**Experiment – 1**

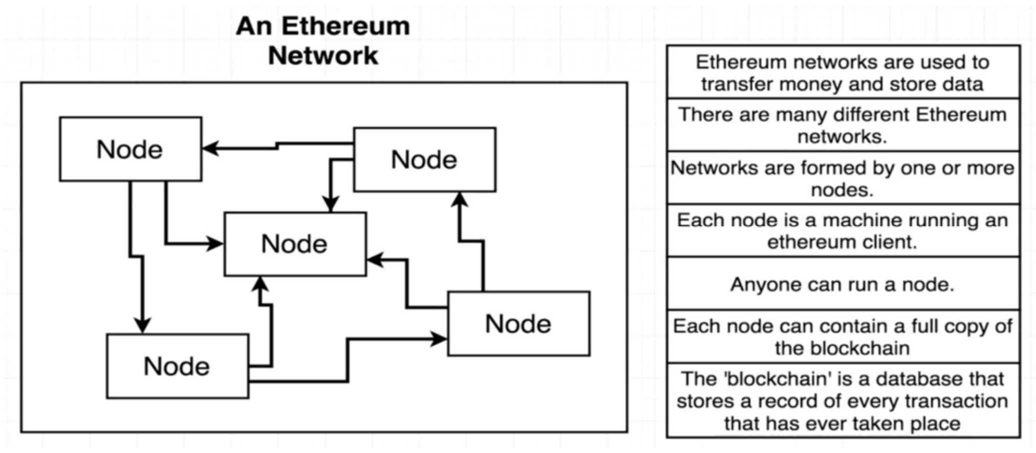
**AIM:** Introduction to Ethereum , Ethereum Nodes, Ethereum Accounts and Ethereum Addresses

## Introduction to Ethereum

Ethereum is a decentralized platform that supports smart contracts, programs that execute exactly as intended with no chance of fraud or outside influence. This blockchain may be customized. It enables users to start ICOs and develop decentralized applications (DApps). These applications are powered by a specially developed blockchain, a potent worldwide shared infrastructure that can transfer value and reflect property ownership.

This makes it possible for developers to build markets, keep records of obligations or promises, transfer money following directives left behind in the past [like a will or a futures contract) and do a lot of other things that are still in the future without the need for a middleman or counterparty risk. Ethereum protocol is powered by ETH, the native cryptocurrency of the Ethereum blockchain and it is an essential part of the wcb3 stack. The Ethereum protocol is Turing complete, meaning it can run any program.

## Ethereum Nodes


An Ethereum Network
Node
Node
Node
Node
Node
Ethereum networks are used to transfer money and store data
There are many different Ethereum networks.
Networks are formed by one or more nodes.
Each node is a machine running an ethereum client.
Anyone can run a node.
Each node can contain a full copy of the blockchain
The 'blockchain' is a database that stores a record of every transaction that has ever taken place

(i) **Full Node** - A full node is a computer that stores a copy of the entire Ethereum blockchain. full nodes help to keep the network secure by validating and propagating transactions and blocks. They also provide the necessary data for light clients to access the network.

(ii) **Light Client** -A light client is a computer that does not store a copy of the blockchain but instead relies on full nodes to provide data. Light clients can be used to send and receive transactions and to interact with smart contracts.

(iii) **Contract** - A contract is a program that runs on the Ethereum network and can store data and execute transactions. Contracts can be used to create decentralized applications or to interact with other contracts.

## Ethereum Accounts

An Ethereum account is an entity with an ether (ETH) balance that can send transactions on Ethereum. Accounts can be user-controlled or deployed as smart contracts.

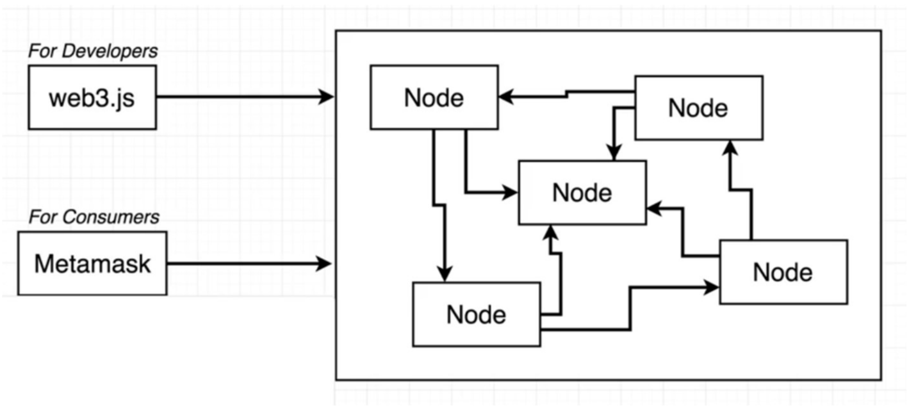
### Account types

Ethereum has two account types:

- Externally-owned account (EOA) – controlled by anyone with the private keys
- Contract account – a smart contract deployed to the network, controlled by code.

Both account types have the ability to:

- Receive, hold and send ETH and tokens
- Interact with deployed smart contracts



## Ethereum Addresses

An Ethereum address is a unique identifier for an account on the Ethereum blockchain. It is a 20-byte hexadecimal number that is made up of a string of alphanumeric characters and usually begins with "0x"

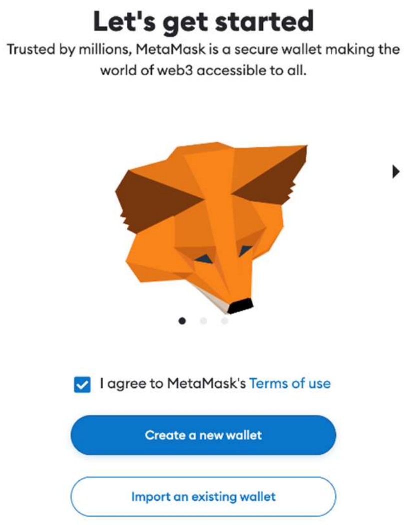
# Experiment – 2

**AIM:** Creating an Ethereum Account Using Meta Mask, Creating an Ethereum Account Using My Ether Wallet (MEW), Ether (ETH)

## Metamask Setup and Configuration

This lecture will provide instructions for setting up the Metamask extension in your browser.

1. First, install the metamask extension for your specific browser (Chrome or Firefox recommended)
2. After the extension has been installed, a modal should automatically pop up in your browser. Check the I Agree box and click the Create a new wallet button.

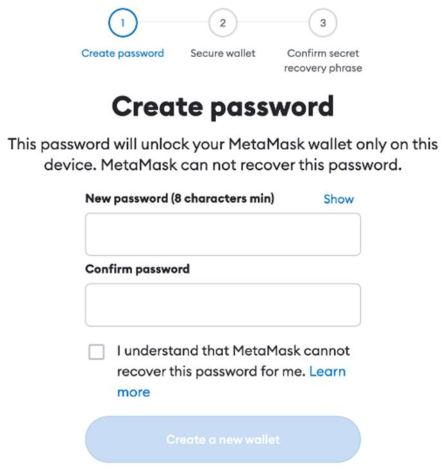


Let's get started

Trusted by millions, MetaMask is a secure wallet making the world of web3 accessible to all.

☑ I agree to MetaMask's Terms of use

3. Agree or Decline the Help us improve Metamask question.
4. Create a password for your Metamask wallet and then click the Create a new wallet button.



Create password

This password will unlock your MetaMask wallet only on this device. MetaMask can not recover this password.

New password (8 characters min) Show

Confirm password

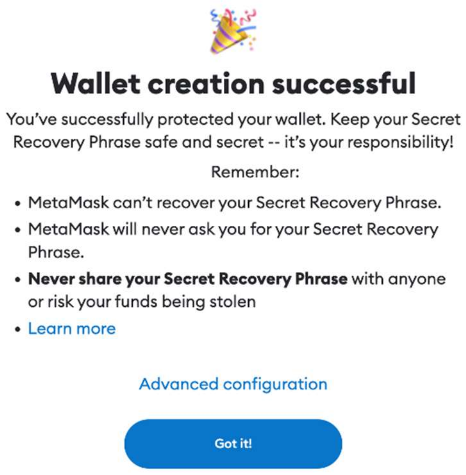
☐ I understand that MetaMask cannot recover this password for me. Learn more

5. Click the Secure my wallet button

6. Click the Reveal seed phrase button. Copy this 12-word mnemonic (recovery phrase) to a secure place. Then, click the Next button.

7. Use the phrase copied in the previous step to complete the Confirm Secret Recovery Phrase form. Click Confirm when finished.

8. You should get a success message. If so, click the Got it! button.



**Wallet creation successful**

You've successfully protected your wallet. Keep your Secret Recovery Phrase safe and secret -- it's your responsibility!

