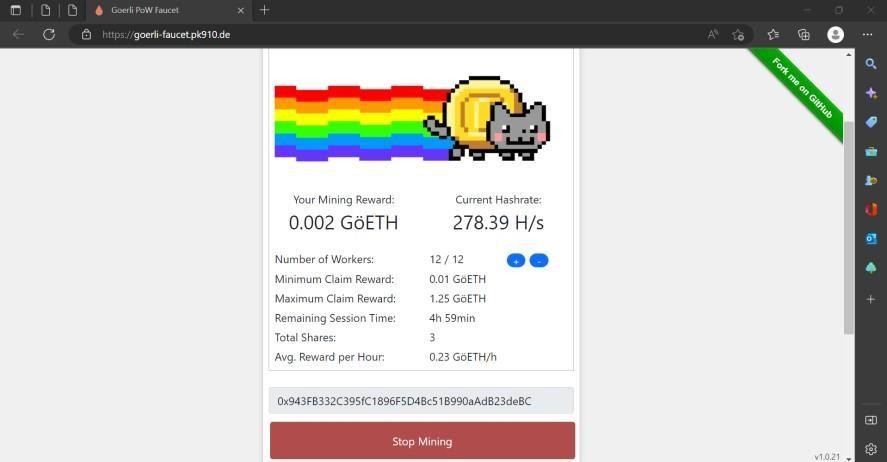
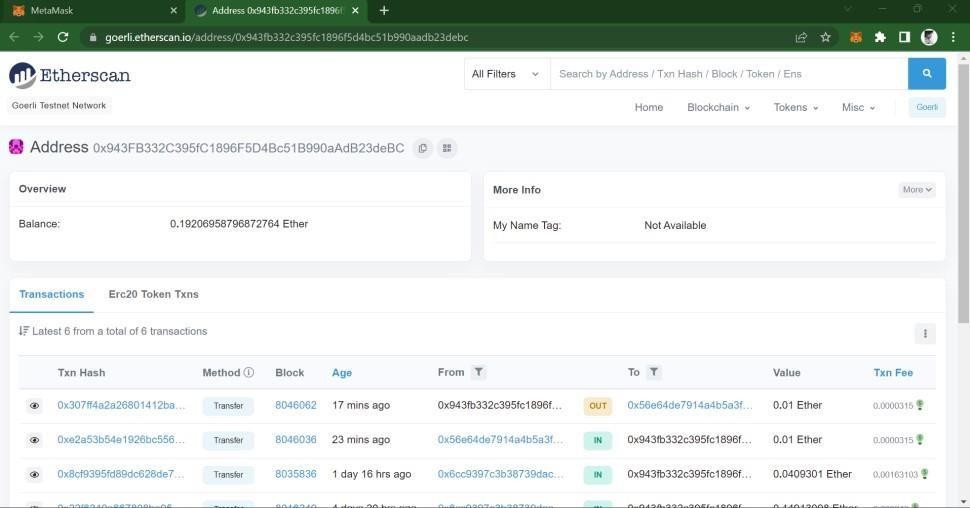
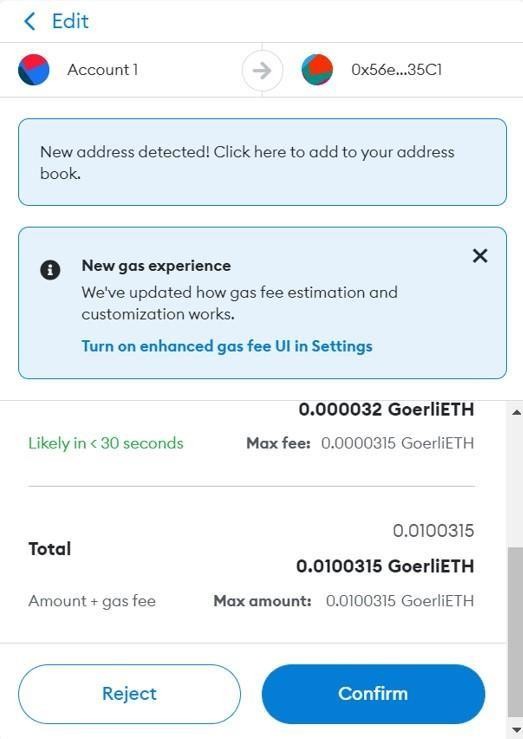
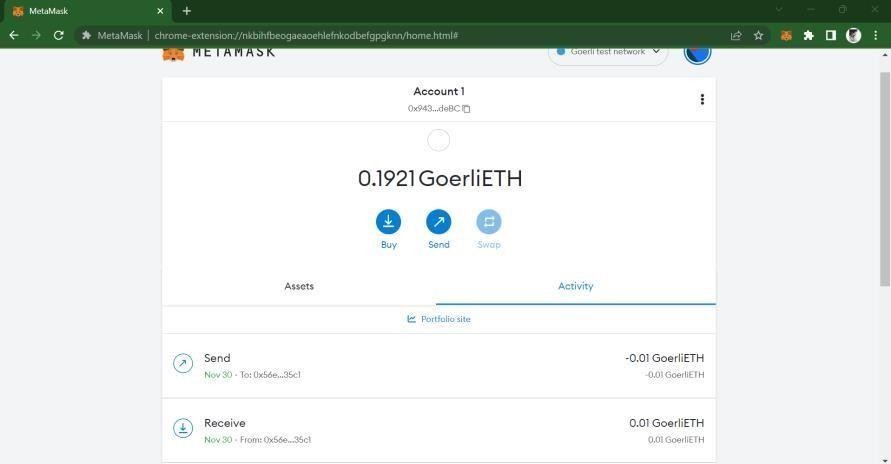
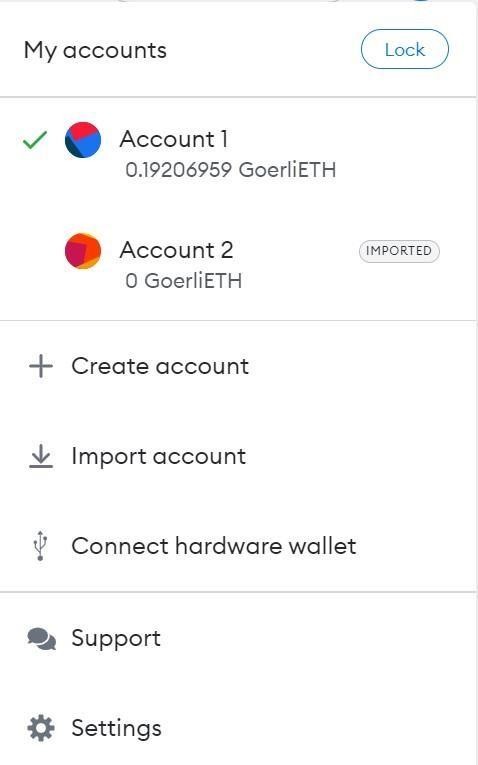
Practical No.04

