BCT LAB RECORD

1)Brave browser

2)Mata mask installation
Account creation
12 phrase key

3)After account have details of a) public key
b) private key
c) send money
d) buy

4) Installation of node js

install Node.js

Checking installed file like **node -v.**
Checking npm (node package manager)version(default installed file) using command **npm -v**

5)configure npm by using command **npm init**
npm(node package manager)

6)web3 installation
Using command **npm install web3**

7) Folder and file creation
Using commands **mkdir myproject (folder name)**
For enter into folder **cd myproject**

8) for file creation and editing using **vi filename.js** for windows **Notepad filename.txt**

For enter into file ----→**I + enter**
exit the file using ----→**esc +: wq** (for save and quit)

9)Execution of file using **node filename.js**

```
const { Web3 } = require("web3");
const { ETH_DATA_FORMAT, DEFAULT_RETURN_FORMAT } = require("web3");
async function main() {
  // Configuring the connection to an Ethereum node
  const web3 = new Web3(
    new Web3.providers.HttpProvider(
      "https://sepolia.infura.io/v3/35c2b7c8ecc6493cb073943d1bb7d15a" //add your
api key
    )
  );
  const latestBlock = await web3.eth.getBlock("latest");
  const baseFeePerGas = latestBlock.baseFeePerGas;
  const maxFeePerGas =
    BigInt(baseFeePerGas) + BigInt(web3.utils.toWei("2", "gwei"));
  // Creating a signing account from a private key
  const signer = web3.eth.accounts.privateKeyToAccount(
    "45e8183d1609d34a0008ed95b6ec625c07ffbe46aa593a52c2051806163a0874"
  ); //add your private

  web3.eth.accounts.wallet.add(signer);
  await web3.eth
    .estimateGas(
      {
        from: signer.address,
        to: "0x1328b19533f4bA383c5721845c4478b3B4e7A388", //Add
recipient_address
        value: web3.utils.toWei("0.0001", "ether"),
      },
      "latest",
      ETH_DATA_FORMAT
    )
    .then((value) => {
      limit = value;
    });
  // Creating the transaction object
  const tx = {
    from: signer.address,
    to: "0x1328b19533f4bA383c5721845c4478b3B4e7A388",
    value: web3.utils.toWei("0.001", "ether"), // change AMOUNT to send
    gas: limit,
    nonce: await web3.eth.getTransactionCount(signer.address),
    maxPriorityFeePerGas: web3.utils.toWei("2", "gwei"),

    maxFeePerGas: maxFeePerGas.toString(),
    chainId: 11155111,
    type: 0x2,
  };
  signedTx = await web3.eth.accounts.signTransaction(tx, signer.privateKey);
  console.log("Raw transaction data: " + signedTx.rawTransaction);
  // Sending the transaction to the network
  const receipt = await web3.eth
    .sendSignedTransaction(signedTx.rawTransaction)
    .once("transactionHash", (txhash) => {
      console.log(`Mining transaction ...`);
      console.log(`https://sepolia.etherscan.io/tx/${txhash}`);
    });
  // The transaction is now on chain!
  console.log(`Mined in block ${receipt.blockNumber}`);
}
main();
```

```
const { Web3 } = require("web3");
const web3 = new Web3(
  "https://mainnet.infura.io/v3/9ef65ae1fe6c4c68b0a842493dfadeba"
);
// web3.eth.getBlockNumber().then(console.log);

const ganacheUrl = "HTTP://127.0.0.1:7545";
web3.eth.net
  .getId()
  .then((networkId) => {
    console.log("Connected to network ID:", networkId);
  })
  .catch((error) => {
    console.log("Connected to network ID:", networkId);
  })
  .catch((error) => {
    console.error("Error connecting to Ganache:", error);
  });
const accountAddress = "0xbc14dDeCD661d9de02ba1320d0C6204eB0BC160F";
web3.eth.getBalance(accountAddress).then((balance) => {
    console.log(
      "Account balance:",
      web3.utils.fromWei(balance, "ether"),
      "ETH"
    );
  })
  .catch((error) => {
    console.error("Error fetching balance:", error);
  });
```

Using Web3.js to Interact with Smart Contracts.

