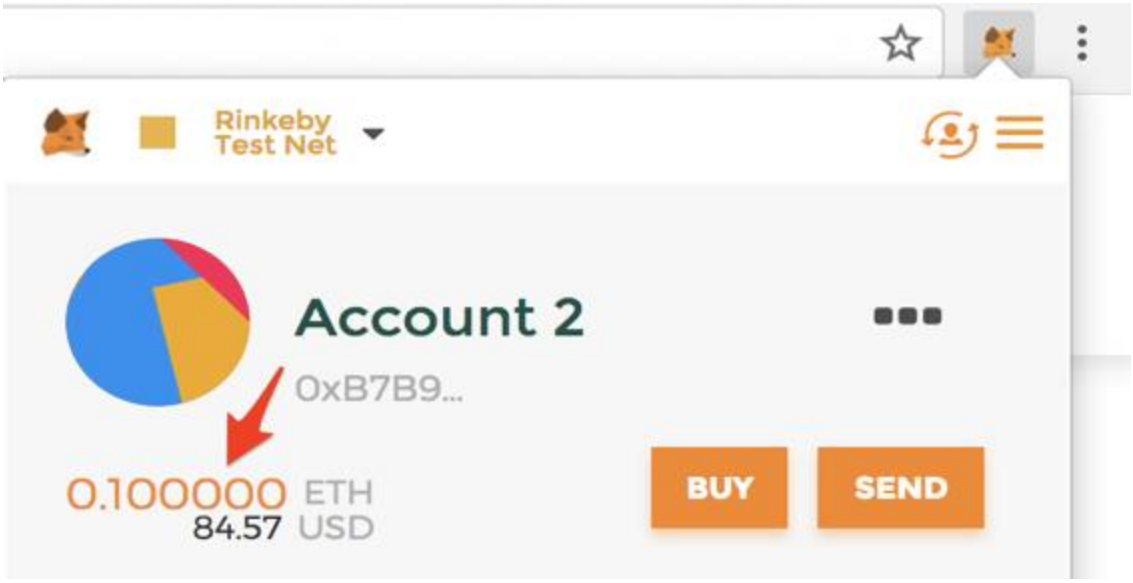
Paste **Account2** address in Recipient Address, then enter **0.1** Ether, and then click **Next**.



Then click **Submit** to confirm Transaction. It takes few seconds for processing.



Choose **Account2** and make sure **0.1** Ether is added**.**



EXPERIMENT NO: 03

Write a smart contract on a test network, for Bank account of a customer for following operations: Deposit money Withdraw Money

Aim: Write a smart contract on a test network, for Bank account of a customer for following operations

- Deposit money
- Withdraw Money

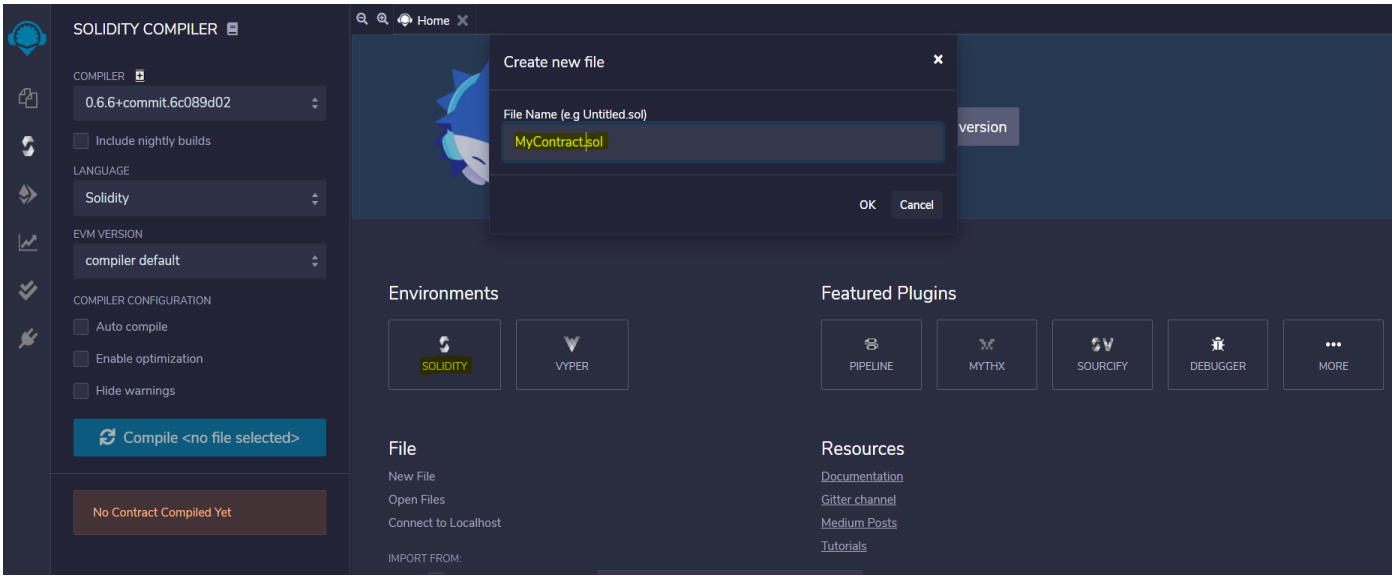
Theory:

A smart contract is a self-executing contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. The code controls the execution, and transactions are trackable and irreversible. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism.

Every smart contract is owned by an address called as owner. A smart contract can know its owner's address using sender property and its available balance using a special built-in object called msg.

Step 1: Open Remix-IDE.

Step 2: Select *File Explorer* from the left side icons and select *Solidity* in the environment. Click on *New* option below the Solidity environment. Enter the file name as ***MyContract.sol*** and Click on the *OK* button.

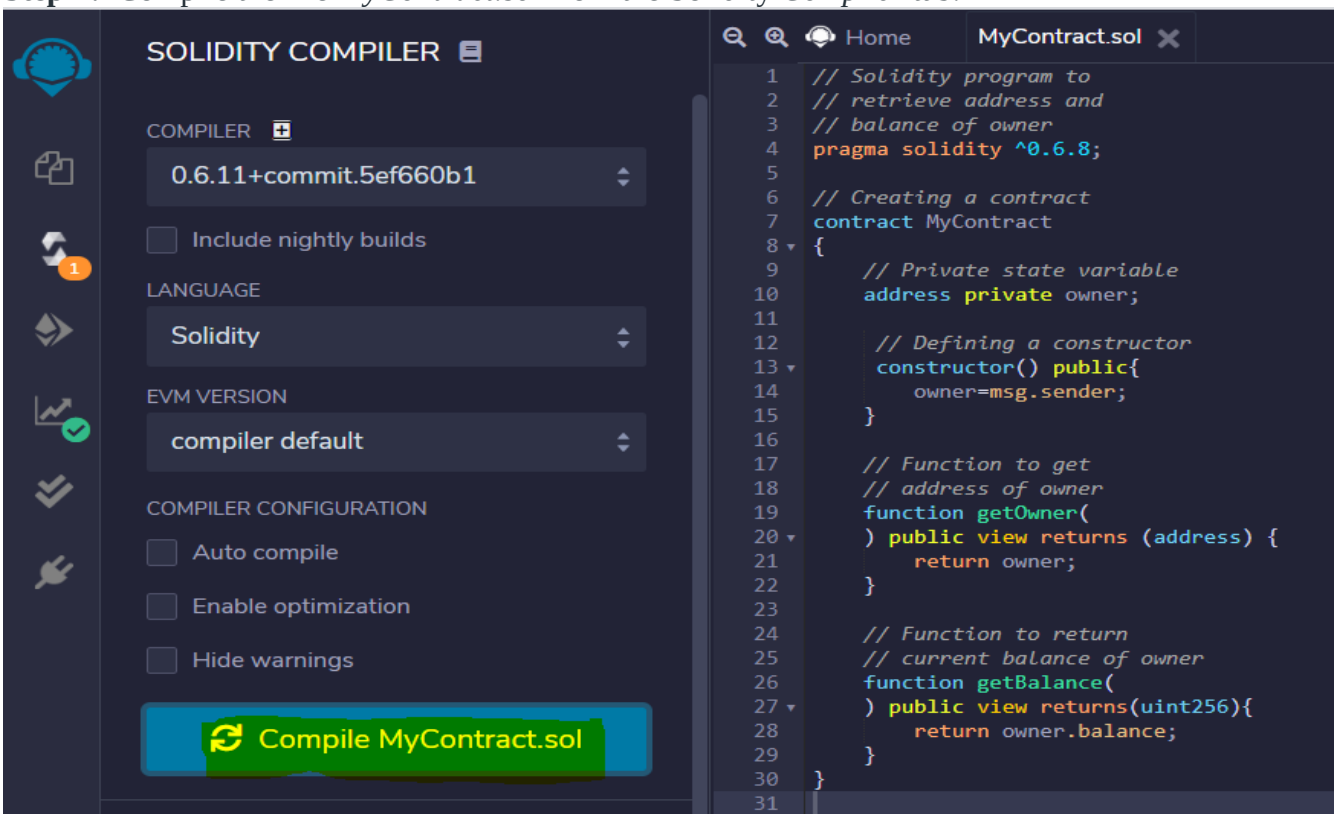


Step 3: Enter the following Solidity Code.