Remember:

- MetaMask can't recover your Secret Recovery Phrase.
- MetaMask will never ask you for your Secret Recovery Phrase.
- **Never share your Secret Recovery Phrase** with anyone or risk your funds being stolen
- Learn more

Advanced configuration

Got it!

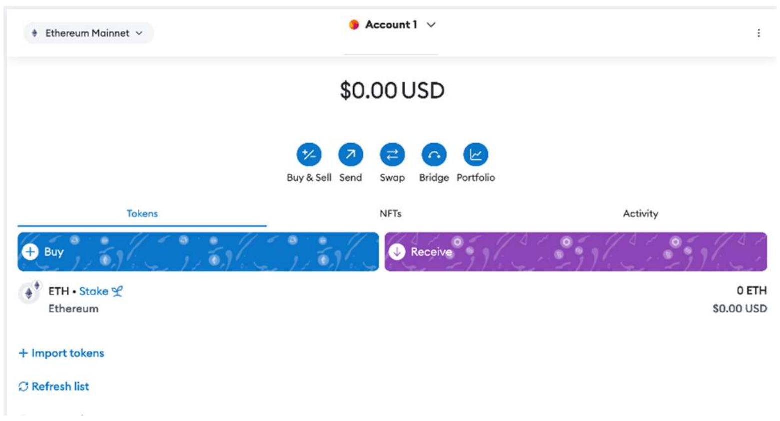
9. Click the Next button on the install is complete screen and then click Done.

10. Exit out of any advertisements and you should now be at the Metamask dashboard.

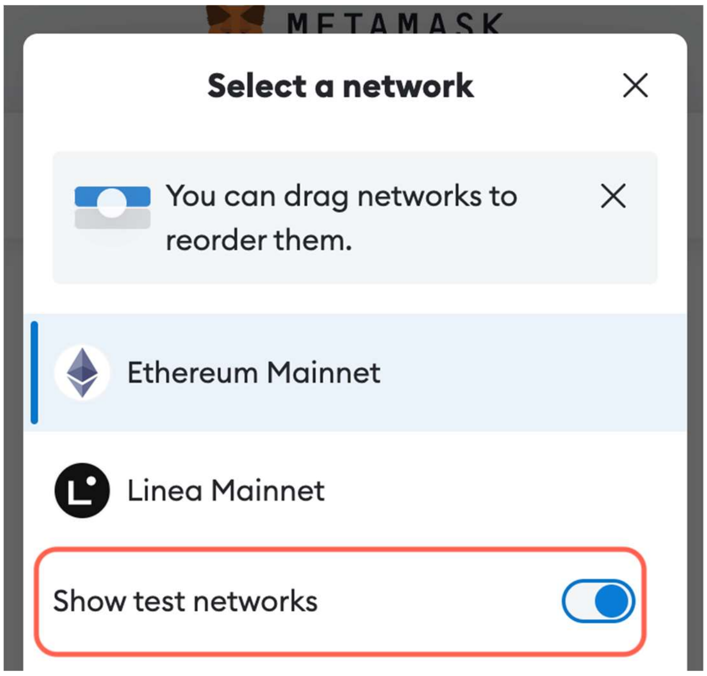
## Experiment – 3

**AIM:** Sepolia Faucet, How to Transfer ETH, Gas, Gas Price, Gas Limit and Opcodes, Ethereum Block Explorer, Ethereum Transactions Blocks , Ethereum Transaction's Fields

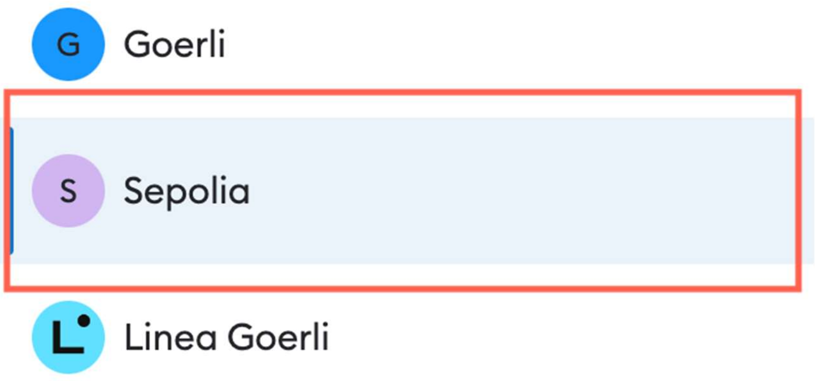
1. Login with credential and open metamask account



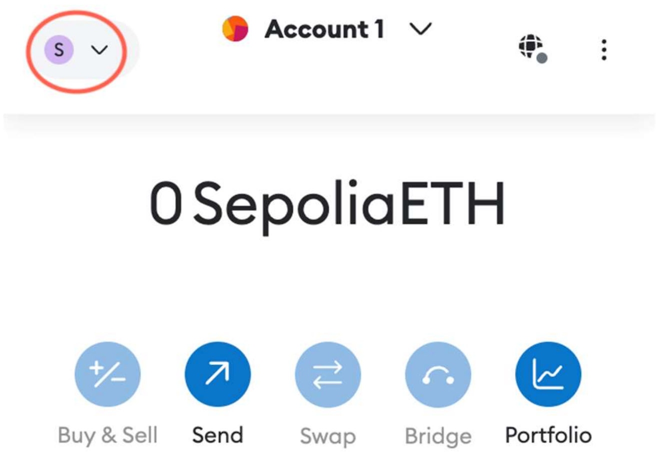
2. Click Ethereum Mainnet in the top left corner to open the Select a Network menu and toggle Show test networks to on.



3. You will need to select the Sepolia network. This is what we will use(Rinkeby network has been deprecated).



4. You can now close out the setup page and interact directly with the extension in your browser. If you have the correct network selected (Sepolia), you should see an S in the top left corner of the extension window.



**Getting Test Ether**

we will need to obtain some test Ether!

Now, we will use the Sepolia Network. This is 100% interchangeable with Rinkeby and Goerli, so, there will be no actual difference.

Since we will be using the Sepolia Network, we will need to use a Sepolia faucet to obtain test ether. Nearly all current faucets require some level of authentication to protect against DOS attacks and bots. Most faucets now require that you have real mainnet Ether to even request testnet Ether.

1. Go to website to get test ether
"https://cloud.google.com/application/web3/faucet/ethereum/sepolia"

2. Enter your Metamask wallet address and click the Recieve button.

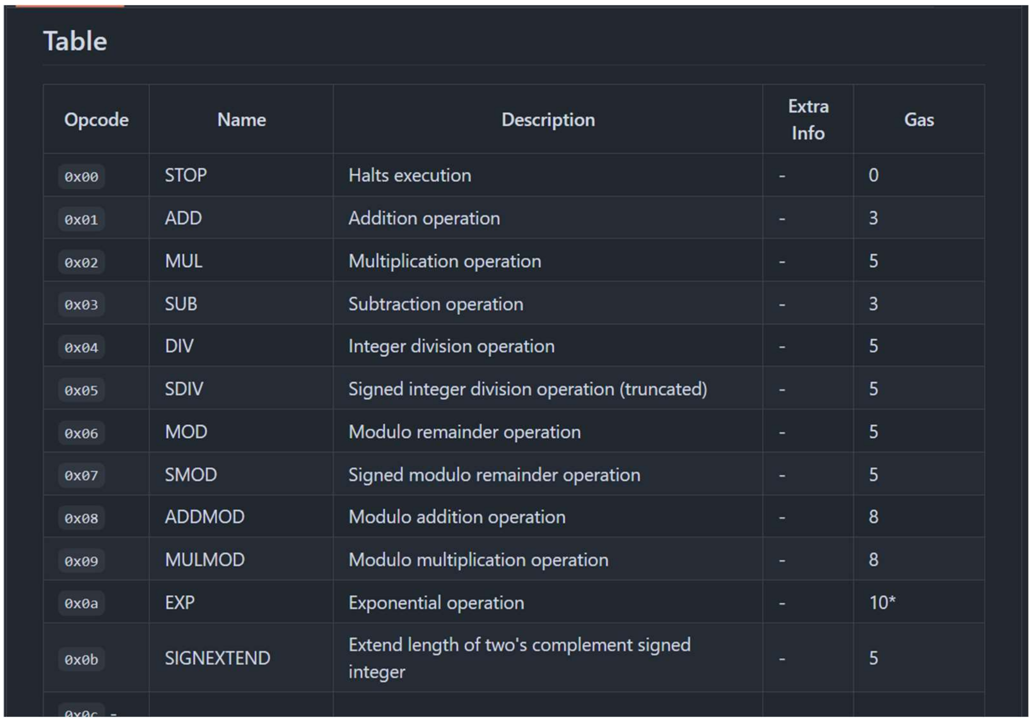
Your wallet should eventually receive .05 ether. You will only be able to request ether once a day, so, you should try to do this a few times over the span of a few days.

**Ethereum gas**

Ethereum gas is a measurement unit used to determine how much computational efforts is required to execute a particular transaction or smart contract on the Ethereum blockchain. In other words, it is a way of measuring how much "work" is required to be done to complete a transaction. The more complex the transaction, the more gas it will require. For example, a simple transfer of ETH from one address to another requires less gas than a smart contract that involves data storage, calculations and other operations.

Gas is essential because it prevents the Ethereum network from overloading many transactions. If a transaction requires too much it will be rejected by the network. Users are not charged for gas directly. Instead, they must pay a small amount of ETH for each transaction that they make. This ETH is then used to pay the miners who confirm the transactions on the blockchain.

**Opcode gas table**



Table

| Opcode | Name | Description | Extra Info | Gas |
|---|---|---|---|---|
| `0x00` | STOP | Halts execution | - | 0 |
| `0x01` | ADD | Addition operation | - | 3 |
| `0x02` | MUL | Multiplication operation | - | 5 |
| `0x03` | SUB | Subtraction operation | - | 3 |
| `0x04` | DIV | Integer division operation | - | 5 |
| `0x05` | SDIV | Signed integer division operation (truncated) | - | 5 |
| `0x06` | MOD | Modulo remainder operation | - | 5 |
| `0x07` | SMOD | Signed modulo remainder operation | - | 5 |
| `0x08` | ADDMOD | Modulo addition operation | - | 8 |
| `0x09` | MULMOD | Modulo multiplication operation | - | 8 |
| `0x0a` | EXP | Exponential operation | - | 10* |
| `0x0b` | SIGNEXTEND | Extend length of two's complement signed integer | - | 5 |

**Ethereum block explorer**

An Ethereum block explorer is a tool that allows users to explore and understand the Ethereum blockchain. It provides information on a variety of Ethereum activities, such as: Wallet balances, ERC-20 token transactions, NFT mints, Smart contract details, and Cryptocurrency prices.

## Experiment – 4

**AIM:** Ethereum 2.0 (ETH2). PoW vs. PoS , Eth2 Sharding, Links to Original Bitcoin & Ethereum White Papers

**Ethereum 2.0**

Ethereum 2.0, sometimes called Eth2 or Serenity, is an upgrade to the Ethereum blockchain. Its goal is to increase the speed, efficiency, and scalability of the Ethereum network while not compromising its security or decentralization.

**Proof of work.**

Cryptocurrencies do not have centralized gatekeepers to verify the accuracy of new transactions and data that are added to the blockchain. Instead, they rely on a distributed network of participants to validate incoming transactions and add them as new blocks on the chain.

Proof of work is a consensus mechanism to choose which of these network participants, called miners are allowed to handle the lucrative task of verifying new data. It is lucrative because the miners are rewarded with new crypto when they accurately validate the new data and do not cheat the system.

"Proof of work is a software algorithm used by Bitcoin and other blockchains to ensure blocks are only regarded as valid if they require a certain amount of computational power to produce". It is a consensus mechanism that allows anonymous entities in decentralized networks to trust one another.

The "work" in proof of work is key -- The system requires miners to compete with each other to be the first to solve arbitrary mathematical puzzles to prevent anybody from gaming the system. The winner of this race is selected to add the newest batch of data or transactions to the blockchain.

Winning miners only receive their reward of new cryptocurrency after other participants in the network verify that the data being added to the chain is correct and valid.

**Proof of stake.**

Proof of stake is a type of consensus mechanism used to validate cryptocurrency transactions. With this system, owners of the cryptocurrency can stake their coins, which gives them the right to check new blocks of transactions and add them to the blockchain.

This method is an alternative to proof of work, the first consensus mechanism developed for cryptocurrencies. Since proof of stake is much more energy-efficient, it has gotten more popular as attention has turned to how crypto mining affects the planet.

Working of Proof of Stake - The proof-of-stake model allows owners of a cryptocurrency to stake coins and create their own validator nodes. Staking is when you pledge your coins to be used for verifying transactions. Your coins are locked up while you stake them, but you can unstack them if you want to trade them.

When a block of transactions is ready to be processed, the crypto- currency's proof-of-stake protocol will choose a validator node to review the block. The validator checks if the transactions in the block are accurate. If so, they add the block to the blockchain and receive crypto rewards for their contribution. However, if a validator proposes adding a block with inaccurate information, they lose some of their staked holdings as a penalty.

**Ethereum 2.0 sharding**

Ethereum 2.0 sharding is a scaling solution that divides the Ethereum network into smaller, interconnected networks called shards:

Sharding's goal is to improve the network's capacity and transaction speed, and reduce gas fees

**Ethereum Whitepaper**

This introductory paper was originally published in 2014 by Vitalik Buterin, the founder of Ethereum, before the project's launch in 2015. It's worth noting that Ethereum, like many community-driven, open-source software projects, has evolved since its initial inception.

**The Bitcoin Whitepaper**

The Bitcoin Whitepaper was originally published on the 31st of October, 2008 by an individual or a group of people that called themselves Satoshi Nakamoto in a cryptography mailing list on a platform called Metzdowd. Wild theories and myths surround the actual identity of the creator(s) of Bitcoin - a number of individuals known in the world of cryptocurrencies have claimed to be Satoshi Nakamoto or to know them.

Published in October 2008, by an individual or a group under the name of Satoshi Nakamoto, the Bitcoin Whitepaper laid the groundwork for a fundamental change in the execution of global payments and the transformation of entire industries in terms of data management.

Bitcoin was introduced as a digital currency to the world in the paper called Bitcoin: A Peer-to-Peer Electronic Cash System

The Bitcoin Whitepaper is only nine pages long and is a proposal for a trustless system of electronic transactions

The Bitcoin network creates a structure for making payments without a trusted third party acting as intermediary

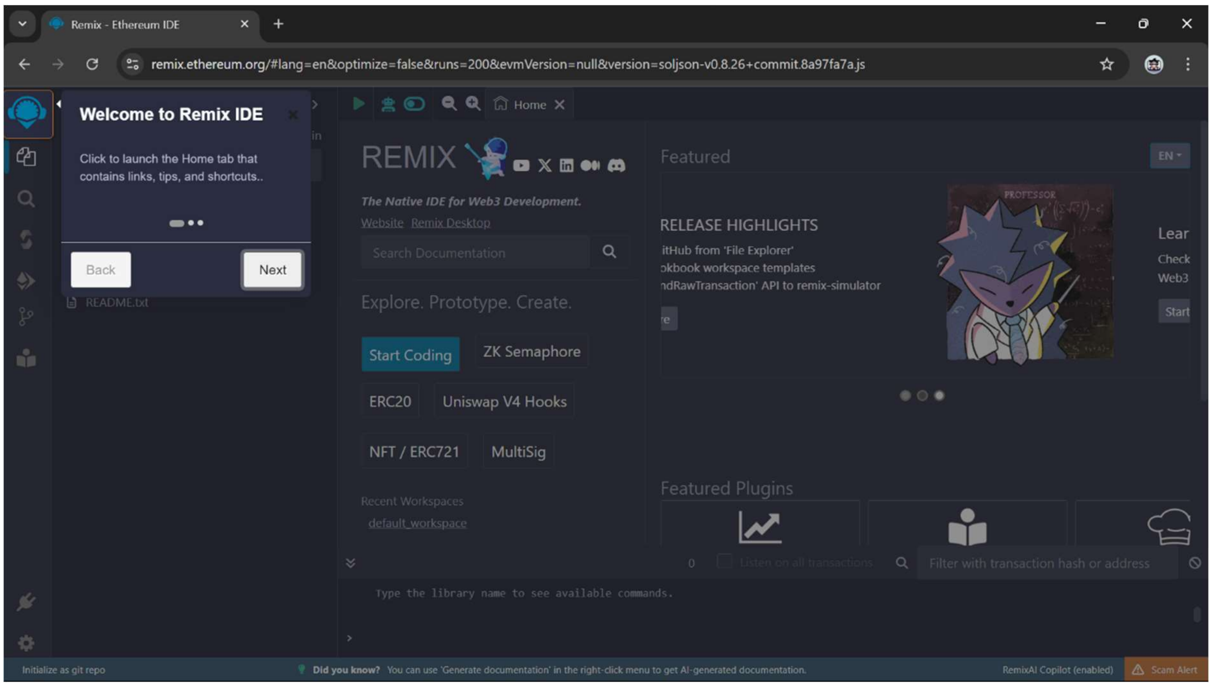
The concept behind Bitcoin is based on cryptography, the study of secure communication technologies and development of protocols preventing the public or third parties from reading private information.

## Experiment – 5

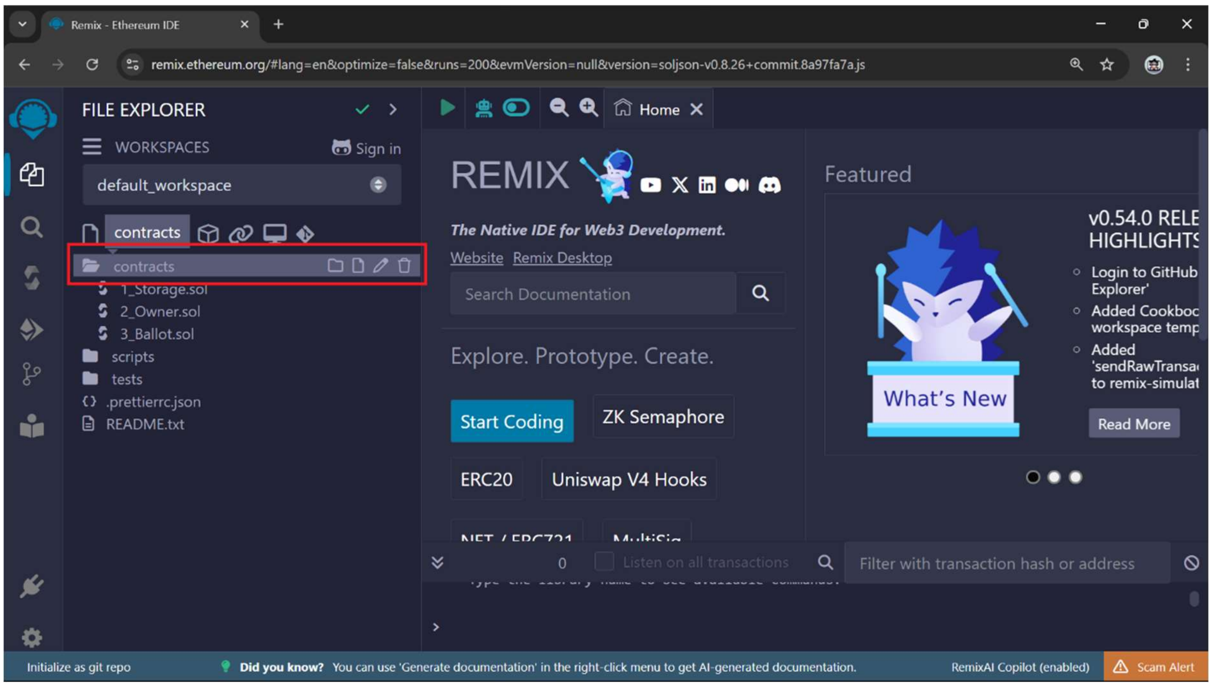
**AIM:** Remix IDE, Compilation In Depth: ABI and Bytecode, Contract Deployment on JSVM Contract Deployment on Sepolia Using Remix and MetaMask,

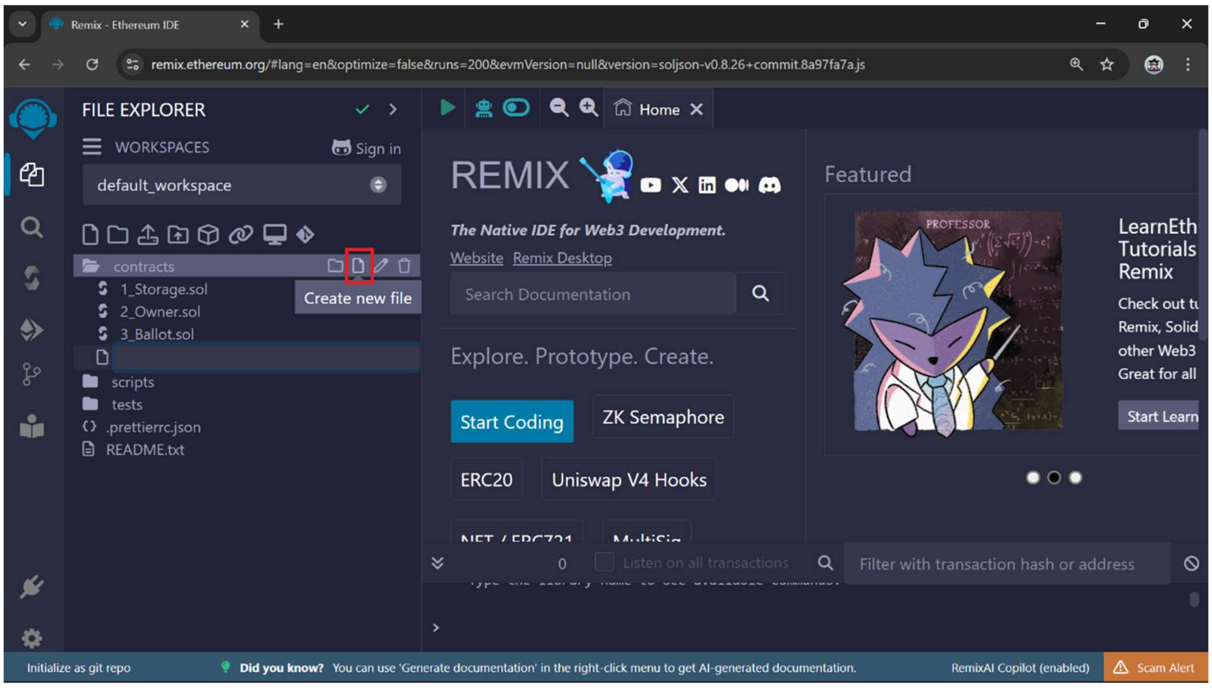
Remix IDE is a free, open-source, browser-based tool for developing, debugging, deploying, and testing smart contracts for the Ethereum network and other blockchains

1. Go to “https://remix.ethereum.org/”



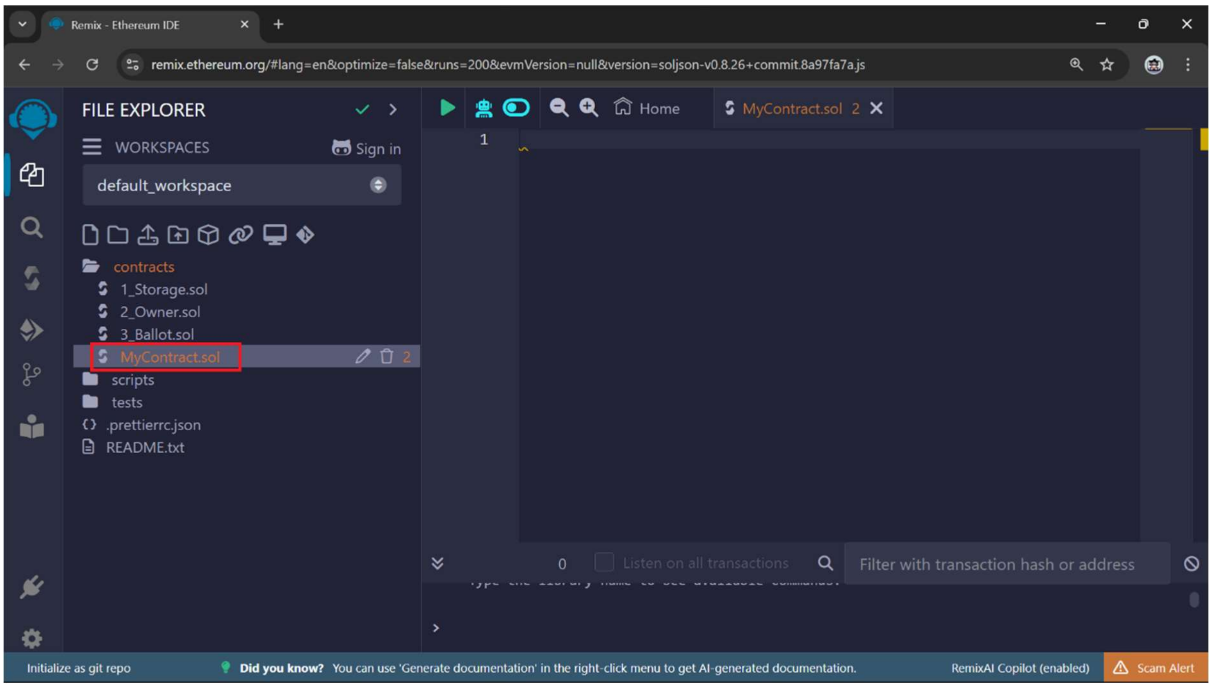
2. Create a file with name





Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

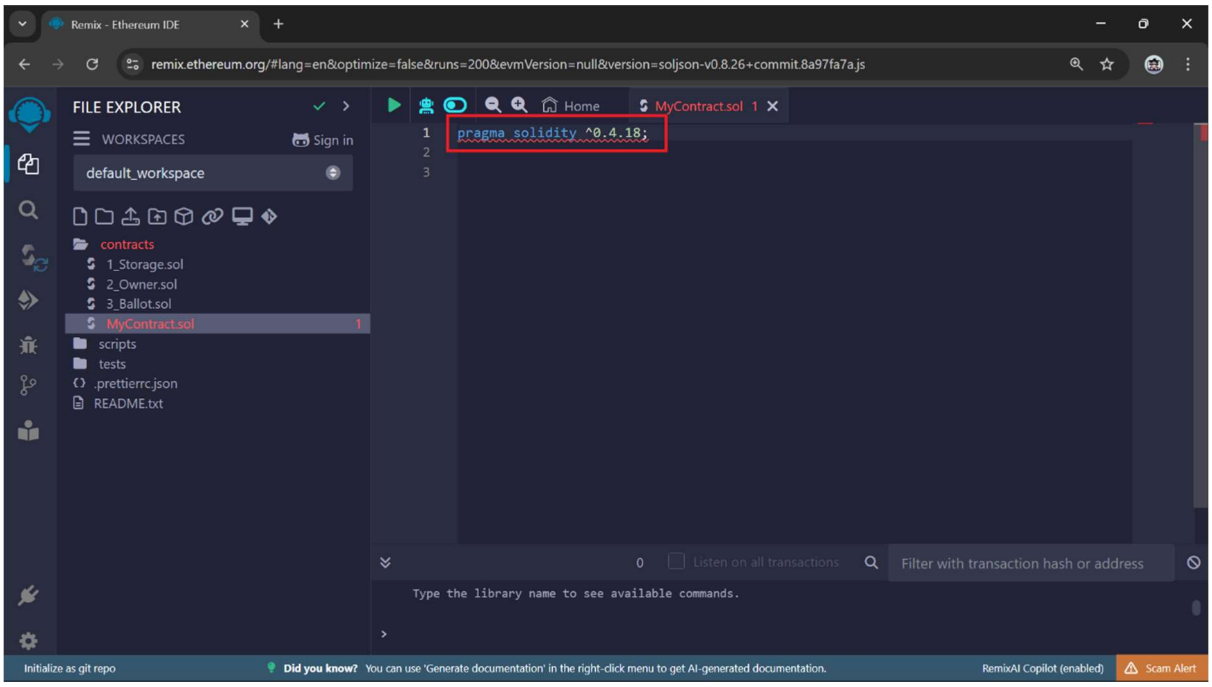
RemixAI Copilot (enabled)



Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

RemixAI Copilot (enabled)

3. writing smart contract version



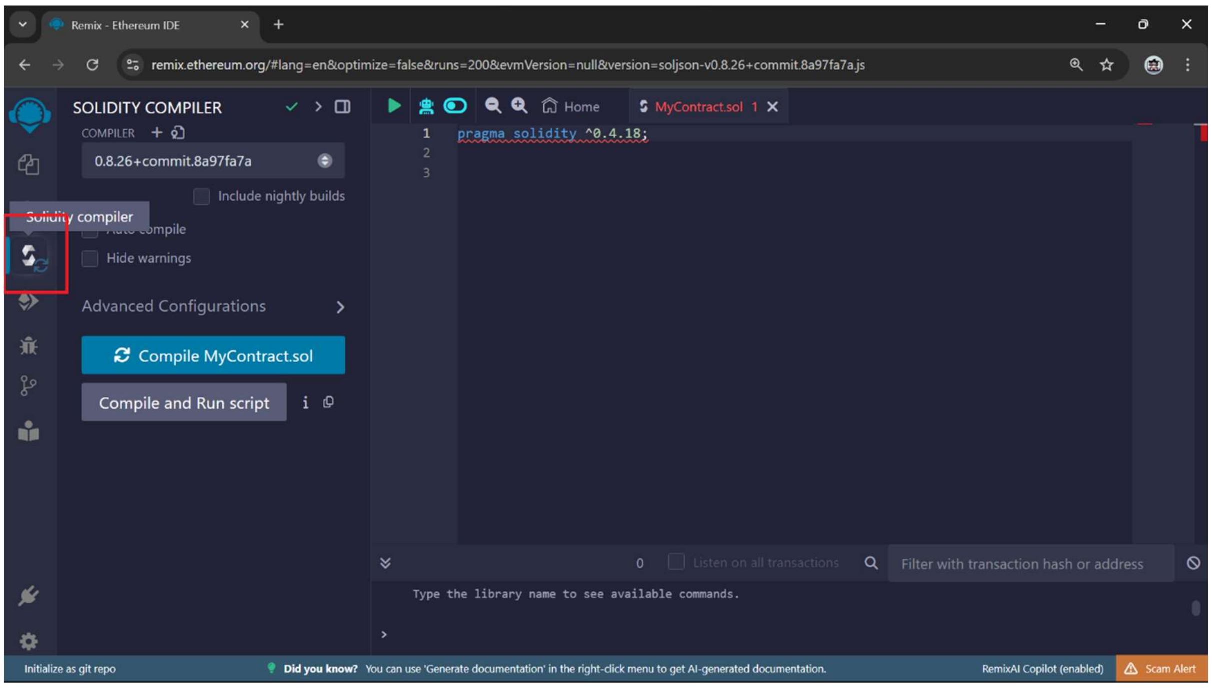
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

```
pragma solidity ^0.4.18;
```

```
Type the library name to see available commands.
```

Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

4. Selecting compiler version



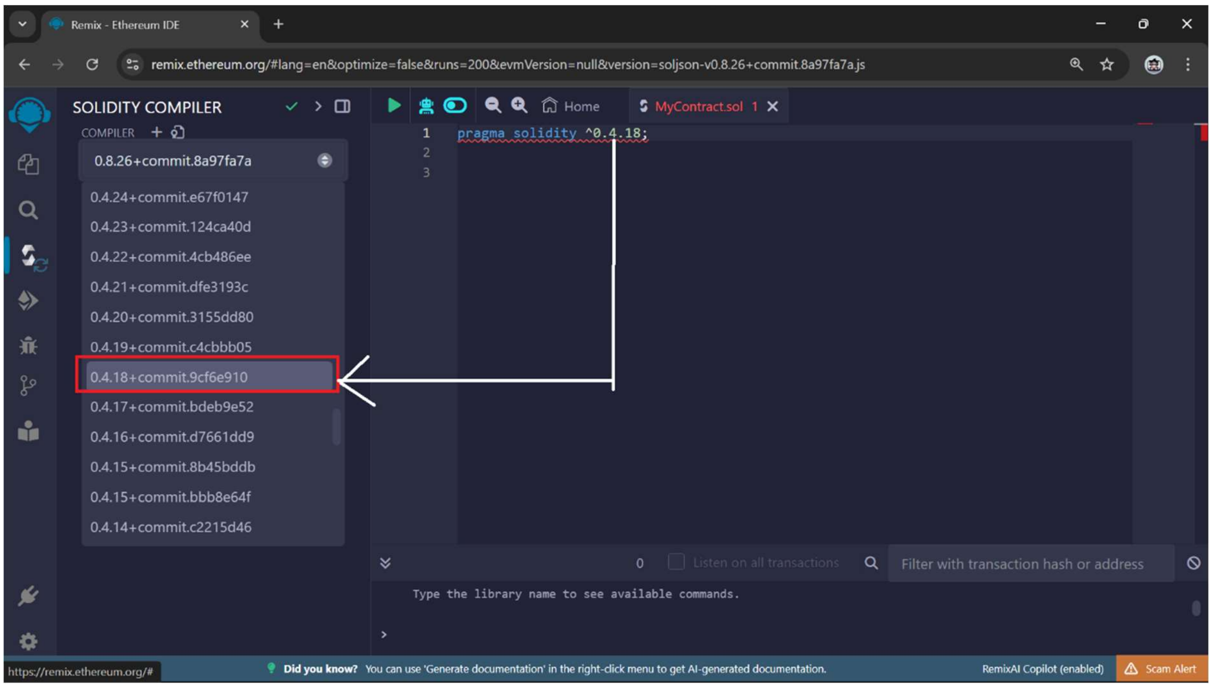
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

COMPILER: 0.8.26+commit.8a97fa7a

```
pragma solidity ^0.4.18;
```

```
Type the library name to see available commands.
```

Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.



remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js

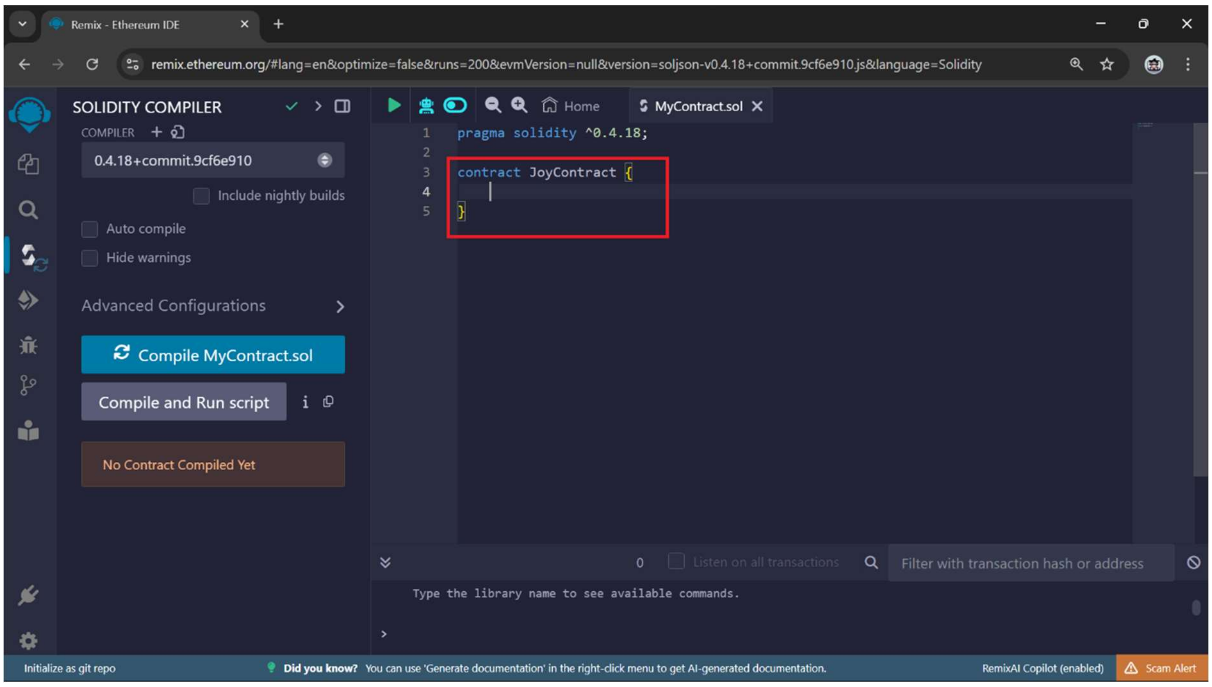
```
pragma solidity ^0.4.18;
```

```
Type the library name to see available commands.
```

https://remix.ethereum.org/#

**Did you know?** You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

5. writing first contract



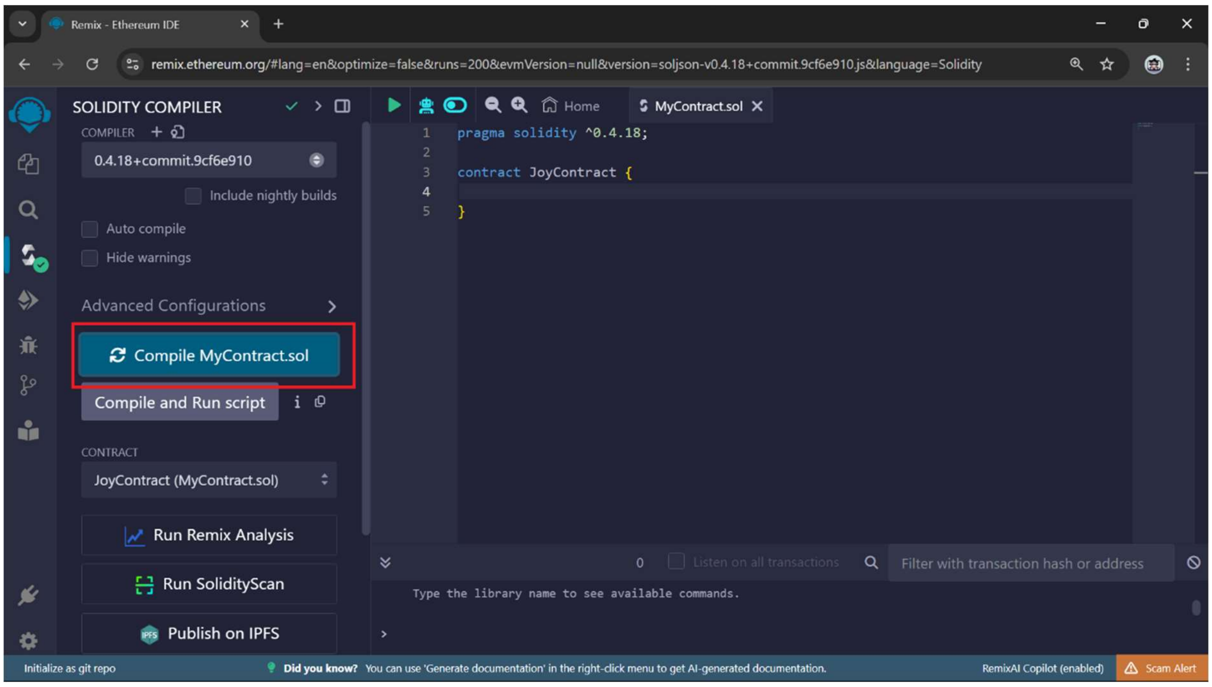
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.18+commit.9cf6e910.js&language=Solidity

```
pragma solidity ^0.4.18;

contract JoyContract {

}
```

```
Type the library name to see available commands.
```

**Did you know?** You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

6. compiling the smart contract


remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.18+commit.9cf6e910.js&language=Solidity

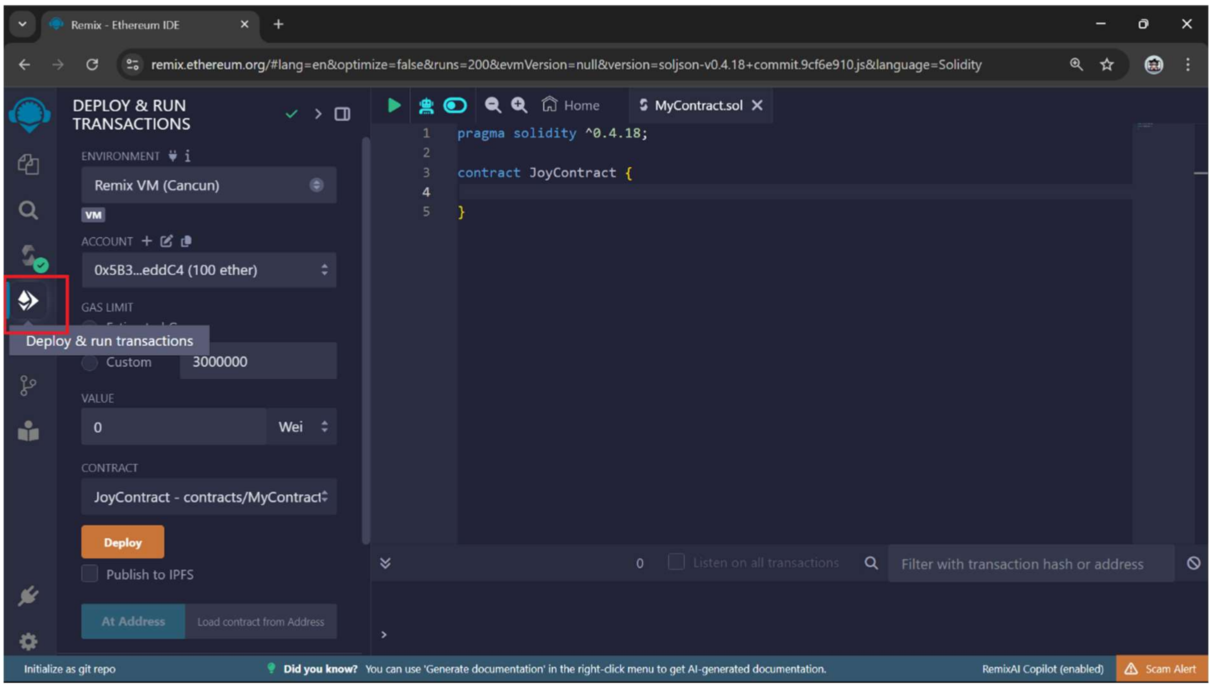
SOLIDITY COMPILER

COMPILER: 0.4.18+commit.9cf6e910

CONTRACT: JoyContract (MyContract.sol)

```
pragma solidity ^0.4.18;

contract JoyContract {

}
```

```
Type the library name to see available commands.
```


remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.18+commit.9cf6e910.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Cancun)

ACCOUNT: 0x5B3...eddC4 (100 ether)

GAS LIMIT: Custom 3000000

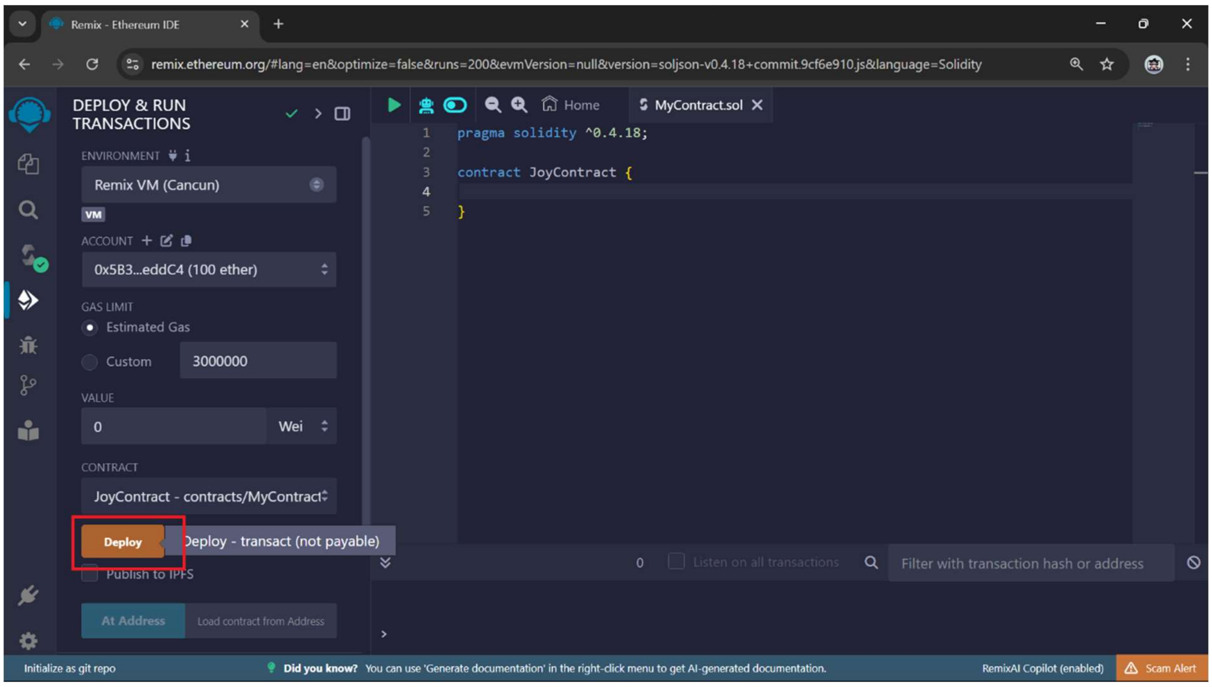
VALUE: 0 Wei

CONTRACT: JoyContract - contracts/MyContract

```
pragma solidity ^0.4.18;

contract JoyContract {

}
```

Remix - Ethereum IDE
VM
Deploy & run transactions
Deploy
Publish to IPFS
At Address
Load contract from Address
Home
MyContract.sol
Listen on all transactions
Filter with transaction hash or address
Initialize as git repo
Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.
RemixAI Copilot (enabled)
Scam Alert

7. Deploying smart contract on JavaScript Ethereum virtual blockchain network for testing purpose


remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.4.18+commit.9cf6e910.js&language=Solidity

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Cancun)

ACCOUNT: 0x5B3...eddC4 (100 ether)

GAS LIMIT: Estimated Gas; Custom 3000000

VALUE: 0 Wei

CONTRACT: JoyContract - contracts/MyContract.sol

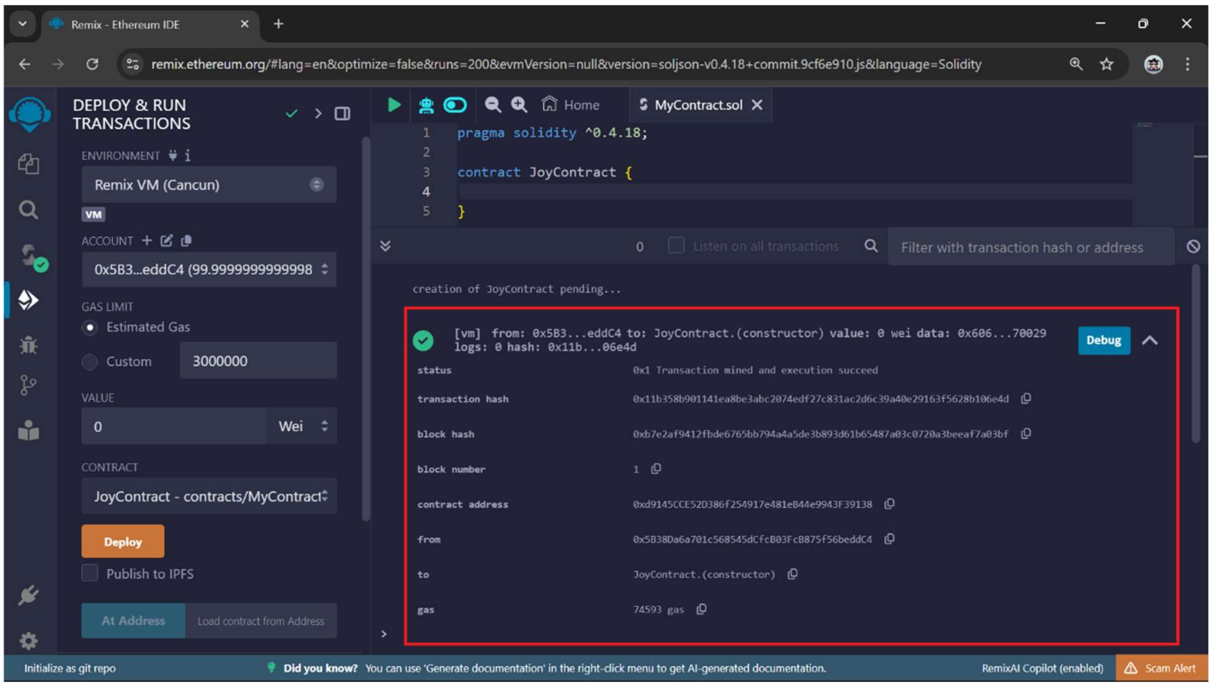
Deploy — Deploy - transact (not payable)

```
pragma solidity ^0.4.18;

contract JoyContract {

}
```

Did you know? You can use 'Generate documentation' in the right-click menu to get AI-generated documentation.

Remix - Ethereum IDE
Home
MyContract.sol
Publish to IPFS
At Address
Load contract from Address
Listen on all transactions
Filter with transaction hash or address
Initialize as git repo
RemixAI Copilot (enabled)
Scam Alert

8. Checking out fields of deployed block for instance gas use, address, block number, etc.


ACCOUNT: 0x5B3...eddC4 (99.9999999999998

```
creation of JoyContract pending...

[vm] from: 0x5B3...eddC4 to: JoyContract.(constructor) value: 0 wei data: 0x606...70029
logs: 0 hash: 0x11b...06e4d
status              0x1 Transaction mined and execution succeed
transaction hash    0x11b358b901141ea8be3abc2074edf27c831ac2d6c39a40e29163f5628b106e4d
block hash          0xb7e2af9412fbde6765bb794a4a5de3b893d61b65487a03c0720a3beeaf7a03bf
block number        1
contract address    0xd9145CCE52D386f254917e481eB44e9943F39138
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  JoyContract.(constructor)
gas                 74593 gas
```

## Bytecode in Ethereum

Bytecode in Ethereum refers to the low-level machine-readable code that is stored on the blockchain and executed by the Ethereum Virtual Machine (EVM). Smart contracts in Ethereum are written in a high-level programming language like Solidity and then compiled into bytecode that can be executed by the EVM

When a smart contract is deployed on the Ethereum blockchain, its bytecode is stored on the blockchain as a transaction. The EVM executes the bytecode to validate transactions and execute smart contract logic since the bytecode is executed by the EVM. Hence, it is platform-independent, meaning smart contracts can be executed on any Ethereum node regardless of the hardware and software it is running on.

## ABI

ABI stands for Application Binary Interface, and it is a specification that defines how two pieces of software interact with each other. Specifically, an ABI specifies things like how functions are called, how data is passed between the two programs, and how errors are handled. By standardizing these interactions, an ABI makes it possible for software written in different programming languages or on different platforms to work together seamlessly.

One way to think about an ABI is as a contract between two programs. Just like a legal contract specifies the rights and responsibilities of each party, an ABI specifies how two programs will interact with each other. This makes it possible for developers to write software that works with other software, even if they don't have direct access to the source code.

# Experiment – 6

**AIM:** The Structure of a Smart Contract, Solidity Basic Syntax Rules, State and Local Variables, Functions, Setters, and Getters

Solidity is a brand-new programming language created by Ethercum which is the second-largest market of cryptocurrency by capitalization, released in the year 2015.

Some key features of solidity are as follows -

(i) Solidity is a high-level programming language designed for implementing smart contracts.

(ii) It is a statistically typed objcct-orientcd (contract-oriented) language.

(iii) Solidity is highly influenced by Python. C++ and JavaScript which run on the Ethercum Virtual Machinc (EVM).

(iv) Solidity supports complex user-defined programming, libraries and inheritance.

(v) Solidity is the primary language for blockchains running platforms.

(vi) Solidity can be used to create contracts like voting, blind auctions, crowdfunding. multi-signature wallets, etc.

## The Basic Structure of a Smart Contract

```
// declaring smart contract solidity language version
pragma solidity ^0.4.18;

// creating smart contract
contract JoyContract {

}
```

### Solidity Variable declarations

```
pragma solidity ^0.4.18;

contract JoyContract {
// How to use data tye and declare variable in solidity
   string name = "joy peter";
   uint age = 70;
}
```

**Solidity function deceleration getter & setter**

```
pragma solidity ^0.4.18;

contract JoyContract {
    string name;
    uint age;

    // setter function
    function setMessage(string _name, uint _age) public {
        name = _name;
        age = _age;
    }

    // getter function
    function getMessage() public view returns (string, uint) {
        return (name,age);
    }

}
```

## Experiment – 7

**AIM:** The Constructor, Coding - Variables and Functions, Variable Types: Booleans and Integers, SafeMath, Overflows and Underflows

**Variables**

In Solidity, a variable is a container for storing data. These variables can represent a wide range of data types, from simple numbers to complex structures. Solidity variables are crucial for defining the state of a contract and are used to store values that are manipulated during the contract's execution.

Solidity supports both value types and reference types, each with its specific use cases and behavior. Let's delve into these types and explore their uses.

**uint:** Unsigned integer types. These types are used to represent a positive integer or natural number. The keyword uint is a shorthand for uint256 which is a number that can range between 0 and 2256-1 . Other uint types include uint8, uint16, uint40…

**int:** Signed integer types. Similar to unsigned integers but these are normal integers that can be positive or negative.

**bool:** Boolean values that are used to declare whether a variable is true or false.

address: This is a type unique to Ethereum and Solidity. It's used to store the addresses of users or other contracts.

**enum:** User-defined types for creating a set of named constants.

bytes and bytes32: Eventually all the other types are stored in the form of bytes. Bytes themselves are used to store data that is not represented by those other types.

```
pragma solidity ^0.4.18;

contract JoyContract {
   // value types
   uint256 positiveNumber;
   int256 positiveOrNegativeNumber;
   bool trueOrFalse;
   address userOrContractAddress;
   bytes32 someData;
   string name;
   uint age;

    enum Colors {
      Red,
      Green,
      Blue
   }
   // constructor function
```

```
    function JoyContract () public {
      name = "joy peter";
      age = 70;
   }

}
```

**Experiment – 8**

**AIM:** Bytes and String Types, Structs and Enums, Coding - Structs and Enums, Mappings ,Coding – Mappings

Reference types store a reference to the data, which means that they don't hold a value themselves but rather hold a pointer to a certain location in memory or storage that can have many values. They are used because values such as text words and item lists don't have a definite size and so cannot be accessed directly like numbers or Booleans. The primary reference types in Solidity include:

strings: A collection of one or more characters. Any text from "Hello!" to "123" to "?" is considered a string.

arrays: A collection of data of the same type, like uint[] or bool[];

mappings: Key-value stores where data is organized in a mapping from keys to values.

structs: User-defined data structures that can contain a combination of different data types.

```
pragma solidity ^0.4.18;

contract JoyContract {
    string name;
    uint age;

   // reference types
    string text;
    bytes data;
    uint256[] positiveNumbersList;
    address[] userOrContractAddressesList;
    mapping(address => uint256) addressToPositiveNumber;
    mapping(address => uint256[]) addressToPositiveNumbersList;
    mapping(address => mapping(address => bool)) addressToAddressToTrueOrFalse;
    struct User {
        uint256 positiveNumber;
        address userOrContractAddress;
    }

    // constructor function
```

```
     function JoyContract () public {
        name = "joy peter";
        age = 70;
    }

    // setter
    function setMessage(string _name, uint _age) public {
        name = _name;
        age = _age;
    }

    // getter
    function getMessage() public view returns (string, uint) {
        return (name,age);
    }

}
```

**Experiment – 9**

**AIM:** Coding - Global Variables

Solidity is a popular programming language for writing smart contracts on the Ethereum blockchain. In Solidity, global variables play a crucial role in the functioning of smart contracts. These variables provide important information about the state and context of the blockchain, making it possible to create complex, decentralized applications. In this article, we will explore Solidity global variables, their different types, and their various uses.

Global variables in Solidity are predefined variables that hold information about the current state of the Ethereum blockchain and the execution context of a smart contract. These variables are accessible from any part of a contract and are often used to access blockchain-specific data or manipulate the contract's state. Understanding the types and uses of these global variables is essential for writing efficient and secure smart contracts.

**Block**

The block is where the block—chain got its name. The blockchain works by creating new blocks and chaining them together linearly. Anytime a new block is created, it carries a set of transactions with it, those being invocations to smart contract functions. Those same functions can have access to their current carrier block, which grants them access to its information.

A block's information includes details about its number in the blockchain, its mining difficulty, and a bunch of other stuff. The main attribute that we need to focus on however, is the block timestamp.

The timestamp of a block indicates when this block has been mined. This is very important because it allows us to keep records of when a transaction has occurred. We can use those records to restrict interaction with a contract for a certain amount of time, or reward certain users based on timely dependent interactions with the contract.

```
pragma solidity ^0.8.18;

contract GlobalVariables {
  uint256 constant MIN_TIME_BETWEEN_INTERACTIONS = 15 minutes;
  uint256 lastInteraction;

  function doSomeStuff() external {
    require(
      block.timestamp - lastInteraction > MIN_TIME_BETWEEN_INTERACTIONS,
      "Not enough time has passed since last interaction"
    );
    lastInteraction = block.timestamp;
    // do some stuff
  }
}
```

**Msg**

Similar to how a block carries information about its attributes in the blockchain, a msg carries information about a contract function call.

A msg also has a few attributes, the most common of which are the sender and value.

The sender of a msg, as the name suggest, is the person or contract calling the function. The value on the other hand, contrary to what its name suggests, is not about the inputs being given to the function. The value is actually the amount of WEi sent to the contract, which is a currency in the Ethereum ecosystem that can be converted to Ether.

We can use the msg sender to keep track of certain users or contracts and their interactions with ours. And we can use the msg value to handle payments in our contract.

```
pragma solidity ^0.8.18;

contract GlobalVariables {
    uint256 constant MIN_DONATION = 0.01 ether;
    uint256 constant MAX_DONATION = 100 ether;
    mapping(address => uint256) public donations;

    function donate() external payable {
        require(
            msg.value >= MIN_DONATION,
            "Donation must be at least 0.01 ether"
        );
        require(msg.value <= MAX_DONATION, "Donation can be at most 100 ether");
        donations[msg.sender] += msg.value;
        // do some stuff
    }
}
```

**Tx**

The word tx is a shorthand for the word transaction. You might be wondering what is the difference between a msg and a tx then. While the msg carries data about the function call, the tx carries data about the entire transaction chain. The only properties exposed by tx are gasprice and origin.

To give an example: if a user calls contract A, which in turn calls contract B, the msg sender in contract B would be contract A, while the tx origin would be the user themselves.

The tx is rarely a variable that you would use, and it’s generally a good practice to avoid it.

```
pragma solidity ^0.8.18;

contract GlobalVariables {
    mapping(address => uint256) public donations;

    //* good
    function donate() external payable {
        donations[msg.sender] += msg.value;

        // do some stuff
    }

    //! bad
    function donateFromOrigin() external payable {
        donations[tx.origin] += msg.value;

        // do some stuff
    }
}
```

## Experiment – 10

**AIM:** Variables and Functions Visibility: Private, Public, Internal, External

### Solidity Variables: Visibility

Variables can be declared with different visibility modifiers, such as public, private, internal, which determine who can access the variable. Let’s take a look at Solidity public variables, private variables and internal variables.

Public variables, as their name suggests, are accessible by everyone including the contract itself.

Private variables on the other hand, are ones that can only be read by the contract itself.

Internal variables are a special case that can be used by contracts that inherit our contract. We will go in depth about contract inheritance in a separate article so stay tuned.

```
contract Variables {
   uint256 public notSecretNumber;
   uint256 private secretNumber;
   uint256 internal internalNumber;

 // when you don't specify a visibility, it defaults to public
   uint256 number; // same as "public uint256 number";
}

contract VariablesClone is Variables {
   // we can access notSecretNumber
   // we can access internalNumber
   // we can not access secretNumber

   function accessNumbers() public {
      notSecretNumber = 1;
      internalNumber = 2;
      // secretNumber = 3; // this will fail
   }
}
```

It must be clarified that once a Solidity contract is live on the blockchain, anyone and everyone can have access to the data inside of it. Changing a variable’s visibility affects how other contracts can interact with that variable from within our contract, but it doesn’t mean that the variable cannot be publicly accessed.