

Group C

Assignment No: 1

Title of the Assignment: Installation of MetaMask and study spending Ether per transaction

Objective of the Assignment: Students should be able to learn new technology such as metamask. Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain
-

Contents for Theory:

1. Introduction Blockchain
 2. Cryptocurrency
 3. Transaction Wallets
 4. Ether transaction
 5. Installation Process of Metamask
-

Introduction to Blockchain

- Blockchain can be described as a data structure that holds transactional records and while ensuring security, transparency, and decentralization. You can also think of it as a chain or records stored in the forms of blocks which are controlled by no single authority.
- A blockchain is a distributed ledger that is completely open to any and everyone on the network. Once an information is stored on a blockchain, it is extremely difficult to change or alter it.
- Each transaction on a blockchain is secured with a digital signature that proves its authenticity. Due to the use of encryption and digital signatures, the data stored on the blockchain is tamper-proof and cannot be changed.
- Blockchain technology allows all the network participants to reach an agreement, commonly known as consensus. All the data stored on a blockchain is recorded digitally and has a common history which is available for all the network participants. This way, the chances of any fraudulent activity or duplication of transactions is eliminated without the need of a third-party.

Blockchain Features

The following features make the revolutionary technology of blockchain stand out:

- **Decentralized**

Blockchains are decentralized in nature meaning that no single person or group holds the authority of the overall network. While everybody in the network has the copy of the distributed ledger with them, no one can modify it on his or her own. This unique feature of blockchain allows transparency and security while giving power to the users.

- **Peer-to-Peer Network**

With the use of Blockchain, the interaction between two parties through a peer-to-peer model is easily accomplished without the requirement of any third party. Blockchain uses P2P protocol which allows all the network participants to hold an identical copy of transactions, enabling approval through a machine consensus. For example, if you wish to make any transaction from one part of the world to another, you can do that with blockchain all by yourself within a few seconds. Moreover, any interruptions or extra charges will not be deducted in the transfer.

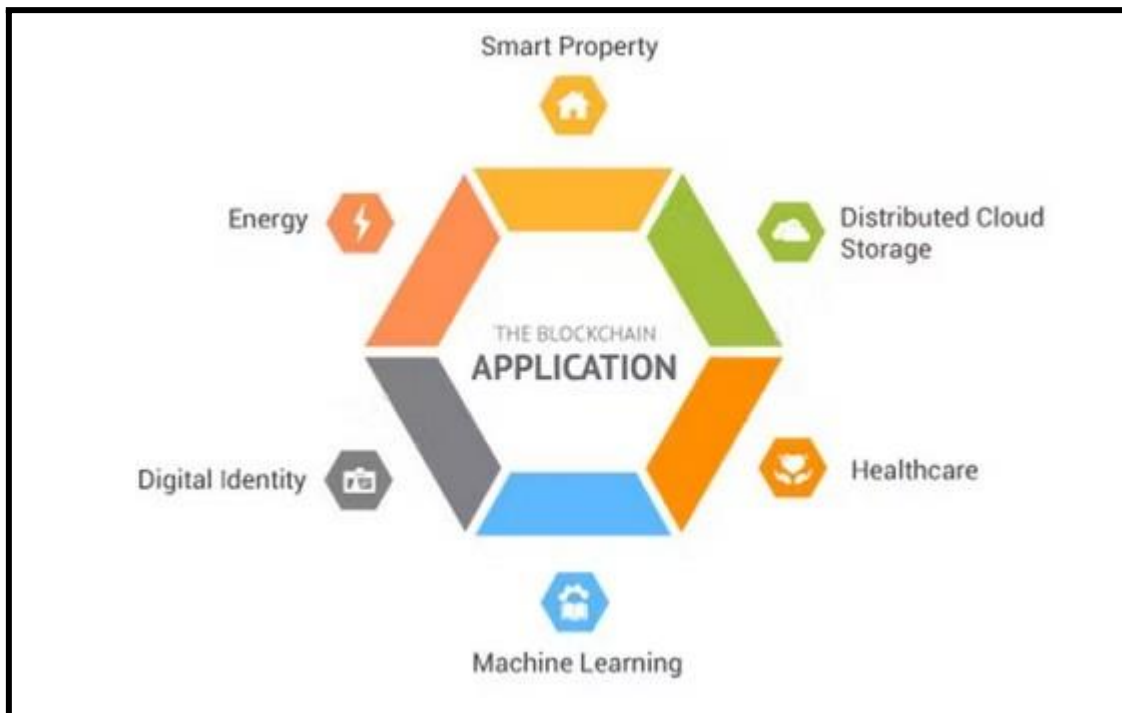
- **Immutable**

The immutability property of a blockchain refers to the fact that any data once written on the blockchain cannot be changed. To understand immutability, consider sending email as an example. Once you send an email to a bunch of people, you cannot take it back. In order to find a way around, you'll have to ask all the recipients to delete your email which is pretty tedious. This is how immutability works.

- **Tamper-Proof**

With the property of immutability embedded in blockchains, it becomes easier to detect tampering of any data. Blockchains are considered tamper-proof as any change in even one single block can be detected and addressed smoothly. There are two key ways of detecting tampering namely, hashes and blocks.

Popular Applications of Blockchain Technology



Benefits of Blockchain Technology:

- **Time-saving:** No central Authority verification needed for settlements making the process faster and cheaper.
- **Cost-saving:** A Blockchain network reduces expenses in several ways. No need for third-party verification. Participants can share assets directly. Intermediaries are reduced. Transaction efforts are minimized as every participant has a copy of shared ledger.
- **Tighter security:** No one can temper with Blockchain Data as it is shared among

millions of participants. The system is safe against cybercrimes and Fraud.

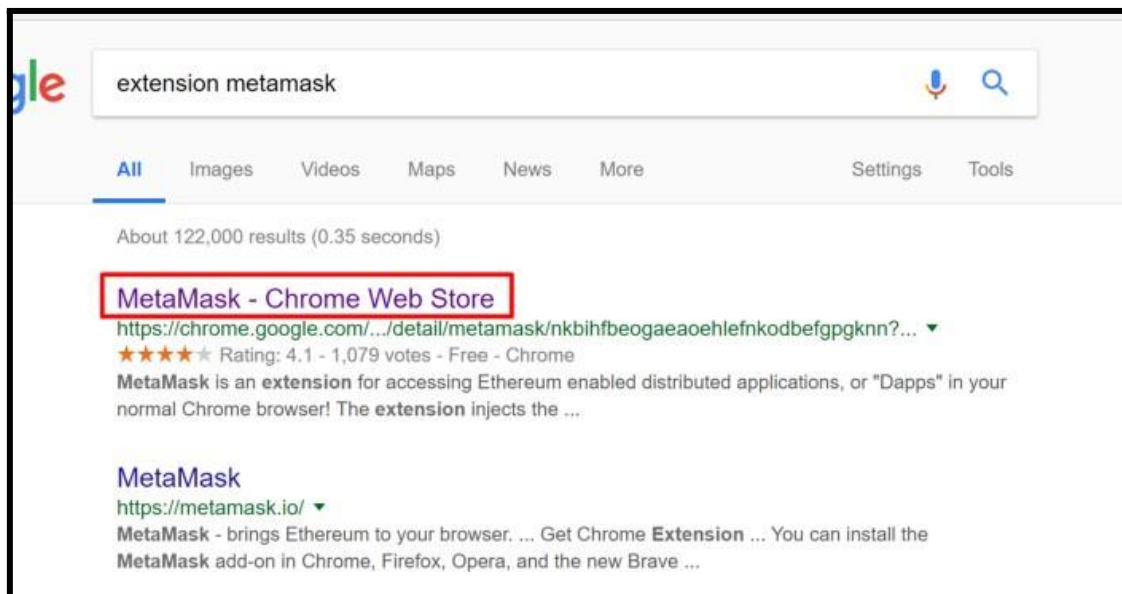
- In finance market trading, Fibonacci retracement levels are widely used in technical analysis.

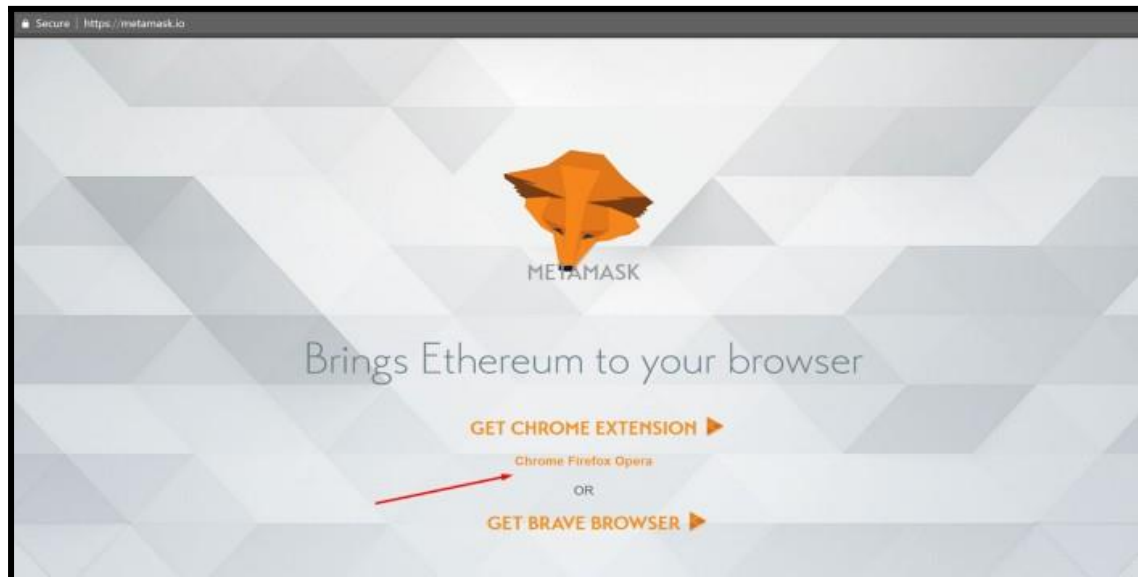
How to use MetaMask: A step by step guide

MetaMask is one of the most popular browser extensions that serves as a way of storing your Ethereum and other [ERC-20 Tokens](#). The extension is free and secure, allowing web applications to read and interact with Ethereum's blockchain.