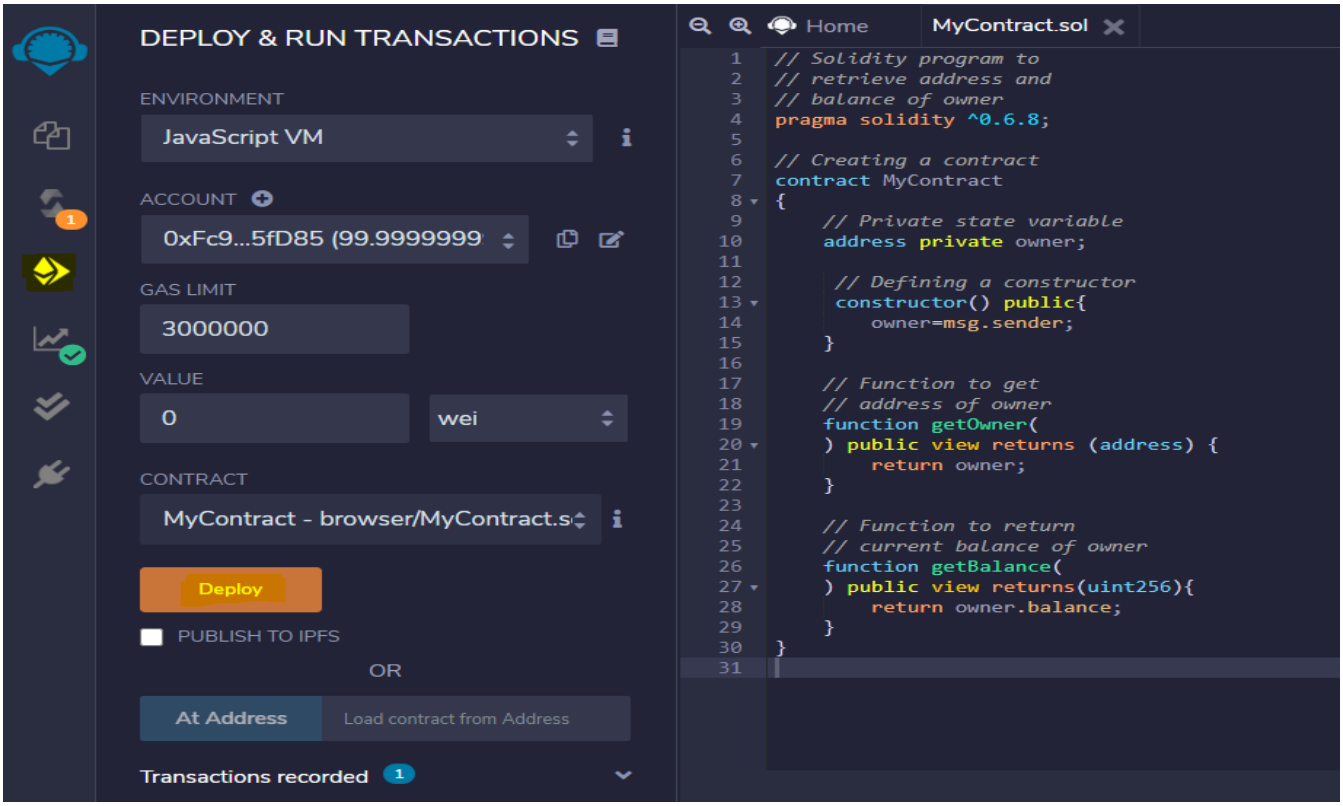
```
// Solidity program to

// retrieve address and

// balance of owner

pragma solidity ^0.6.8;

// Creating a contract
```

```
contract MyContract
{
	// Private state variable
	address private owner;
	// Defining a constructor
	constructor() public{
		owner=msg.sender;
	}
	// Function to get
	// address of owner
	function getOwner(
	) public view returns (address) {
		return owner;
	}
	// Function to return current balance of owner
	function getBalance(
	) public view returns(uint256){
		return owner.balance;
	}
}
```

Step 4: Compile the file *MyContract.sol* from the Solidity *Compiler* tab.



Step 5: Deploy the smart contract from the *Deploy and Run Transaction* tab and you will get the balance and address of the owner.



Step 6: The output below shows the address and the balance of the owner.