1. **Install the metamask in browser. Setup the metamask digital cryptocurrency wallet. Create multiple accounts in metamask and connect with one of the etherum blockchain test network.Perform the task buy ethers and send ethers from one account to another. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction.(Use following url to**

get free ether for Goerli Test Network:



[https://goerlifaucet.pk910.de/)](https://goerli-faucet.pk910.de/)



1. **Write a solidity smart contract to transfer funds (ethers) from user account to contract account using remixIDE and JavaScriptVM environment.**

Program:

pragma solidity ^0.5.2; contract Financialcontract2{ address owner;

constructor() public{ owner=msg.sender;

}

modifier ifOwner(){ if(owner!=msg.sender){

revert();

} else{

\_;

}

}

function receiveDeposit() payable public{

}

function getBalance() public view returns(uint)

{

return address(this).balance;

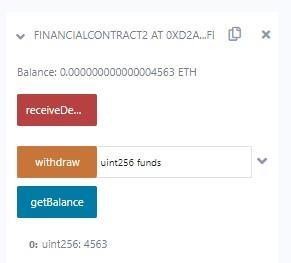
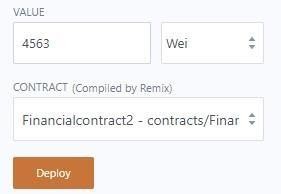
}

function withdraw(uint funds)public ifOwner{ msg.sender.transfer(funds);

}

Output:

}



1. **Write a solidity smart contract to withdraw funds (ethers) from contract account to user account using remixIDE and JavaScriptVM environment. Program:**

pragma solidity ^0.5.2;

contract Financialcontract2{ address owner; constructor()

public{

owner=msg.sender;

}

modifier ifOwner(){ if(owner!=msg.sender){

revert();

} else{

\_;

}

}

function receiveDeposit() payable public{

}

function getBalance() public view returns(uint)

{

return address(this).balance;

}

function withdraw(uint funds)public ifOwner{ msg.sender.transfer(funds);