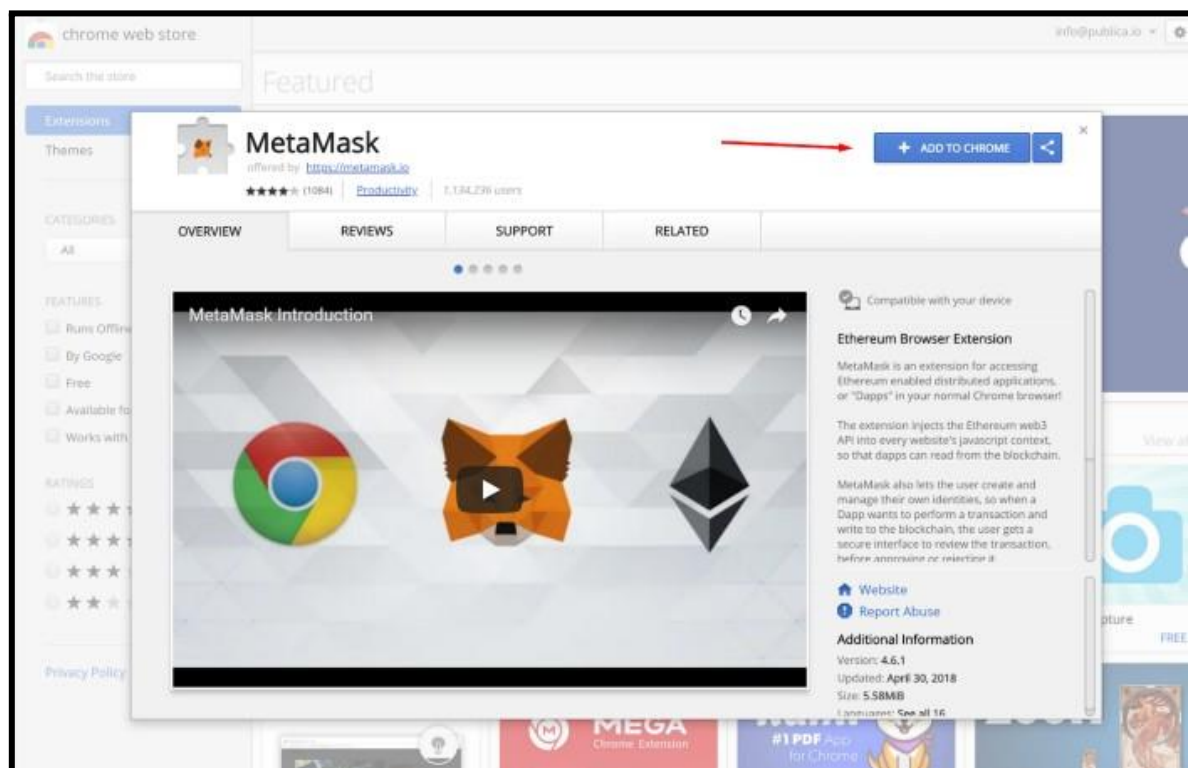
Step 1. Install MetaMask on your browser.

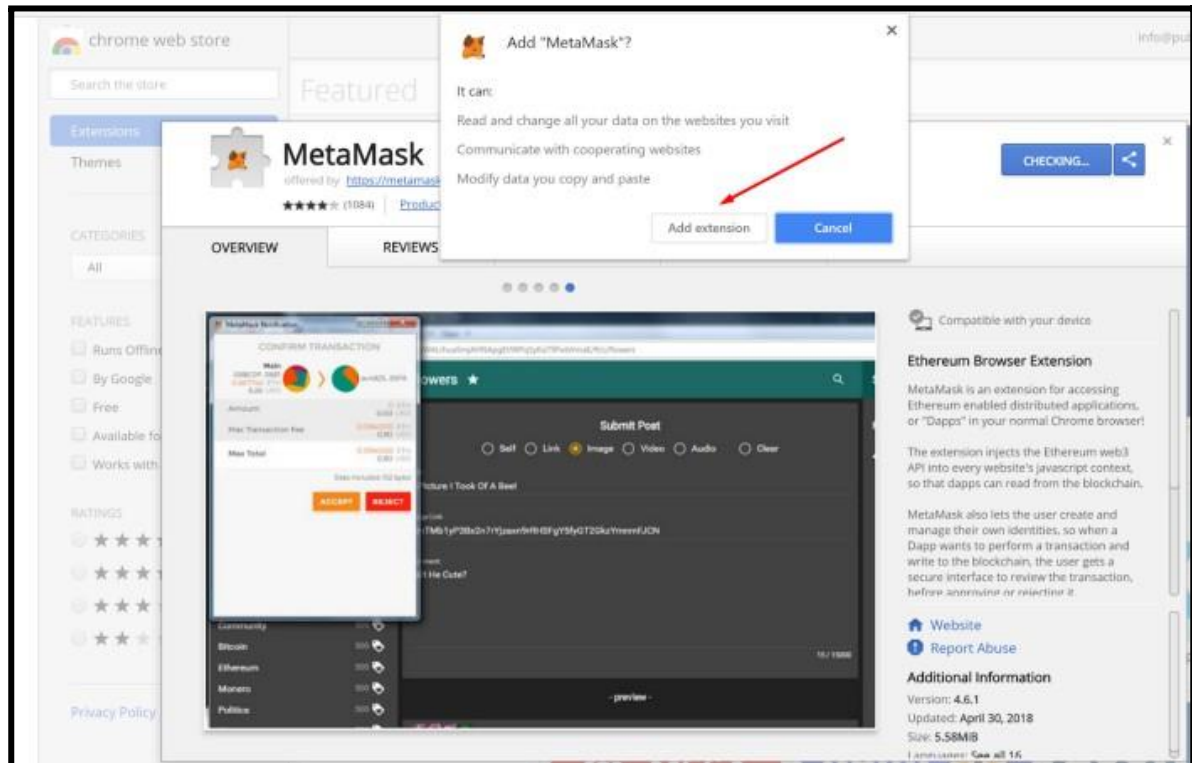
To create a new wallet, you have to install the extension first. Depending on your browser, there are different marketplaces to find it. Most browsers have MetaMask on their stores, so it's not that hard to see it, but either way, here they are [Chrome](#), [Firefox](#), and [Opera](#).





- Click on **Install MetaMask** as a Google Chrome extension.
- Click **Add to Chrome**.
- Click **Add Extension**.

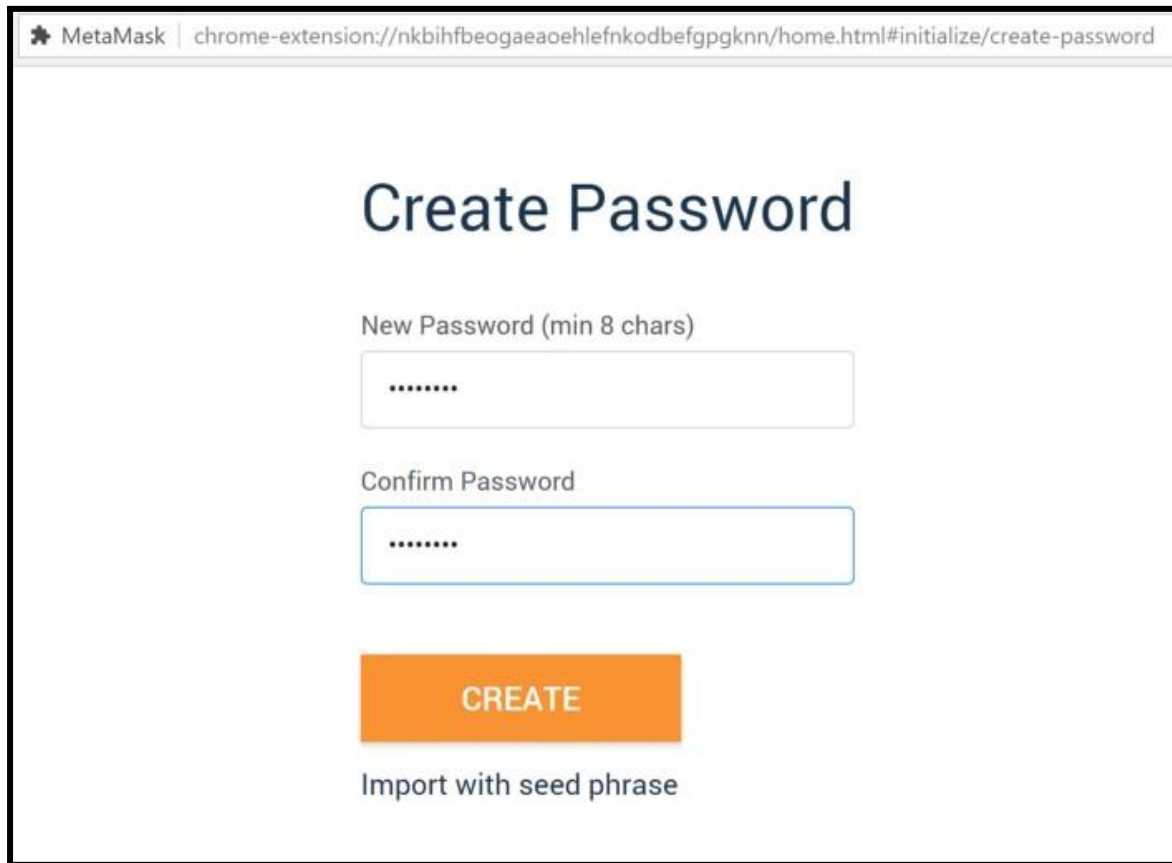




and it's as easy as that to install the extension on your browser, continue reading the next step to figure out how to create an account.