EXPERIMENT NO: 04

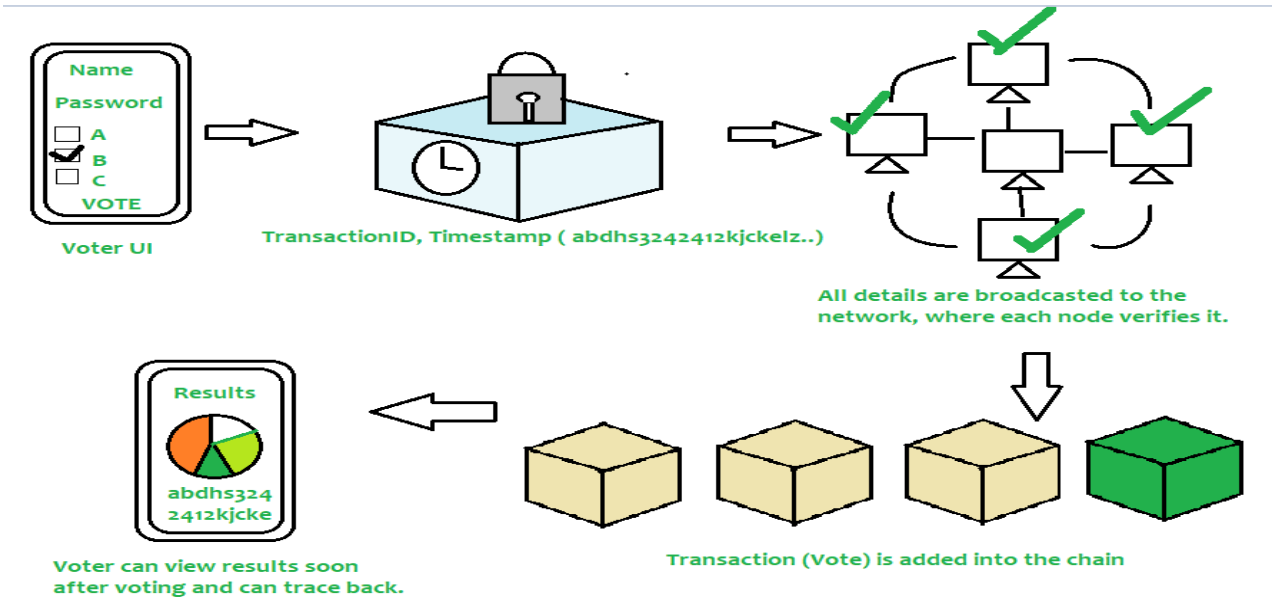
Create a dApp (de-centralized app) for e-voting system

Aim: Create a dApp (de-centralized app) for e-voting system

Theory:

Core Components of Ethereum

1. **Smart Contracts** act as the back-end logic and storage. A contract is written in Solidity, a smart contract language, and is a collection of code and data that resides at a specific address on the Ethereum blockchain. It's very similar to a class in Object Oriented Programming, where it includes functions and state variables. Smart Contracts, along with the Blockchain, are the basis of all Decentralized Applications. They are, like Blockchain, immutable and distributed, which means upgrading them will be a pain if they are already on the Ethereum Network. Fortunately, here are some ways to do that.
2. The **Ethereum Virtual Machine(EVM)** handles the internal state and computation of the entire Ethereum Network. Think of the EVM as this massive decentralized computer that contains "addresses" that are capable of executing code, changing data, and interacting with each other.
3. **Web3.js** is a Javascript API that allows you to interact with the Blockchain, including making transactions and calls to smart contracts. This API abstracts the communication with Ethereum Clients, allowing developers to focus on the content of their application. You must have a web3 instance imbedded in your browser to do so.
4. **Metamask** brings Ethereum to your browser. It is a browser extension that provides a secure web3 instance linked to your Ethereum address, allowing you to use Decentralized Applications. We will not be using Metamask in this tutorial, but it is a way for people to interact with your DApp in production. Instead, we will inject our own web3 instance during development.



According to above diagram, voter needs to enter his/her credentials in order to vote. All data is then encrypted and stored as a transaction. This transaction is then broadcasted to every node in network, which in turn is then verified. If network approves transaction, it is stored in a block and added to chain. Note that once a block is added into chain, it stays there forever and can't be updated. Users can now see results and also trace back transaction if they want.

Since current voting systems don't suffice to security needs of modern generation, there is a need to build a system that leverages security, convenience, and trust involved in voting process. Hence voting systems make use of Blockchain technology to add an extra layer of security and encourage people to vote from any time, anywhere without any hassle and makes voting process more cost-effective and time-saving.

```
# Python program to create Blockchain


# For timestamp
import datetime


# Calculating the hash
# in order to add digital
# fingerprints to the blocks
import hashlib


# To store data
# in our blockchain
import JSON


# Flask is for creating the web
# app and jsonify is for
# displaying the blockchain
from flask import Flask, jsonify



class Blockchain:


        # This function is created
        # to create the very first
        # block and set its hash to "0"
        def __init__(self):
                self.chain = []
```

```
        self.create_block(proof=1, previous_hash='0')

# This function is created
# to add further blocks
# into the chain
def create_block(self, proof, previous_hash):
        block = {'index': len(self.chain) + 1,
                        'timestamp': str(datetime.datetime.now()),
                        'proof': proof,
                        'previous_hash': previous_hash}
        self.chain.append(block)
        return block

# This function is created
# to display the previous block
def print_previous_block(self):
        return self.chain[-1]

# This is the function for proof of work
# and used to successfully mine the block
def proof_of_work(self, previous_proof):
        new_proof = 1
        check_proof = False

        while check_proof is False:
                hash_operation = hashlib.sha256(
                        str(new_proof**2 - previous_proof**2).encode()).hexdigest()
                if hash_operation[:5] == '00000':
                        check_proof = True
                else:
                        new_proof += 1

        return new_proof
```

```
        def hash(self, block):
                encoded_block = json.dumps(block, sort_keys=True).encode()
                return hashlib.sha256(encoded_block).hexdigest()

        def chain_valid(self, chain):
                previous_block = chain[0]
                block_index = 1

                while block_index < len(chain):
                        block = chain[block_index]
                        if block['previous_hash'] != self.hash(previous_block):
                                return False

                        previous_proof = previous_block['proof']
                        proof = block['proof']
                        hash_operation = hashlib.sha256(
                                str(proof**2 - previous_proof**2).encode()).hexdigest()

                        if hash_operation[:5] != '00000':
                                return False
                        previous_block = block
                        block_index += 1

                return True
# Creating the Web
# App using flask
app = Flask(__name__)

# Create the object
# of the class blockchain
blockchain = Blockchain()

# Mining a new block
@app.route('/mine_block', methods=['GET'])
```

```
def mine_block():
        previous_block = blockchain.print_previous_block()
        previous_proof = previous_block['proof']
        proof = blockchain.proof_of_work(previous_proof)
        previous_hash = blockchain.hash(previous_block)
        block = blockchain.create_block(proof, previous_hash)

        response = {'message': 'A block is MINED',
                                'index': block['index'],
                                'timestamp': block['timestamp'],
                                'proof': block['proof'],
                                'previous_hash': block['previous_hash']}

        return jsonify(response), 200

# Display blockchain in json format
@app.route('/get_chain', methods=['GET'])
def display_chain():
        response = {'chain': blockchain.chain,
                                'length': len(blockchain.chain)}
        return jsonify(response), 200

# Check validity of blockchain
@app.route('/valid', methods=['GET'])
def valid():
        valid = blockchain.chain_valid(blockchain.chain)

        if valid:
                response = {'message': 'The Blockchain is valid.'}
        else:
                response = {'message': 'The Blockchain is not valid.'}
        return jsonify(response), 200
```

Run the flask server locally

app.run(host='127.0.0.1', port=5000)

Output (mine_block):


```
{
"index":2,
"message":"A block is MINED",
"previous_hash":"2d83a826f87415edb31b7e12b35949b9dbf702aee7e383cbab119456847b957c"
,
"proof":533,
"timestamp":"2020-06-01 22:47:59.309000"
}
```


Output (get_chain):


```
{
"chain":[{"index":1,
"previous_hash":"0",
"proof":1,
"timestamp":"2020-06-01 22:47:05.915000"},{"index":2,
"previous_hash":"2d83a826f87415edb31b7e12b35949b9dbf702aee7e383cbab119456847b957c"
,
"proof":533,
"timestamp":"2020-06-01 22:47:59.309000"}],
"length":2
}
```


Output(valid):


```
{"message":"The Blockchain is valid."}
```

EXPERIMENT NO: 05

Write a program in solidity to create Employee data. Use the following constructs:

- **Structures**
- **Arrays**
- **Fallback**

Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Aim: To Implement Structures, Arrays and Fallback