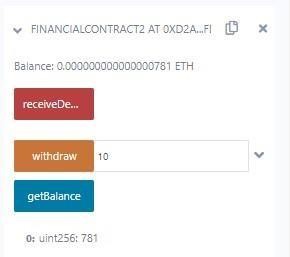
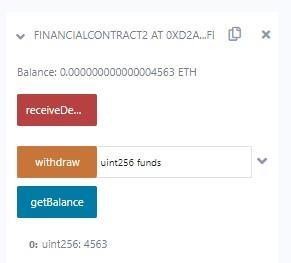
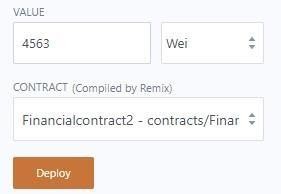
}

function getMoney(){

}

}

Output:



1. **Write a solidity smart contract to apply restriction that only owner of the contract can withdraw funds (ethers) from contract account to his/her user account using remixIDE and JavaScriptVM environment.**

Program:

pragma solidity ^0.5.2;

contract Financialcontract2{ address owner; constructor()

public{

owner=msg.sender;

}

modifier ifOwner(){ if(owner!=msg.sender){

revert();

} else{

\_;

}

}

function receiveDeposit() payable public{

}

function getBalance() public view returns(uint)

{

return address(this).balance;

}

function withdraw(uint funds)public ifOwner{ msg.sender.transfer(funds);

}

function getMoney(){

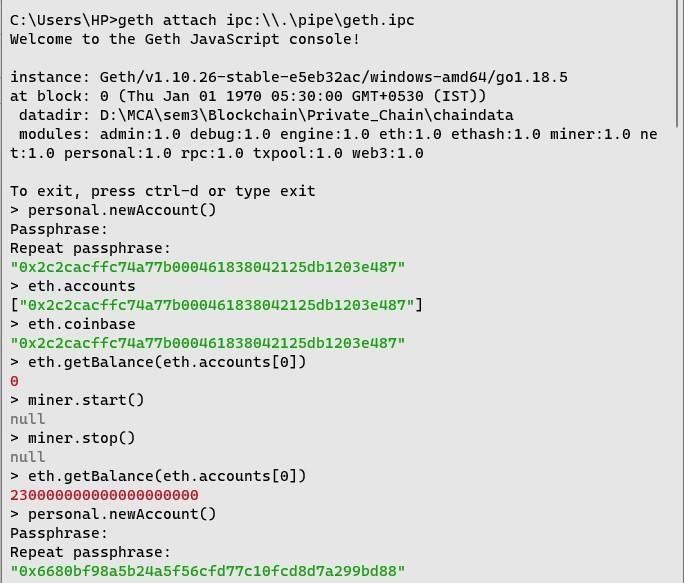
} }

Output:

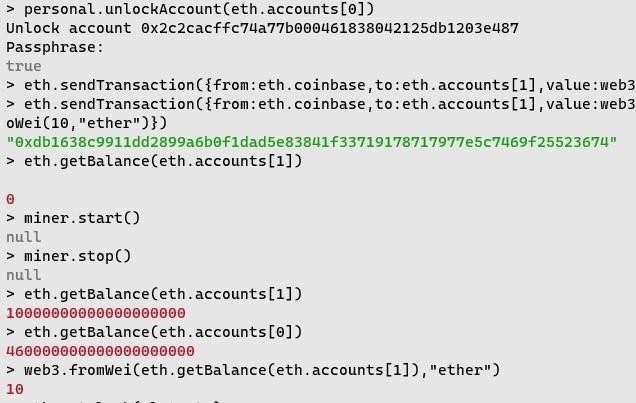


1. **Create Ethereum node using Geth and create genesis block and create your personal private Ethereum blockchain. And use IPC to interact with Geth node to perform following task: create account, transfer funds using send transaction, mine the block, show the account balance before and after the mining the block, show the specific block details and access chain details.**

Create Account



Transfer funds, mine block and account balance before and after the mining of block