Step 2. Create an account.

- Click on the extension icon in the upper right corner to open MetaMask.
- To install the latest version and be up to date, **click Try it now.**
- **Click Continue.**
- You will be prompted to create a new password. **Click Create.**



MetaMask | chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#initialize/create-password

Create Password

New Password (min 8 chars)

.....

Confirm Password

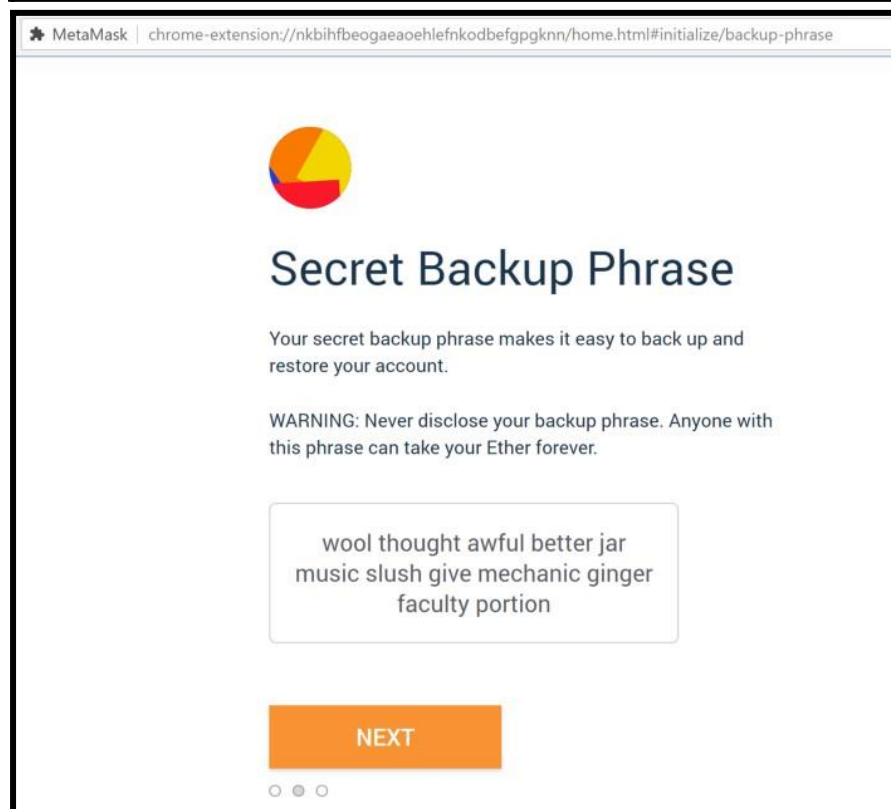
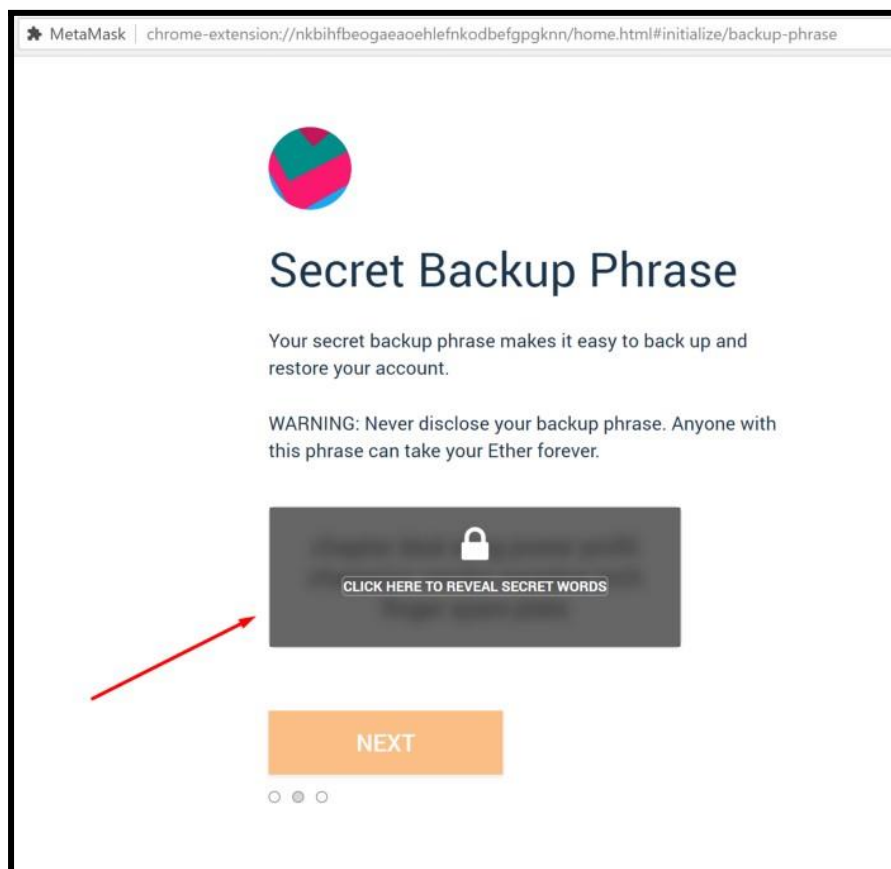
.....

CREATE

[Import with seed phrase](#)

- Proceed by **clicking Next** and accept the Terms of Use.

Click Reveal Secret Words. There you will see a 12 words seed phrase. This is really important and usually not a good idea to store digitally, so take your time and write it down



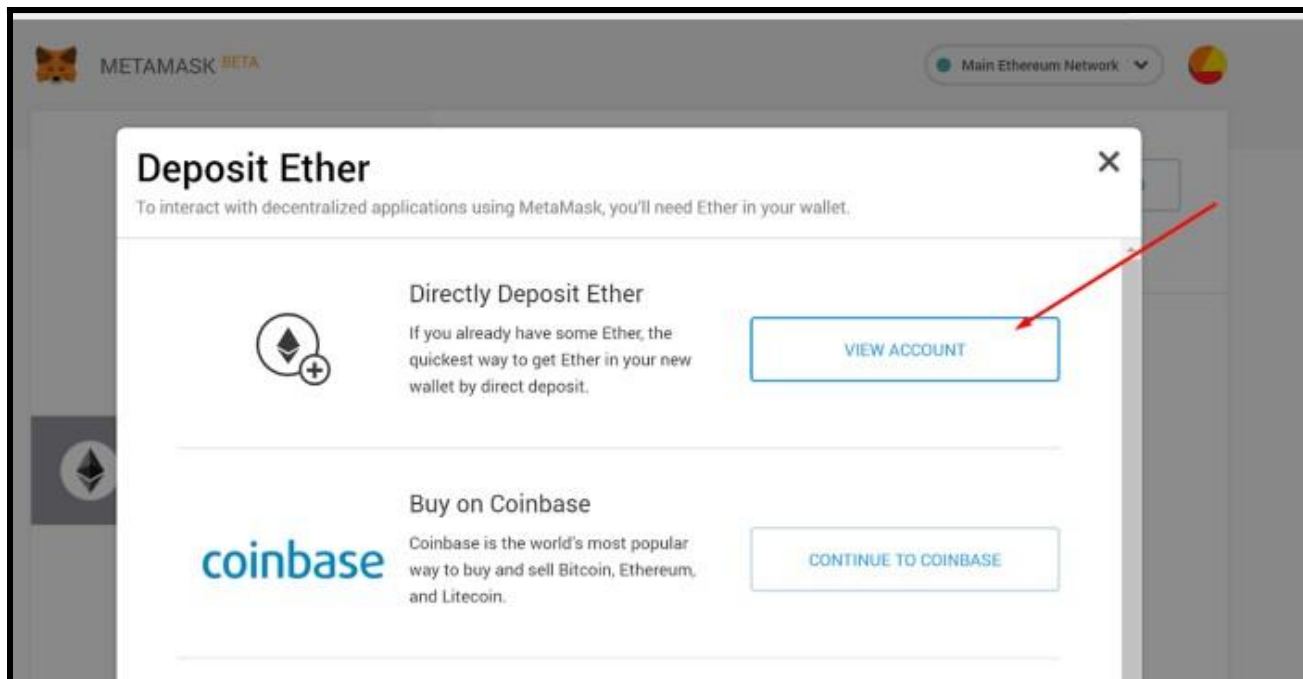
- Verify your secret phrase by selecting the previously generated phrase in order. **Click Confirm.**

And that's it; now you have created your MetaMask account successfully. A new Ethereum wallet

address has just been created for you. It's waiting for you to deposit funds, and if you want to learn how to do that, look at the next step below.