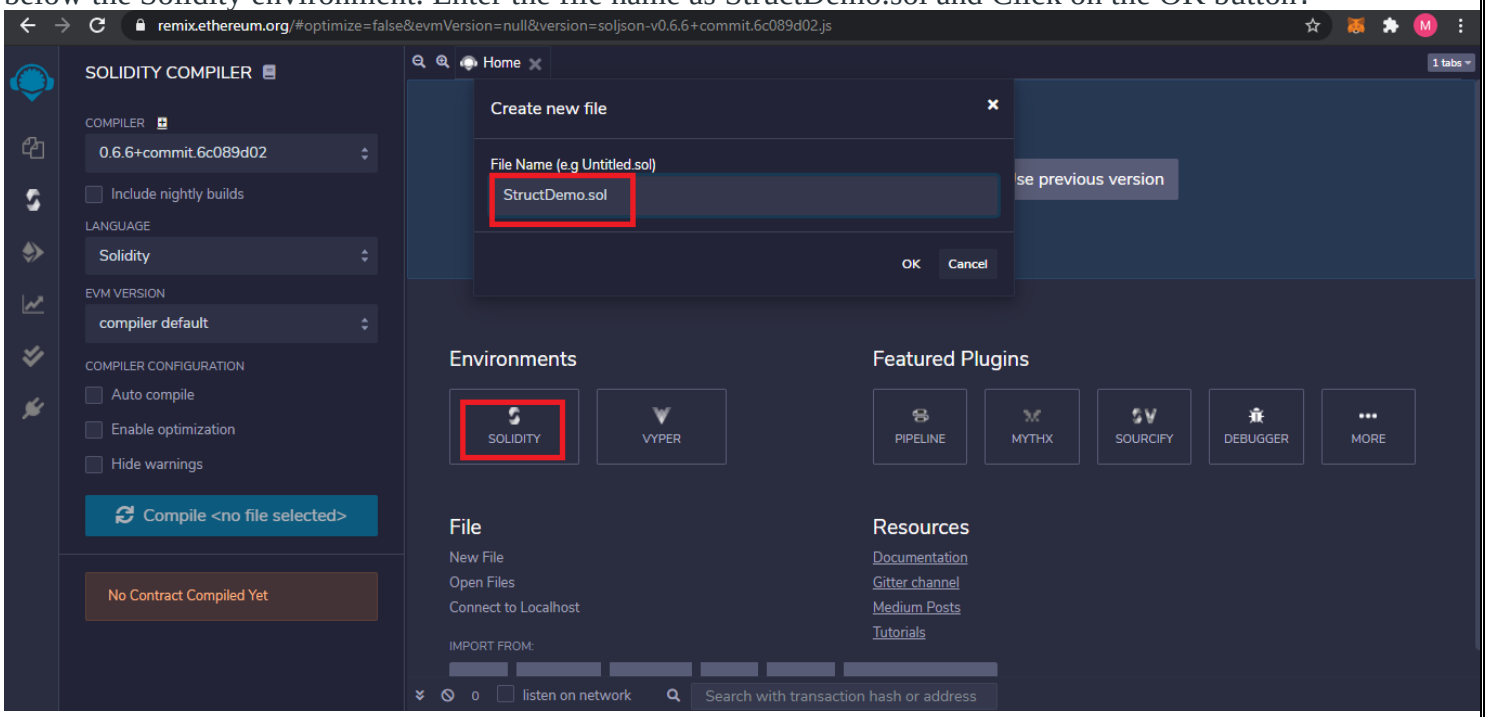
Theory:

1. Create a smart contract, StructDemo having structure Employee with data members as empid, name, department, designation. Create a dynamic array of Employee as emps.
2. Create a function addEmployee() which takes the data of the employee and stores into a dynamic array called emps.
3. Create a function getEmployee() which takes the employee id, searches the record in the emps array, and returns the details like name, department, and designation.

Implementation:

Step 1: Open Remix-IDE.

Step 2: Select File Explorer from the left side icons and select Solidity in the environment. Click on New option below the Solidity environment. Enter the file name as StructDemo.sol and Click on the OK button.



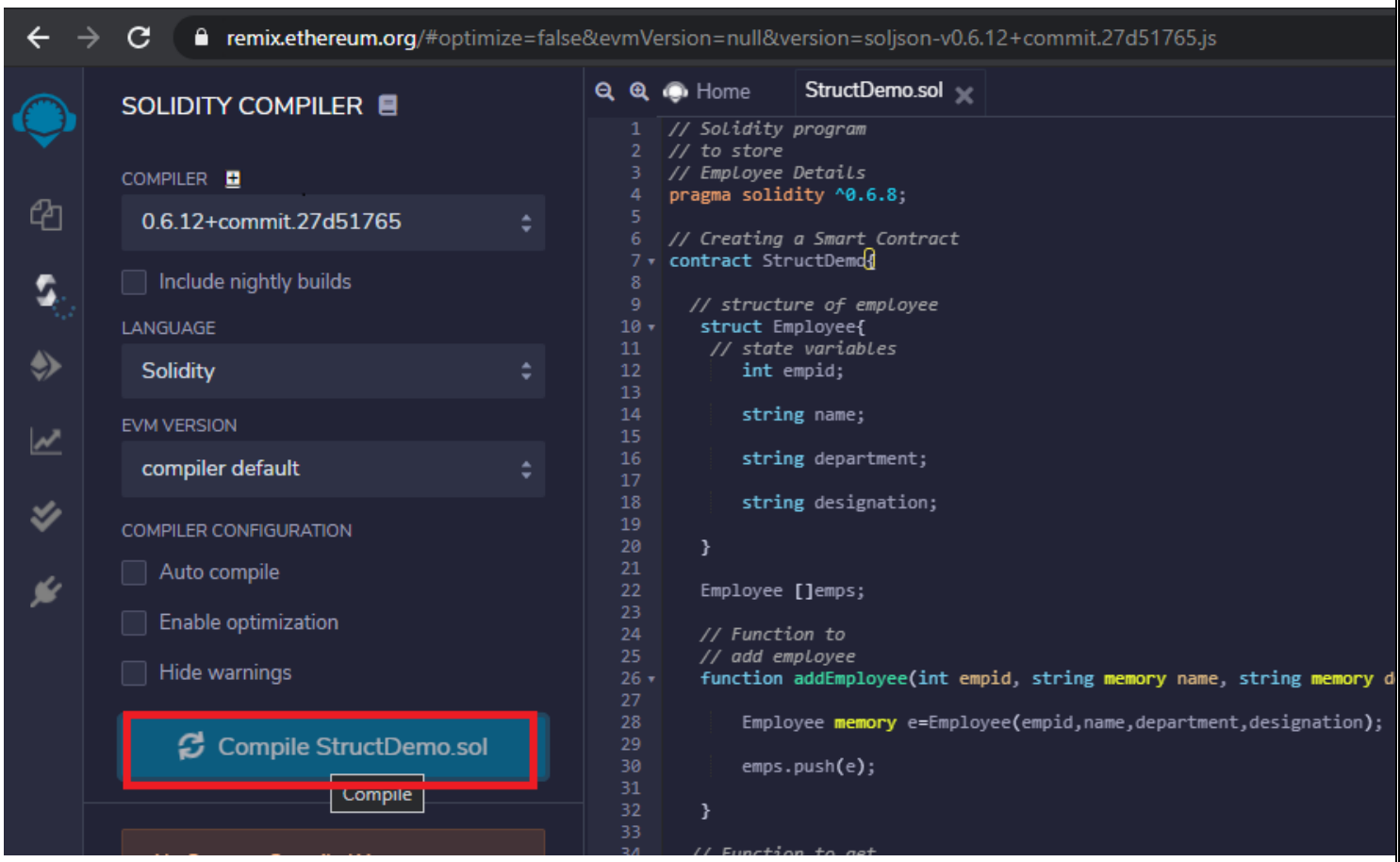
Step 3: Enter the following Solidity Code. Select the same solidity version as in your code.

- Solidity

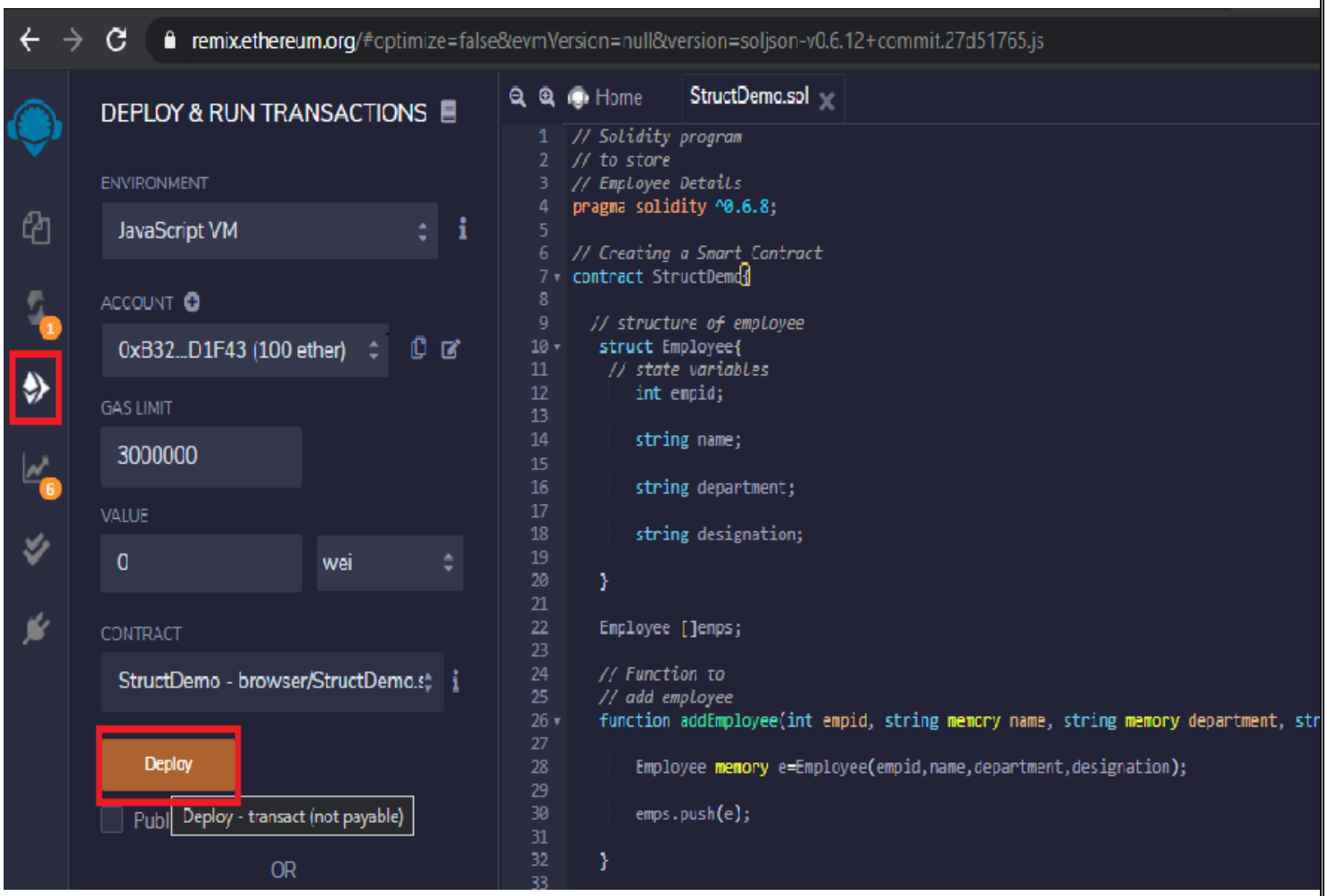
```
// Solidity program
// to store
// Employee Details
pragma solidity ^0.6.8;

// Creating a Smart Contract
contract StructDemo{

   // Structure of employee
   struct Employee{

       // State variables
       int empid;
       string name;
       string department;
       string designation;
   }

   Employee []emps;

   // Function to add
   // employee details
   function addEmployee(
     int empid, string memory name,
     string memory department,
     string memory designation
   ) public{
       Employee memory e
         =Employee(empid,
                   name,
                   department,
                   designation);
       emps.push(e);
   }

  // Function to get
  // details of employee
   function getEmployee(
     int empid
   ) public view returns(
     string memory,
     string memory,
     string memory){
       uint i;
       for(i=0;i<emps.length;i++)
```

```
        {
            Employee memory e
              =emps[i];

            // Looks for a matching
            // employee id
            if(e.empid==empid)
            {
                    return(e.name,
                        e.department,
                        e.designation);
            }
        }

      // If provided employee
      // id is not present
      // it returns Not
      // Found
      return("Not Found",
             "Not Found",
             "Not Found");
    }
}
```

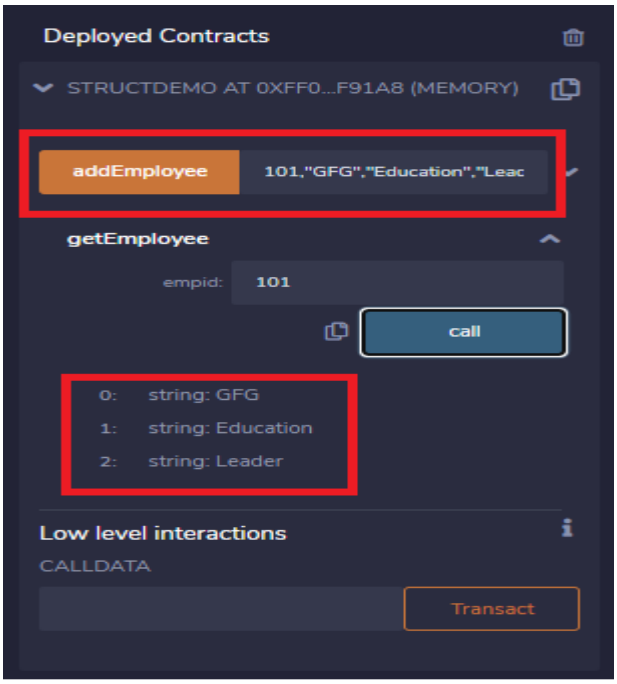
Step 4: Compile the file StructDemo.sol from the Solidity Compiler tab.



Step 5: Deploy the smart contract from the Deploy and Run Transaction tab.



Step 6: Then add the employee details through **addEmployee()** then after that, you can view details of any employee using employee id through **getEmployee().**



EXPERIMENT NO: 06

Write a survey report on types of Blockchains and its real time use cases