- Make sure you have web3 installed and install truffle and verify installation.
  Install command:

  ```
  npm install -g truffle
  ```

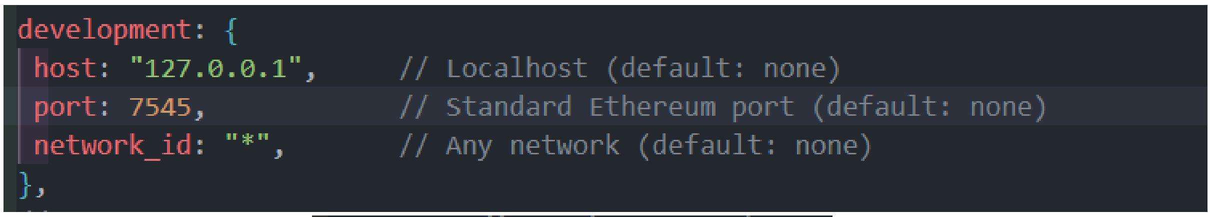
  check:

  ```
  truffle version
  ```

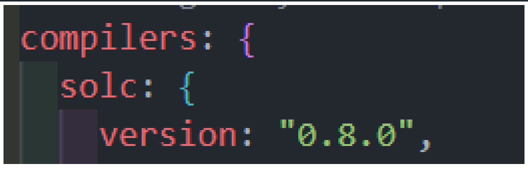
- Create a new folder and initialize truffule

  ```
  mkdir MyTruffleProject
  cd MyTruffleProject
  truffle init
  ```

- We get to see truffle-config.js
  In that uncomment the following part and make the required changes



```
development: {
  host: "127.0.0.1",     // Localhost (default: none)
  port: 7545,            // Standard Ethereum port (default: none)
  network_id: "*",       // Any network (default: none)
},
```



```
compilers: {
  solc: {
    version: "0.8.0",
```

- Create a solidity file which will be your contract and put in MyTruffleProject/contracts

  SimpleStorage.sol:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 public storedData;

    function set(uint256 x) public {
        storedData = x;
    }

    function get() public view returns (uint256) {
        return storedData;
    }
}
```

- Compile the file using the following command

  `sudo truffle compile` for windows `truffle compile`

- In MyTruffleProject/migrations create a file “2_deploy_contracts.js” with following code.

  2_deploy_contracts.js:

  ```
  const SimpleStorage = artifacts.require("SimpleStorage");

  module.exports = function (deployer) {
    deployer.deploy(SimpleStorage);
  };
  ```
- Deploy contracts using the following command in parent directory

  ```
  truffle migrate --network development
  ```
- Now create a ‘.js’ file in MyTruffleProject Folder with the following code

  testContract.js :

  ```
  const {Web3} = require('web3');
  const web3 = new Web3('http://127.0.0.1:7545');  // Ganache RPC server address


  // Get the contract ABI and address from the build files

  const contractABI = /* ABI generated by Truffle */;

  const contractAddress = /* Deployed contract address */;


  const simpleStorage = new web3.eth.Contract(contractABI, contractAddress);


  // Interact with the contract

  async function interactWithContract() {

    const accounts = await web3.eth.getAccounts();

    const receipt = await simpleStorage.methods.set(42).send({ from: accounts[0] });

    console.log('Transaction receipt:', receipt);


    const value = await simpleStorage.methods.get().call();

    console.log('Stored value:', value);

  }

  interactWithContract();
  ```
- You can find the ABI in MyTruffleProject/Build/SimpleStorage.json
- Change the contractAddress to the address you got while compiled in quotes
- Run using the command `node testContract.js`

```rust
fn main(){
    print!("hello");
}
```

```rust

fn main()
{
    let x:i32 = 5;
    let y:f32 = 6.14;
    let z:char = 'a';
    let a:bool = true;
    let b:bool = false;
    let name:&str = "Lokesh";
    println!("Integer:{}",x);
    println!("Float:{}",y);
    println!("Character:{}",z);
    println!("Boolean:{}",a);
    println!("Boolean:{}",b);
    println!("String:{}",name);
}
```

```rust
fn main()
{
    let age:i32 = 20;
    let height:f32 = 5.9;
    let name:&str = "Lokesh";

    println!("Age:{},name:{},height:{}",age,name,height);
    println!("Binary: {:b}, Hex: {:x}, Octal: {:o}", age, age, age);
}
```

```rust
use std::io;

fn main() {
    // Prompt the user for the first number
    println!("Enter the first number:");
    let mut first_number = String::new();
    io::stdin().read_line(&mut first_number).expect("Failed to read line");
    let first_number: f64 = first_number.trim().parse().expect("Invalid input");

    // Prompt the user for the second number
    println!("Enter the second number:");
    let mut second_number = String::new();
    io::stdin().read_line(&mut second_number).expect("Failed to read line");
    let second_number: f64 = second_number.trim().parse().expect("Invalid
input");

    // Perform arithmetic operations
    let sum = first_number + second_number;
    let difference = first_number - second_number;
    let product = first_number * second_number;
    let quotient = first_number / second_number;

    // Display the results
    println!("Sum: {}", sum);
    println!("Difference: {}", difference);
    println!("Product: {}", product);
    println!("Quotient: {}", quotient);
}
```

```
fn main() {
    let a: u8 = 0b1100;
    let b: u8 = 0b1010;

    println!("Bitwise AND: {:08b}", a & b);
    println!("Bitwise OR: {:08b}", a | b);
    println!("Bitwise XOR: {:08b}", a ^ b);
    println!("Bitwise NOT: {:08b}", !a);
    println!("Left shift: {:08b}", a << 1);
    println!("Right shift: {:08b}", a >> 1);

    let x = true;
    let y = false;

    println!("Logical AND: {}", x && y);
    println!("Logical OR: {}", x || y);
    println!("Logical NOT: {}", !x);
}
```

```
fn main()
{
    let mut x:i32 = 5;
    let mut y:i32 = 6;

    println!("before swapping x={},y = {}",x,y);

    x = x + y;
    y = x-y;
    x = x - y;
    println!("After swapping x={},y = {}",x,y);
}
```

```rust
use std::io;

#[derive(Debug)]
struct Person {
    name: String,
    age: u32,
}

fn main() {
    // Tuple components
    let mut int_input = String::new();
    let mut float_input = String::new();
    let mut string_input = String::new();
    let mut arr = [0; 3];
    let mut name_input = String::new();
    let mut age_input = String::new();

    // Input for integer
    println!("Enter an integer:");
    io::stdin().read_line(&mut int_input).expect("Failed to read line");
    let int_value: i32 = int_input.trim().parse().expect("Please enter a valid
integer");

    // Input for float
    println!("Enter a float:");
    io::stdin().read_line(&mut float_input).expect("Failed to read line");
    let float_value: f64 = float_input.trim().parse().expect("Please enter a
valid float");

    // Input for array
    println!("Enter 3 integers for the array:");
    for i in 0..3 {
        let mut input = String::new();
        io::stdin().read_line(&mut input).expect("Failed to read line");
        arr[i] = input.trim().parse().expect("Please enter a valid number");
    }

    // Input for string
    println!("Enter a string:");
    io::stdin().read_line(&mut string_input).expect("Failed to read line");
    let string_value = string_input.trim().to_string();

    // Input for struct
    println!("Enter a name for the struct:");
    io::stdin().read_line(&mut name_input).expect("Failed to read line");
    let name = name_input.trim().to_string();
    println!("Enter an age for the struct:");
    io::stdin().read_line(&mut age_input).expect("Failed to read line");
    let age: u32 = age_input.trim().parse().expect("Please enter a valid age");

    let person = Person { name, age };

    // Create the tuple
    let tuple = (int_value, float_value, arr, string_value, person);

    // Print the tuple
    println!("Tuple: {:?}", tuple);
    println!("Struct inside tuple: Name = {}, Age = {}", tuple.4.name,
tuple.4.age);
}
```

```rust
fn main() {
    // Example of a loop
    let mut count = 0;
    loop {
        println!("Count: {}", count);
        count += 1;
        if count >= 5 {
            break;
        }
    }

    // Example of a while loop
    let mut num = 0;
    while num < 3 {
        println!("Number: {}", num);
        num += 1;
    }

    // Example of a for loop
    let numbers = [1, 2, 3, 4, 5];
    for number in numbers.iter() {
        println!("Number: {}", number);
    }

    // Example of a conditional loop
    let mut i = 0;
    loop {
        if i % 2 == 0 {
            println!("Even number: {}", i);
        } else {
            println!("Odd number: {}", i);
        }
        i += 1;
        if i >= 5 {
            break;
        }
    }
}
```