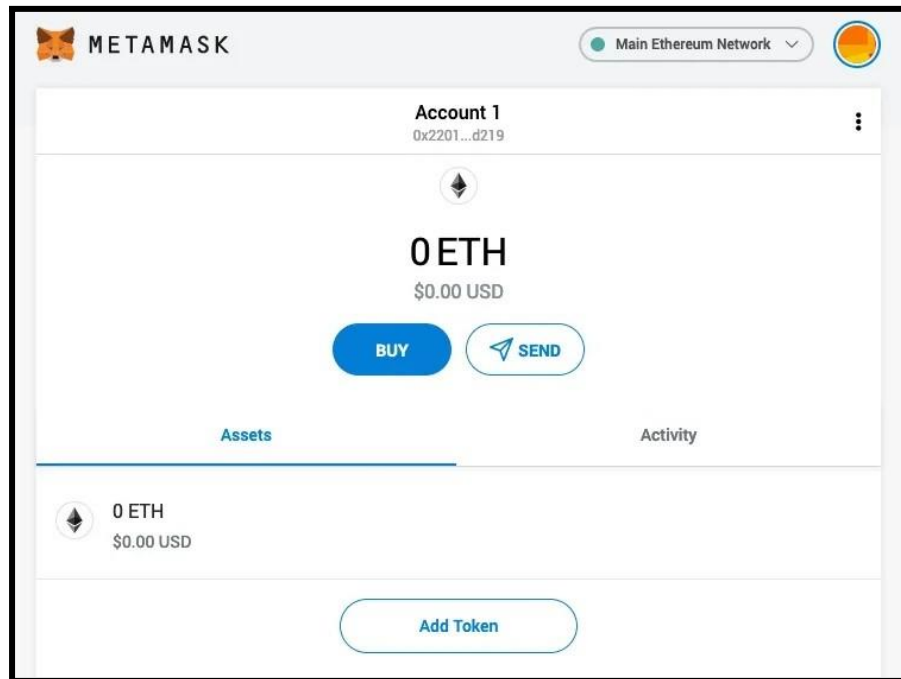
Step 3. Depositing funds.

- Click on **View Account**.



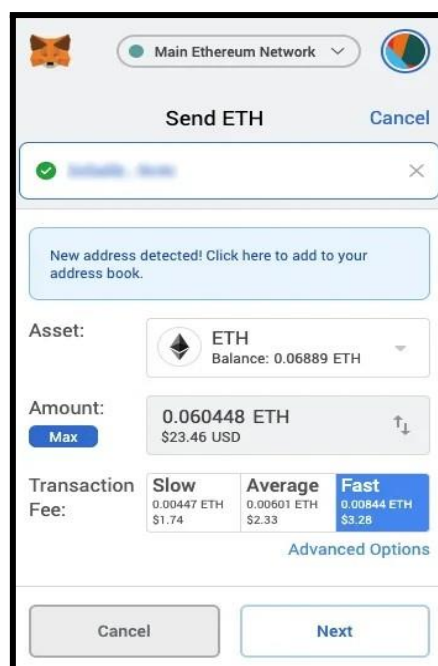
You can now see your public address and share it with other people. There are some methods to buy coins offered by MetaMask, but you can do it differently as well; you just need your address.

If you ever get logged out, you'll be able to log back in again by clicking the MetaMask icon, which will have been added to your web browser (usually found next to the URL bar).



You can now access your list of assets in the 'Assets' tab and view your transaction history in the 'Activity' tab.

- Sending crypto is as simple as clicking the 'Send' button, entering the recipient address and amount to send, and selecting a transaction fee. You can also manually adjust the transaction fee using the 'Advanced Options' button, using information from ETH Gas Station or similar platforms to choose a more acceptable gas price.
- After clicking 'Next', you will then be able to either confirm or reject the transaction on the subsequent page.



- To use MetaMask to interact with a dapp or [smart contract](#), you'll usually need to find a 'Connect to Wallet' button or similar element on the platform you are trying to use. After clicking this, you should then see a prompt asking whether you want to let the dapp connect to your wallet.

What advantages does MetaMask have?

- **Popular** - It is commonly used, so users only need one plugin to access a wide range of dapps.
- **Simple** - Instead of managing private keys, users just need to remember a list of words, and transactions are signed on their behalf.
- **Saves space** - Users don't have to download the Ethereum blockchain, as MetaMask sends requests to nodes outside of the user's computer.
- **Integrated** - Dapps are designed to work with MetaMask, so it becomes much easier to send Ether in and out.

Conclusion- In this way we have explored Concept Blockchain and metamask wallet for transaction of digital currency

Assignment Question

1. What Are the Different Types of Blockchain Technology?
2. What Are the Key Features/Properties of Blockchain?
3. What Type of Records You Can Keep in A Blockchain?
4. What is the difference between Ethereum and Bitcoin?
5. What are Merkle Trees? Explain their concept.
6. What is Double Spending in transaction operation
7. Give real-life use cases of blockchain.

Reference link

- <https://hackernoon.com/blockchain-technology-explained-introduction-meaning-and-applications-edbd6759a2b2>
- <https://levelup.gitconnected.com/how-to-use-metamask-a-step-by-step-guide-f380a3943fb1>
- <https://decrypt.co/resources/metamask>

Group C

Assignment No: 2

Title of the Assignment: Create your own wallet using Metamask for crypto transactions

Objective of the Assignment: Students should be able to learn about cryptocurrencies and learn how transaction done by using different digital currency

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain
-

Contents for Theory:

1. Cryptocurrency
 2. Transaction Wallets
 3. Ether transaction
-

Introduction to Cryptocurrency

- Cryptocurrency is a digital payment system that doesn't rely on banks to verify transactions. It's a peer-to-peer system that can enable anyone anywhere to send and receive payments. Instead of being physical money carried around and exchanged in the real world, cryptocurrency payments exist purely as digital entries to an online database describing specific transactions. When you transfer cryptocurrency funds, the transactions are recorded in a public ledger. Cryptocurrency is stored in digital wallets.
- Cryptocurrency received its name because it uses encryption to verify transactions. This means advanced coding is involved in storing and transmitting cryptocurrency data between wallets and to public ledgers. The aim of encryption is to provide security and safety.
- The first cryptocurrency was Bitcoin, which was founded in 2009 and remains the best known today. Much of the interest in cryptocurrencies is to trade for profit, with speculators at times driving prices skyward.

How does cryptocurrency work?

- Cryptocurrencies run on a distributed public ledger called blockchain, a record of all transactions updated and held by currency holders.
- Units of cryptocurrency are created through a process called mining, which involves using computer power to solve complicated mathematical problems that generate coins. Users can also buy the currencies from brokers, then store and spend them using cryptographic wallets.
- If you own cryptocurrency, you don't own anything tangible. What you own is a key that allows you to move a record or a unit of measure from one person to another without a trusted third party.
- Although Bitcoin has been around since 2009, cryptocurrencies and applications of blockchain technology are still emerging in financial terms, and more uses are expected in the future. Transactions including bonds, stocks, and other financial assets could eventually be traded using the technology.

Cryptocurrency examples

There are thousands of cryptocurrencies. Some of the best known include:

- **Bitcoin:**

Founded in 2009, Bitcoin was the first cryptocurrency and is still the most commonly traded. The currency was developed by Satoshi Nakamoto – widely believed to be a pseudonym for an individual or group of people whose precise identity remains unknown.

- **Ethereum:**

Developed in 2015, Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum. It is the most popular cryptocurrency after Bitcoin.

- **Litecoin:**

This currency is most similar to bitcoin but has moved more quickly to develop new innovations, including faster payments and processes to allow more transactions.

- **Ripple:**

Ripple is a distributed ledger system that was founded in 2012. Ripple can be used to track different kinds of transactions, not just cryptocurrency. The company behind it has worked with various banks and financial institutions.

- Non-Bitcoin cryptocurrencies are collectively known as “altcoins” to distinguish them from the original.

How to store cryptocurrency

- Once you have purchased cryptocurrency, you need to store it safely to protect it from hacks or theft. Usually, cryptocurrency is stored in crypto wallets, which are physical devices or online software used to store the private keys to your cryptocurrencies securely. Some exchanges provide wallet services, making it easy for you to store directly through the platform. However, not all exchanges or brokers automatically provide wallet services for you.
- There are different wallet providers to choose from. The terms “hot wallet” and “cold wallet” are used:
- **Hot wallet storage:** "hot wallets" refer to crypto storage that uses online software to protect the private keys to your assets.
- **Cold wallet storage:** Unlike hot wallets, cold wallets (also known as hardware wallets) rely on offline electronic devices to securely store your private keys.

Conclusion- In this way we have explored Concept Cryptocurrency and learn how transactions are done using digital currency

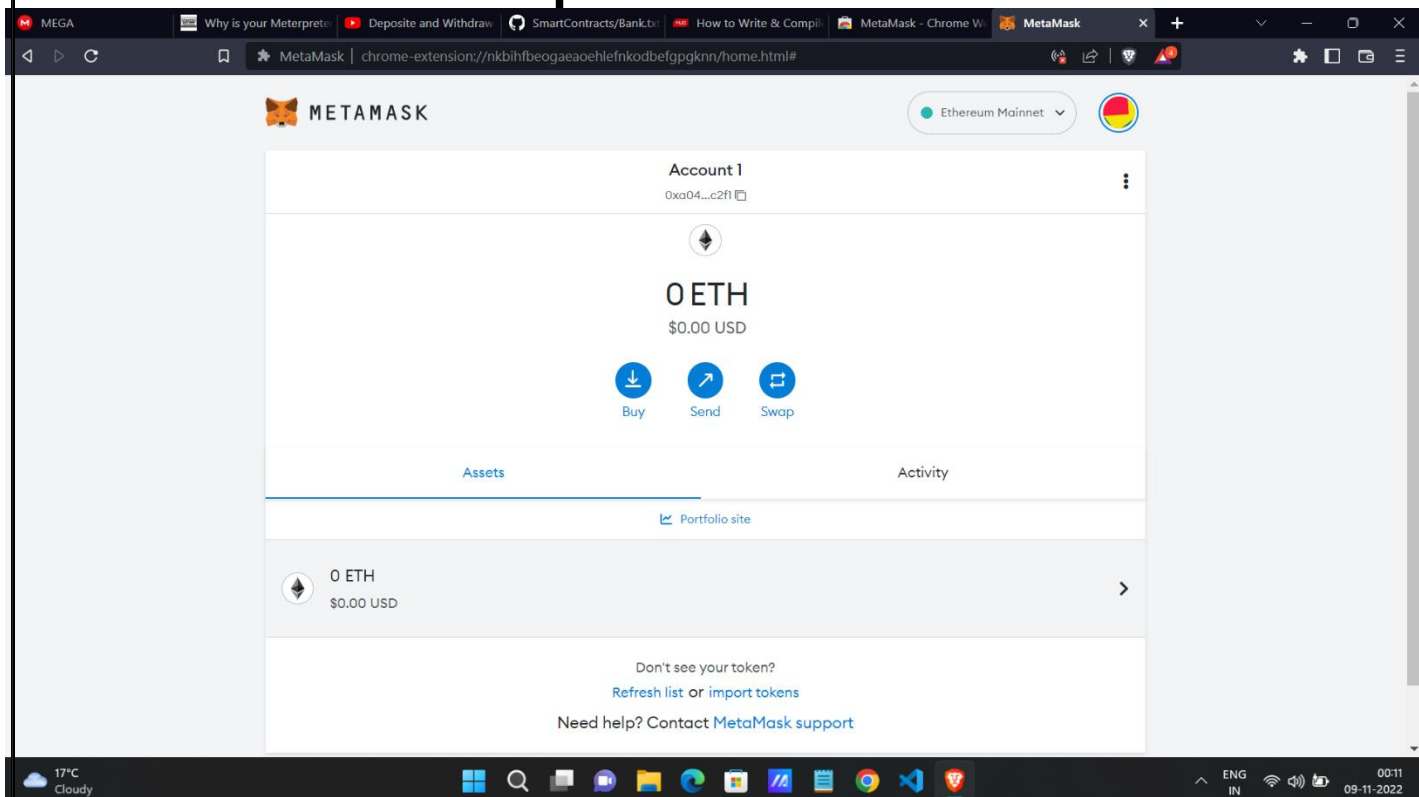
Assignment Question

1. What is Bitcoin?
2. What Are the biggest Four common cryptocurrency scams
3. Explain How safe are money e-transfers?
4. What is cryptojacking and how does it work?

Reference link

- <https://www.kaspersky.com/resource-center/definitions/what-is-cryptocurrency>

Output



Group C

Assignment No: 3

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

- Deposit money
- Withdraw Money
- Show balance

Objective of the Assignment: Students should be able to learn new technology such as metamask. Its application and implementations

Prerequisite:

1. Basic knowledge of cryptocurrency
 2. Basic knowledge of distributed computing concept
 3. Working of blockchain.
-

Contents for Theory:

The contract will allow deposits from any account, and can be trusted to allow withdrawals only by accounts that have sufficient funds to cover the requested withdrawal.

This post assumes that you are comfortable with the ether-handling concepts introduced in our post, [Writing a Contract That Handles Ether](#).

That post demonstrated how to restrict ether withdrawals to an “owner’s” account. It did this by persistently storing the owner account’s address, and then comparing it to the msg.sender value for any withdrawal attempt. Here’s a slightly simplified version of that smart contract, which allows anybody to deposit money, but only allows the owner to make withdrawals:

pragma solidity ^0.4.19;

contract TipJar {

 address owner; *// current owner of the contract*

 function TipJar() public
 { owner =
 msg.sender;
 }

 function withdraw() public {
 require(owner ==
 msg.sender);
 msg.sender.transfer(address(this).balance);
 }

 function deposit(uint256 amount) public payable
 { require(msg.value == amount);
 }

 function getBalance() public view returns (uint256)


```

    {return address(this).balance;
  }
}

```

I am going to generalize this contract to keep track of ether deposits based on the account address of the depositor, and then only allow that same account to make withdrawals of that ether. To do this, we need a way keep track of account balances for each depositing account—a mapping from accounts to balances. Fortunately, Solidity provides a ready-made mapping data type that can map account addresses to integers,

which will make this bookkeeping job quite simple. (This mapping structure is much more general key/value mapping than just addresses to integers, but that's all we need here.)

Here's the code to accept deposits and track account balances:

pragma solidity

```

^0.4.19;contract Bank {

    mapping(address => uint256) public balanceOf; //balances, indexed by addresses

    function deposit(uint256 amount) public payable
        {require(msg.value == amount);

        balanceOf[msg.sender] += amount;    // adjust the account's balance
    }
}

```

Here are the new concepts in the code above:

- mapping(address => uint256) public balanceOf; declares a persistent public variable, balanceOf, that is a mapping from account addresses to 256-bit unsigned integers. Those integers will represent the current balance of ether stored by the contract on behalf of the corresponding address.
- Mappings can be indexed just like arrays/lists/dictionaries/tables in most modern programming languages.
- The value of a missing mapping value is 0. Therefore, we can trust that the beginning balance for all account addresses will effectively be zero prior to the first deposit.

It's important to note that balanceOf keeps track of the ether balances assigned to each account, but it does not actually move any ether anywhere. The bank contract's ether balance is the sum of all the balances of all accounts—only balanceOf tracks how much of that is assigned to each account.

Note also that this contract doesn't need a constructor. There is no persistent state to initialize other than the balanceOf mapping, which already provides default values of 0.

Given the balanceOf mapping from account addresses to ether amounts, the remaining code for a fully-functional bank contract is pretty small. I'll simply add a withdrawal function:

bank.sol

```
pragma solidity ^0.4.19;

contract Bank {

    mapping(address => uint256) public balanceOf; // balances, indexed by addresses

    function deposit(uint256 amount) public payable {
        require(msg.value == amount);
        balanceOf[msg.sender] += amount;          // adjust the account's balance
    }

    function withdraw(uint256 amount) public {
        require(amount <= balanceOf[msg.sender]);
        balanceOf[msg.sender] -= amount;
        msg.sender.transfer(amount);
    }
}
```

The code above demonstrates the following:

- The `require(amount <= balanceOf[msg.sender])` checks to make sure the sender has sufficient funds to cover the requested withdrawal. If not, then the transaction aborts without making any state changes or ether transfers.
- The `balanceOf` mapping must be updated to reflect the lowered residual amount after the withdrawal.
- The funds must be sent to the sender requesting the withdrawal.

In the `withdraw()` function above, it is very important to adjust `balanceOf[msg.sender]` **before** transferring ether to avoid an exploitable vulnerability. The reason is specific to smart contracts and the fact that a transfer to a smart contract executes code in that smart contract. (The essentials of Ethereum transactions are discussed in [How Ethereum Transactions Work](#).)

Now, suppose that the code in `withdraw()` did not adjust `balanceOf[msg.sender]` before making the transfer *and* suppose that `msg.sender` was a malicious smart contract. Upon receiving the transfer—handled by `msg.sender`'s fallback function—that malicious contract could initiate *another* withdrawal from the banking contract. When the banking contract handles this second withdrawal request, it would have already transferred ether for the original withdrawal, but it would not have an updated balance, so it would allow this second withdrawal!

This vulnerability is called a “reentrancy” bug because it happens when a smart contract invokes code in a different smart contract that then calls back into the original, thereby reentering the exploitable contract. For this reason, it's essential to always make sure a contract's internal state is fully updated before it potentially invokes code in another smart contract. (And, it's essential to remember that every transfer to a smart contract executes that contract's code.)

To avoid this sort of reentrancy bug, follow the “Checks-Effects-Interactions pattern” as [described in the Solidity documentation](#). The `withdraw()` function above is an example of implementing this pattern

Program :

```
pragma solidity ^0.6.0;
contract MyBank

mapping(address=> uint ) private _balances;
address public owner;
```

```
event LogDepositMade(address accountHoder, uint amount );
constructor () public
{
    owner=msg.sender;
    emit LogDepositMade(msg.sender, 1000);
}
function deposit() public payable returns (uint)
{
    require ((_balances[msg.sender] + msg.value) > _balances[msg.sender] && msg.sender!=address(0));
    _balances[msg.sender] += msg.value;
    emit LogDepositMade(msg.sender , msg.value);
    return _balances[msg.sender];
}
function withdraw (uint withdrawAmount) public returns (uint)
{
    require (_balances[msg.sender] >= withdrawAmount);
    require(msg.sender!=address(0));
    require (_balances[msg.sender] > 0);
    _balances[msg.sender]-= withdrawAmount;
    msg.sender.transfer(withdrawAmount);
    emit LogDepositMade(msg.sender , withdrawAmount);
    return _balances[msg.sender];
}
function viewBalance() public view returns (uint)
{
    return _balances[msg.sender];
}
```

Group C

Assignment No: 4

Title: survey report on types of Blockchains and its real time use cases.

Problem Statement: Write and survey report on blockchain technology

Prerequisites: Blockchain Technology

Objectives: The objective is to explore deep into blockchain concepts, types, its real time use cases. Survey aggregates all the core concepts of blockchain technologies for future researchers and readers who are initiating their studies in the particular technology.

Outcomes: To understand the of blockchain concepts and implementation of realtime use cases.

Theory:

Blockchain Concepts:

Blockchain can be defined as an immutable distributed digital ledger, which is secured using advanced cryptography, replicated among the peer nodes in the peer-to-peer network, and uses consensus mechanism to agree upon the transaction log, whereas control is decentralized. With this definition, paper identifies following concepts as the core concepts to unwrap the meaning of blockchain—immutable, distributed, digital ledger, cryptography, peer-to-peer network, consensus mechanism, decentralization. In accounting, a ledger is a place to record and store all the transactions with regard to an entity. A digital ledger could be a computer file, or database, or even distributed database like blockchain, where transactions are recorded electronically. Blockchain transaction ledger is pretty unique to other ledgers in a manner, which ensures that transaction log is computationally impractical to change, as long as honest nodes in the network control the majority of CPU power, thus making it immutable. The origins of ledger can be traced back to over 5000 years ago in Mesopotamia. The Earliest and simplest form of recording transactions is called single entry accounting, which enters transactions into a list to keep track of adding or deducting assets. The single entry accounting was managed by owners or family members, as this kind of recordings are error-prone as well as difficult to track down, when recorded fraudulently. Double entry accounting added a clear strategy to identify and remove errors, where there are two entries recorded against each transaction, so that the ledger is balanced all the time. Grigg proposed triple entry accounting in 2005, an alternative to traditional double entry accounting, which secures transactions using cryptography in order to make it difficult to change. Blockchain implements triple entry accounting concept to permanently store transactions in blockchain, ensuring that the sender has authority to execute non-reversible transactions using public-key cryptography.

Cryptography can be defined as techniques used for secure communication to protect confidential information, in the presence of adversaries. Blockchain uses concepts from public key cryptosystems to verify the authority of the user to execute transactions, and cryptographic hash functions to achieve consensus between network nodes on blockchain data. The use of public key cryptosystems to provide digital signatures was suggested by Diffie and Hellman. Digital signatures, whether based on public key cryptosystems, conventional encryption functions, on probabilistic computations, or other techniques share several important properties in common—such as an easier way for the sender to generate the personal digital signature, convenient way for receiver to verify the sender of the message, but must be impossible to generate someone else's digital signature by others. In public key cryptography, there exists two keys called public and private and a function or cypher algorithm to encrypt the original text into a ciphertext using the private encryption key. Sender or owner generates the public-private key pair and keeps the private key as the confidential key to encrypt information; public key is distributed to anyone to verify that the information is digitally signed by the original owner. This public key cryptography technique is used in blockchain to verify the ownership of coins or tokens, whenever transferring coins or tokens. One another important concept used in blockchain to secure its data integrity is cryptographic hash function—a one-way function that maps strings of arbitrary size into a bit string of fixed size called hash using a mathematical algorithm. An algorithm required for blockchain hash functions has three main properties—same input should always result in with the same output hash, given the hash no algorithm could produce the original input, small changes in input results in completely different output hash. Bitcoin uses SHA-256 hash function, whereas Ethereum uses Ethash, and Litecoin uses Script when hashing its block data.

Blockchain Types

According to our survey findings, blockchains can be categorized into two main types namely **permissionless blockchains** and **permissioned blockchains**.

Permissionless Blockchains

Permissionless blockchains do not enforce any restrictions on its nodes; anyone can openly read data, inspect data, and participate in validation and writing of the data in accordance with the consensus protocol of the particular blockchain. Bitcoin, Ethereum and many other cryptocurrencies run on permissionless blockchains. These blockchains are considered fully decentralized and secured using advanced cryptography, whereas economic incentives are provided for users who work to keep the integrity of the network. The transactions are completely irreversible on a permissionless blockchain by its design, meaning once confirmed by its nodes the blockchain transactions cannot be reversed. Due to the security considerations and strict restrictions, transaction throughput of a permissionless blockchain is comparatively lesser

than one of a permissioned blockchain. Permissionless blockchains are fully decentralized and transparent.

Permissioned Blockchains:

Permissioned blockchains restrict the writing access for a limited set of participants, and a consensus mechanism is used to validate the writing of data among its privileged participants. Read access could either be open to anyone or closed to the public based on the requirement of the permissioned blockchain. This type of blockchains has evolved as an alternative to initial permissionless blockchains, to address the requirement for running blockchain technology among a set of known and identifiable participants that have to be explicitly responsible to the blockchain network, while participants need not be fully trusting each other. The permissioned blockchains are mainly useful for business and social applications, which requires blockchain distributed ledger technology without the need of a in centifying cryptocurrency. Based on the read access mentioned, permissioned blockchains are further divided as open and closed—open permissioned blockchains are partially decentralized, anyone can read its data, whereas closed permissioned blockchains are fully centralized, data is visible only to the participants.

We thoroughly believe blockchain technology is rather necessary only for permissionless blockchains, and open permissioned blockchains. Closed permissioned blockchains can be argued as restricted distributed databases which are facelifted with the blockchain term. The initial idea of introducing blockchain concept was to remove centralization and add transparency to everyone to read and update its data. Open permissioned blockchains mostly adhere to this principle of transparency even though somewhat centralized in writing its data and could be useful for applications such as identity systems, academic certification systems, where anyone can read its data but only a certain set of participants are privileged to write the data into blockchain. Closed permissioned blockchains are fully centralized and also not transparent to anyone, dismantling the core concept of a blockchain. Therefore, these blockchains can be replaced with distributed database systems with restrictions implemented on top of it. For example, a supply chain management system for a private organization can be implemented without the concepts of blockchain. In order to support our argument on closed permissioned blockchains, we have presented a characteristic comparison of different blockchain types compared with restricted distributed database system. The comparison shows that all of the characteristics in closed permissioned blockchains are comparatively similar to that of restricted database systems. In addition to this categorization, there is also another blockchain categorization called public, consortium, and private blockchains In simple terms, public blockchains are permissionless blockchains, whereas consortium and private blockchains fall into permissioned blockchains

Real Time blockchain use cases

Blockchain technology's core characteristics include decentralization, transparency, immutability, and automation. These elements can be applied to various industries, creating a multitude of use cases. Here are what we believe to be the most pertinent blockchain use cases for enterprises, institutions, and governments.

Capital Markets

For capital markets, blockchain unlocks easier, cheaper, and faster access to capital. It reduces the barriers to issuance and enables peer-to-peer trading, faster and more transparent settlement and clearing, reduced costs, decreased counterparty risks, and streamlined auditing and compliance.

Central Bank Digital Currencies CBDC

CBDCs are a digital form of central bank money that offers central banks unique advantages at the retail and wholesale levels, including increased financial access for individual customers and a more efficient infrastructure for interbank settlements.

Decentralized Finance (DeFi)

Decentralized finance—DeFi—refers to the shift from traditional, centralized financial systems to peer-to-peer finance enabled by decentralized technologies built on Ethereum. Millions are building and participating in this new economic system that is setting new standards for financial access, opportunity, and trust.

Digital Identity

A blockchain-based digital identity system provides a unified, interoperable, and tamper-proof infrastructure with key benefits to enterprises, users, and IoT management systems. The solution protects against theft and provides individuals greater sovereignty over their data.

Finance

Financial services struggle with archaic operational processes, slow payment settlements, limited transparency, and security vulnerabilities. Blockchain enhances the efficient digitization of financial instruments, which increases liquidity, lowers cost of capital, and reduces counterparty risk.

Energy and Sustainability

Oil and gas companies suffer from siloed infrastructures and a lack of transparency, efficiency, and optimization. Enterprise-grade blockchain solutions can significantly increase process efficiencies and reduce costs associated with oil and gas operations and distribution.

Global Trade and Commerce

Major trading companies and consortiums are recognizing the transformative impact of blockchain in operating global supply chains, managing trade finance, and unlocking new business models. ConsenSys' blockchain products offer secure digitization, enabling the tokenization of existing documents, letters of credit, and more.

Government and the Public Sector

Ethereum blockchain technology allows governments to build trust, improve accountability and responsiveness, increase efficiency, reduce costs, and create high-performing government functions with more secure, agile, and cost-effective structures.

Healthcare and the Life Sciences

Blockchain-based healthcare solutions will enable faster, more efficient, and more secure medical data management and medical supply tracking. This could significantly improve patient care, facilitate the advancement to medical discoveries, and ensure the authenticity of drugs circulating global markets.

Real Estate

Enterprise Ethereum enables the digitization of assets and financial instruments. This enhances fractionalization of ownership, expanded access to global markets, increased liquidity, and democratized access to real estate investment opportunities.

Supply Chain Management

Existing global supply chains are inefficient, poorly tracked, and oftentimes exploitative. Blockchain can facilitate accurate asset tracking, enhanced licensing of services, products, and software, and transparency into the provenance of consumer goods, from sourcing to the point of consumption.

Conclusion: Hence, we have successfully studied and survey report on Blockchain technology

Group C

Assignment No: 5

Title: Write a program in solidity to create Student data. and Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values

Problem Statement: create Student data and follow the Structures Arrays Fallback Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Prerequisites: Blockchain Technology

Objectives: Implement solidity programming and smart contract on Ethereum and observe the transaction fee and Gas values

Outcomes: To learn the concept of smart contract on Ethereum and observe the transaction fee and gas values.

Theory:

Ethereum:

Ethereum is a decentralized blockchain designed to be highly secure, fault-tolerant, and programmable. Ethereum blockchain is a choice for many developers and businesses. As said programmable, the main task of Ethereum is to securely execute and verify the application code known as smart contracts. Ethereum helps to build native scripting language (solidity) and EVM. **Overview of Smart Contracts:**

A smart contract is a small program that runs on an Ethereum blockchain. Once the smart contract is deployed on the Ethereum blockchain, it cannot be changed. To deploy the smart contract to Ethereum, you must pay the ether (ETH) cost. Understand it as a digital agreement that builds trust and allows both parties to agree on a particular set of conditions that cannot be tampered with.

To understand the need for a smart contract, suppose there was one grocery shop, and Ram went to buy some groceries. He purchased the groceries for 500 rupees and kept on debt that would pay the money next month when he returned, so the shopkeeper jotted down his purchase in his ledger. In between the period somehow shopkeeper changed 500 to 600 and when next month Ram went to pay the money, the shopkeeper has demanded 600 INR and Ram has no proof to show that he has only bought 500 INR so in this case, smart contracts play an essential role which prevents both the parties to tamper the agreement and only gets terminated when all the conditions satisfy after the deal.

Ethereum Blockchain Platform executes Smart Contracts:

Ethereum Virtual Machine (EVM)

The purpose of EVM is to serve as a runtime environment for smart contracts built on Ethereum. Consider it as a global supercomputer that executes all the smart contracts.

As the name indicates, Ethereum Virtual Machine is not physical but a virtual machine. The functionality of EVM is restricted to virtual machines; for example, it cannot make delayed calls on the internet or produce random numbers. Therefore, it is considered a simple state machine. Writing programs in assembly language do not make any sense, so, Ethereum required a programming language for the EVM.

Gas

In the Ethereum Virtual Machine, gas is a measurement unit used for assigning fees to each transaction with a smart contract. Each computation happening in the EVM needs some amount of gas. The more complex the computation is, the more the gas is required to run the smart contracts. **Transaction fee = Total gas used*gas price**

Solidity:

Solidity is a smart contract programming language on Ethereum. Developed on the top of the EVM, it is similar to the object-oriented programming language that uses class and methods. It allows you to perform arbitrary computations, but it is used to send and receive tokens and store states. When it comes to syntax, Solidity is greatly influenced by C++, Python, and Javascript so that developers can understand its syntax quickly.

Solidity Programming

Solidity is object-oriented, high-level statically-typed programming language used to create smart contracts. Solidity programming looks similar to Javascript, but there are a lot of differences between both languages. In solidity, you need to compile the program first, while in Javascript, you can run the program directly in your browser or by using Node JS. Solidity is a Javascript-like language developed specifically for creating smart contracts. It is typed statically and supports libraries, inheritance and complex user-defined types. Solidity compiler converts code into EVM bytecode which is sent to the Ethereum network as a deployment transaction.

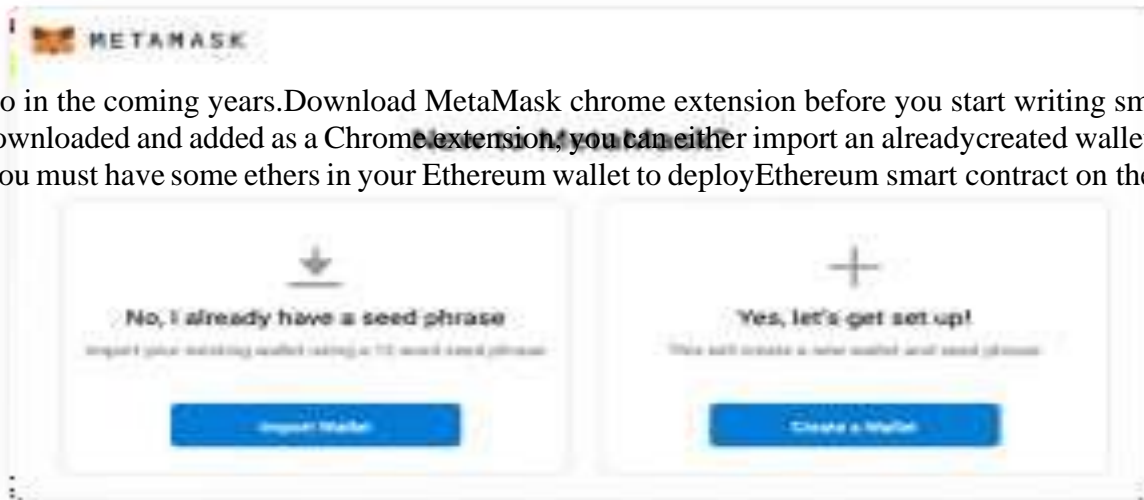
Here's a step-by-step guide to creating and deploying Ethereum Smart Contracts with Solidity

Installing Prerequisites

Meta-mask Chrome Extension

MetaMask acts both as an Ethereum browser and a wallet. It allows you to interact with smart contracts and dApps on the web without downloading the blockchain or installing any software. You only need to add MetaMask as a Chrome Extension, create a wallet and submit Ether. Though MetaMask is currently available for Google Chrome browser, it is expected to launch for Firefox

too in the coming years. Download MetaMask chrome extension before you start writing smart contracts. Once it is downloaded and added as a Chrome extension, you can either import an already created wallet or create a new wallet. You must have some ethers in your Ethereum wallet to deploy Ethereum smart contract on the network.



Steps to develop an Ethereum Smart Contract

Step 1: Create a wallet at meta-mask

Install MetaMask in your Chrome browser and enable it. Once it is installed, click on its icon on the top right of the browser page. Clicking on it will open it in a new tab of the browser. Click on “Create Wallet” and agree to the terms and conditions by clicking “I agree” to proceed further. It will ask you to create a password. After you create a password, it will send you a secret backup phrase used for backing up and restoring the account. Do not disclose it or share it with someone, as this phrase can take away your Ethers.



The next step is to ensure that you are in the “Main Ethereum Network.” If you find a checkmark next to “Main Ethereum Network”, you are in the right place.

Step 2: Select any one test network

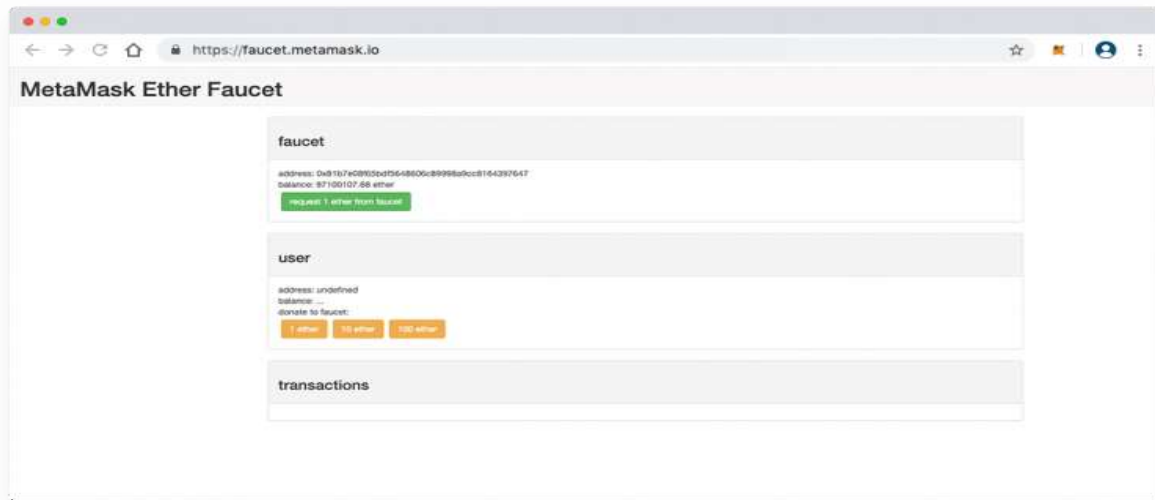
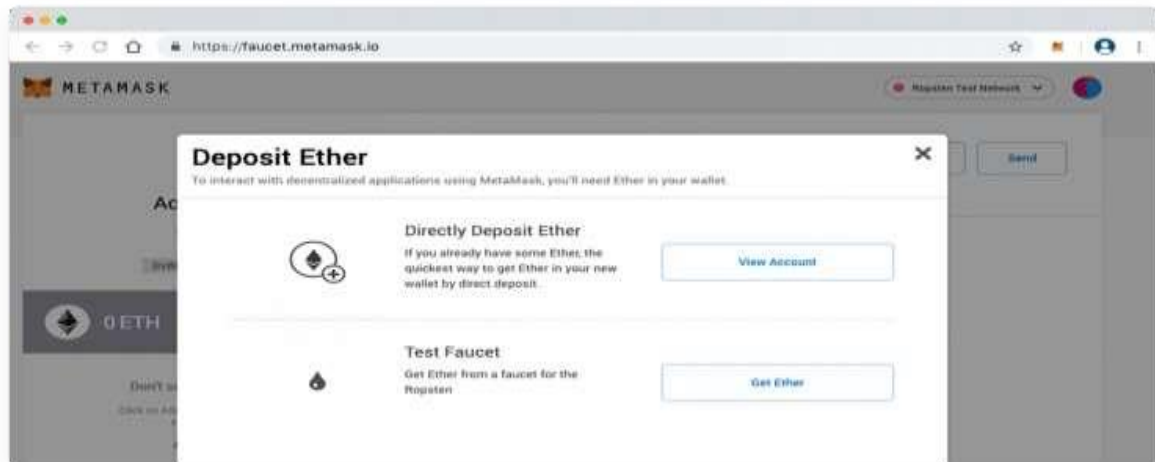
You might also find the following test networks in your MetaMask wallet:

- Robsten Test Network
- Kovan Test Network
- Rinkeby Test Network
- Goerli Test Network

- The above networks are for testing purposes only; note that these networks’ ethers have no real value

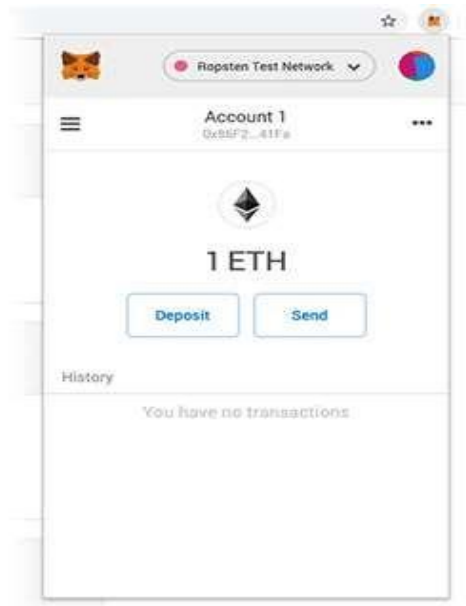
Step 3: Add some dummy Ethers to your wallet

In case you want to test the smart contract, you must have some dummy ethers in your MetaMaskwallet.



For example, if you want to test a contract using the Robsten test network, select it and you will find 0 ETH as the initial balance in your account. To add dummy ethers, click on the “Deposit” and “Get Ether” buttons under Test Faucet. To proceed, you need to click “request one ether fromthe faucet,” and 1 ETH will be added to your wallet. You can add as many Ethers you want to thetest network.

For example, I have added 1 ETH in this scenario.



Once the dummy ethers are added to the wallet, you can start writing smart contracts on the RemixBrowser IDE in the Solidity programming language.

Step 4: Use editor remix to write the smart contract in Solidity

We will use Remix Browser IDE to write our Solidity code. The remix is the best option for writingsmart contracts

as it comes with a handful of features and offers a comprehensive development experience.

Step 5: Create a .sol extension file

Open Remix Browser and click on the plus icon on the top left side, next to the browser to create a .sol extension file.

Step 6: A sample Code

Arrays in Solidity

The array is a special data structure used to create a list of similar type values. The array can be offixed size and dynamic-sized. With the help of index elements can be accessed easily. below is a sample code to create, and access a fixed-sized array in solidity.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Array {
    uint [4] public arr = [10, 20, 30, 40];
    function setter(uint index, uint value) public {
        arr[index] = value;
    }

    function length() public view returns(uint) {return
        arr.length;
    }
}
```

You can compile and deploy the code to try changing the array elements with an index and printing the array length.

Creating Dynamic Array

A dynamic array is an array where we can insert any number of elements and delete the details easily using an index. So solidity has functions like push and pops like python, making it easy to create a dynamic array. Below is a code using which you can create a dynamic array. After writing code, compiles and deploy the code by visiting the deploy section in the left-side navigation bar. After that, try inserting and deleting some elements from an array.

```
pragma solidity >= 0.5.0 < 0.9.0;
contract Array {
    uint [] public arr;
    function PushElement(uint item) public {
        arr.push(item);
    }
    function Length() public view returns(uint) {
        return arr.length;
    }
    function PopElement() public {
        arr.pop();
    }
}
```

Structure in Solidity

The structure is a user-defined data type that stores more than one data member of different data types. As in array, we can only store elements of the same data type, but in structure, you can keep elements of different data types used to create multiple collections. The structure can be made outside and inside the contract storage, and the Structure keyword can be used to declare the form. The structure is storage type, meaning we use it in-store only, and if we want to use it in function, then we need to use the memory keyword as we do in the case of a string.

```
pragma solidity >= 0.5.0 < 0.9.0;
```

```
struct Student {  
    uint rollNo;  
    string name;  
}
```



```
contract Demo {
    Student public s1;
    constructor(uint _rollNo, string memory _name) {
        s1.rollNo = _rollNo;
        s1.name = _name;
    }
    // to change the value we have to implement a setter function
    function changeValue(uint _rollNo, string memory _name) public {
        Student memory new_student = Student( {
            rollNo : _rollNo,
            name : _name
        });
        s1 = new_student;
    }
}
```

Fallback:

```
pragma solidity ^0.4.0;
// Creating a contract
contract fback
{
    // Declaring the state variableuint
    x;
    // Mapping of addresses to their balances
    mapping(address => uint) balance;
    // Creating a constructor
    constructor() public
    {
        // Set x to default
        // value of 10
        x=10;
    }
    // Creating a function
    function SetX(uint _x) public returns(bool)
    {
        // Set x to the
        // value sent
        x=_x;
    }
}
```

```
        return true;
    }

    // This fallback function
    // will keep all the Ether
    function() public payable
    {
        balance[msg.sender] += msg.value;
    }
}

// Creating the sender contract
contract Sender
{
    function transfer() public payable
    {
        // Address of Fback contract
        address _receiver =
            0xabcD310867F1b74142c2f5776404b6bd97165FA56;

        // Transfers 100 Eth to above contract
        _receiver.transfer(100);
    }
}
```

Create a Smart Contract with CRUD Functionality

We have excellent theoretical and hands-on practical knowledge about solidity, and now you can create a primary smart contract like hello world, getter, and setter contracts. So it's a great time to try making some functional smart contracts, and the best way to try all the things in one code is to create one program that performs all CRUD operations.

A sample smart contract code to create ERC20 tokens

```
pragma solidity ^0.4.0; import
"./ERC20.sol"; contract myToken
is ERC20{
    mapping(address =>uint256) public amount;uint256
    totalAmount;
    string tokenName;
    string tokenSymbol;
    uint256 decimal;
```

```
constructor() public{ totalAmount =  
10000 * 10**18;  
amount[msg.sender]=totalAmount;  
tokenName="Mytoken";  
tokenSymbol="Mytoken"; decimal=18;  
}  
function totalSupply() public view returns(uint256){return  
totalAmount;  
}  
function balanceOf(address to_who) public view  
returns(uint256){  
return amount[to_who];  
}  
function transfer(address to_a,uint256 _value) public  
returns(bool){ require(_value<=amount[msg.sender]);  
amount[msg.sender]=amount[msg.sender]-_value;  
amount[to_a]=amount[to_a]+_value;  
return true;  
}  
}
```

Step 7: Deploy your contract

Deploy the smart contract at the Ethereum test network by pressing the deploy button at the Remix window's right-hand side. Wait until the transaction is complete.

After the transaction commits successfully, the address of the smart contract would be visible at the right-hand side of the remix window. At first, all the ERC20 tokens will be stored in the wallet of a user who is deploying the smart contract.

To check the tokens in your wallet, go to the metamask window, click add tokens, enter the smartcontract address and click ok. You would be able to see the number of tokens there.

Steps to deploy Ethereum Smart Contracts

- To make your smart contract live, switch to the main ethereum network at metamask
- Add some real ethers.
- Now again, deploy your smart contract using remix as mentioned in the above steps.
- When a smart contract is deployed successfully, visit <http://www.etherscan.io> and search your smart contract address there. Select your smart contract.
- Now you need to verify your smart contract here, click “verify the contract.”

- Copy your smart contract code and paste it at Etherscan. Select the same compiler version that you selected at remix to compile your code.
- Check “optimization” to Yes, if you had selected optimization at remix; otherwise, select No.
- Click Verify.
- It will take a few minutes and your smart contract will be live if no issue occurs.
- You can now run your smart contract methods at Etherscan.

Conclusion: Hence, we have successfully studied Solidity is an object-oriented high-level programming language for creating a smart contract that runs on the Ethereum blockchain. We have learned about the smart contract and its creation using solidity programming.