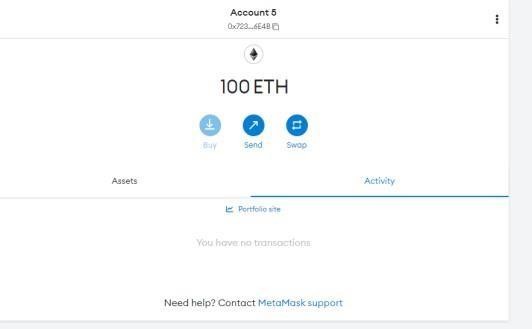
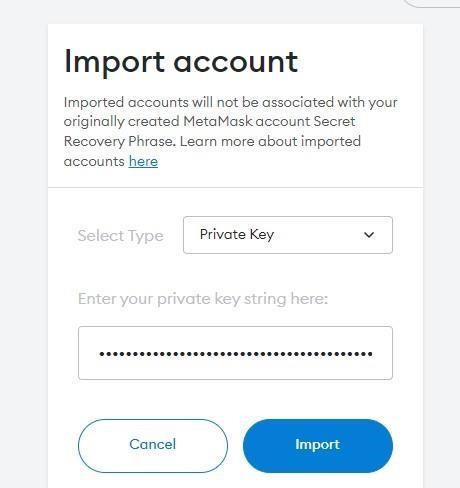
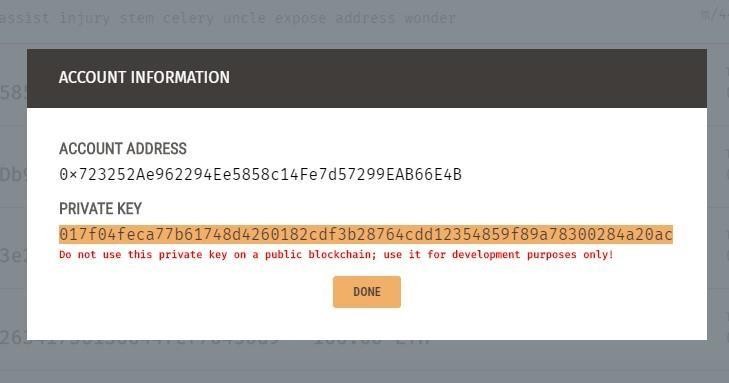


Specific block details

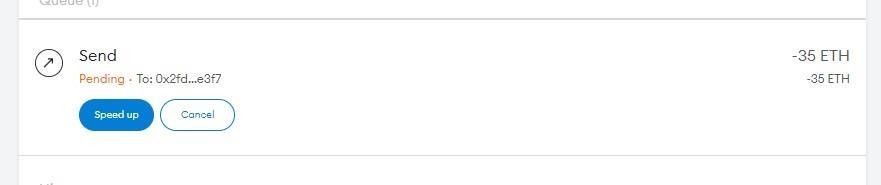
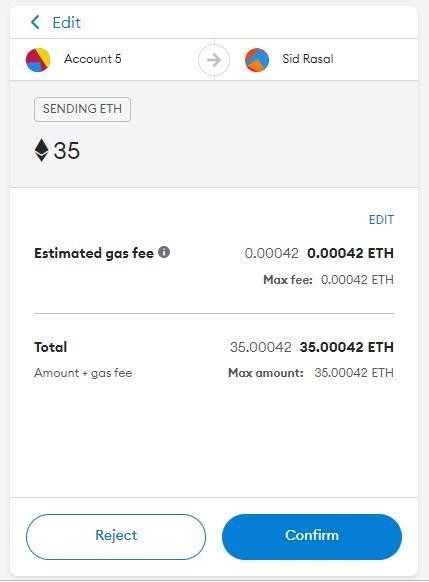
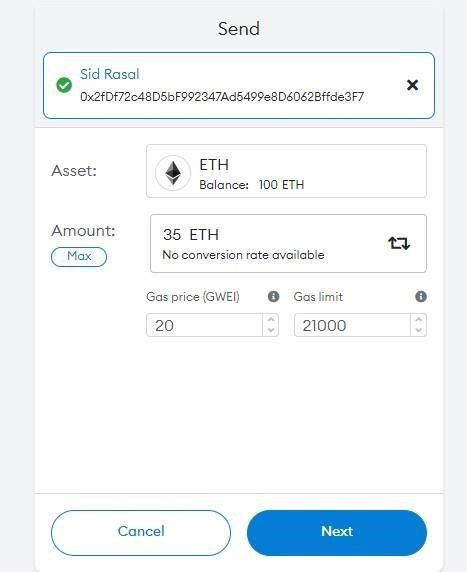


1. **Start Ganache your personal private blockchain network. Connect Ganache with MetaMask and import the account from Ganache to MetaMask. Transfer funds from imported account to other account of MetaMask. Take the screenshots of created accounts, account assets and account transactions which showing the details of transaction from MetaMask and Ganache interface.**

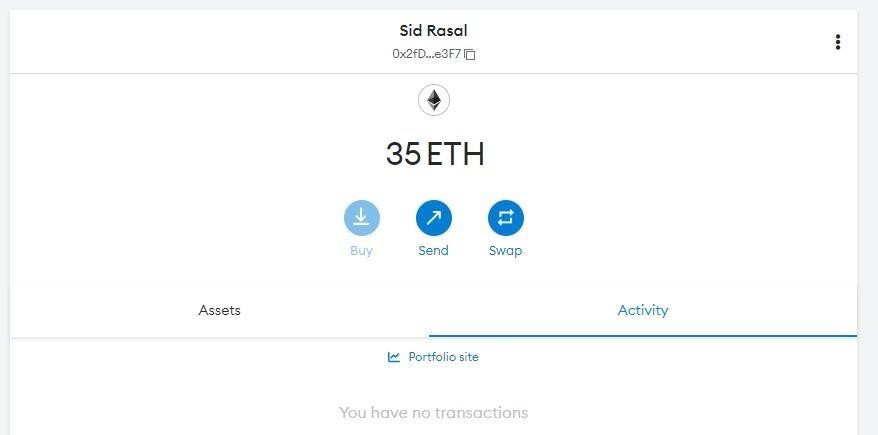
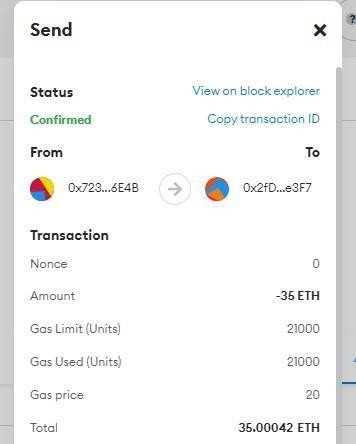
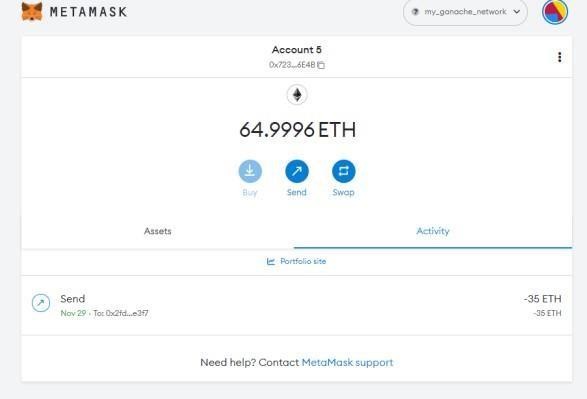
Import Account



Transfer Ether to another Account



Ether Transferred Successfully



1. **Create new truffle project with migration script, smart contract and configuration file. First compile it using truffle suite. Then connect it with personal private blockchain i.e. Ganache and deploy (migrate) smart contract on Ganache. Open truffle console and create instance of deployed (migrated) contract of Ganache. Then interact with smart contract using created instance. Take screenshots of all transaction’s details and block details from Ganache. Truffle-config.js**

module.exports =

{

networks:

{

development:

{ host:

"127.0.0.1", port:

7545, network\_id: "\*" // Match any network id

}

},

solc:

{

optimizer:

{

enabled: true,

runs: 200

}

}

}

Package.json

{

"name": "blockchain-toolkit", "version": "1.0.0",

"description": "The Complete Blockchain Developer Toolkit for 2019 & Beyond", "main": "truffle-config.js",

"directories": { "test": "test"

},

"scripts": {

"dev": "lite-server",

"test": "echo \"Error: no test specified\" && sexit 1"

},

"author": ["grego](mailto:gregory@dappuniversity.com)r[y@dappuniversity.com",](mailto:gregory@dappuniversity.com) "license": "ISC",

"devDependencies": { "bootstrap": "4.1.3",

"chai": "^4.1.2",

"chai-as-promised": "^7.1.1",

"chai-bignumber": "^2.0.2",

"dotenv": "^4.0.0",

"ganache-cli": "^6.1.8",

"lite-server": "^2.3.0",

"nodemon": "^1.17.3", "solidity-coverage": "^0.4.15", "truffle": "5.0.0-beta.0",

"truffle-contract": "3.0.6",

"truffle-hdwallet-provider": "^1.0.0-web3one.0"

}

}

2\_deploy\_contracts.js

var MyContract = artifacts.require("./MyContract.sol"); module.exports = function(deployer)

{

deployer.deploy(MyContract);

};

**MyContract.sol:** pragma solidity

>=0.4.22 <0.9.0; contract MyContract { string value; constructor() public { value = "myValue"; }

function get() public view returns(string memory) { return value; }

function set(string memory \_value) public { value

= \_value; }

}

Output:

