BLOCKCHAIN TECHNOLOGIES

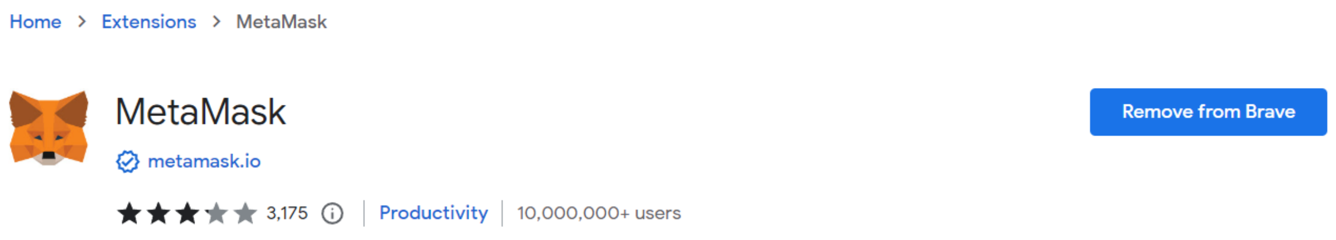
Lab Record

WEEK-1

A) Generate Public private key pairs for Bitcoin and Ethereum addresses.
B) Connect to the Public/Test net Ethereum Blockchain network using popular wallets (MetaMask, Brave browser) and understand various terminologies like gas, gas fee, gas price, priority fee.

Procedure :

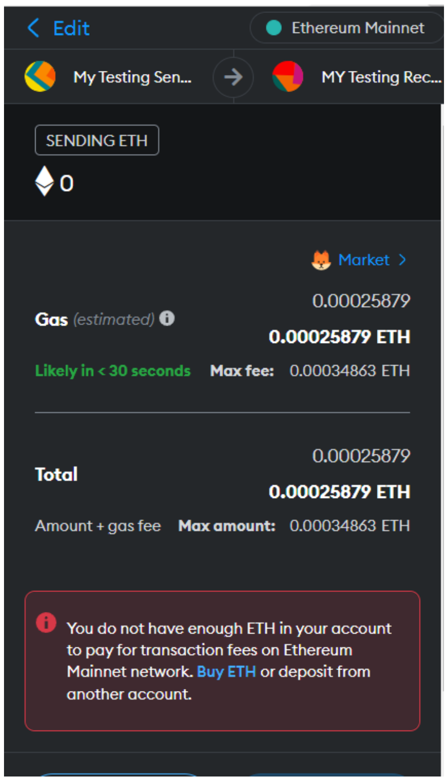
- **Download Brave Browser:** Begin by downloading the Brave Browser, a privacy-focused web browser known for its enhanced security features. This browser will serve as the platform for accessing and utilizing the MetaMask extension.
- **Install MetaMask Extension:** Navigate to the official MetaMask website at https://metamask.io/ using the Brave Browser. Download and install the MetaMask Chrome extension from the provided link. This extension enables seamless interaction with Ethereum-based applications and facilitates the creation and management of Ethereum wallets.
- **Create a MetaMask Account:** Upon successful installation of the MetaMask extension, open it and follow the on-screen instructions to set up a MetaMask account. This involves generating a unique password and securely storing it. Once the account is created, a public key is automatically generated, which is essential for receiving Ethereum transactions.
- **Locate Public Key:** After logging in to your MetaMask account, the public key associated with your Ethereum wallet will be visible within the extension window. This public key, represented by a hexadecimal string (e.g., 0x2B9e0CbB6737615f2749fc3638a8bAD048f4977c), is used for receiving cryptocurrency funds and verifying transactions.
- **Access Private Key:** To view the private key of your Ethereum wallet, navigate to the 'Account Details' section within the MetaMask extension. From there, choose the option to 'Show Private Key' and enter your account password for authentication. The private key is a sensitive piece of information that provides full control over the associated Ethereum wallet. Exercise caution and ensure its secure storage.



C) Send test ether from one account to another.

Procedure :

- **Access MetaMask Extension:** Launch the MetaMask extension, which was previously installed as described in the earlier sections of this lab record. Ensure that you are logged into your MetaMask account.
- **Initiate Transaction:** Within the MetaMask interface, locate and select the 'Send' option. This action signifies your intention to initiate an Ethereum transaction.
- **Enter Recipient Address:** In the subsequent window, input the recipient's Ethereum address, also referred to as the public key. This address is the destination for the test ether transfer and ensures that the cryptocurrency reaches the intended recipient.
- **Provide Transaction Details**: Within the following window, provide specific transaction details. This includes specifying the amount of ether to be transferred and setting the gas price. The gas price determines the fee associated with processing the transaction on the Ethereum network.
- **Proceed to Next Step:** Once transaction details are entered accurately, proceed by clicking the 'Next' button. This advances you to the next stage of the transaction process.
- **Confirm Transaction Details:** Review the transaction details presented on the confirmation screen. Ensure that all information, including the recipient address, transaction amount, and gas price, is correct.
- **Confirm Transaction:** With confidence in the accuracy of the provided details, click on the 'Confirm' button. This action confirms your intent to execute the Ethereum transaction.
- **Finalize Transaction:** MetaMask will initiate the transaction process, broadcasting it to the Ethereum network. Wait for the network to process the transaction. A confirmation notification will indicate the successful transfer of the test ether.



0

Gas (estimated) 0.00025879
0.00025879 ETH
Likely in < 30 seconds Max fee: 0.00034863 ETH

Total 0.00025879
0.00025879 ETH
Amount + gas fee Max amount: 0.00034863 ETH

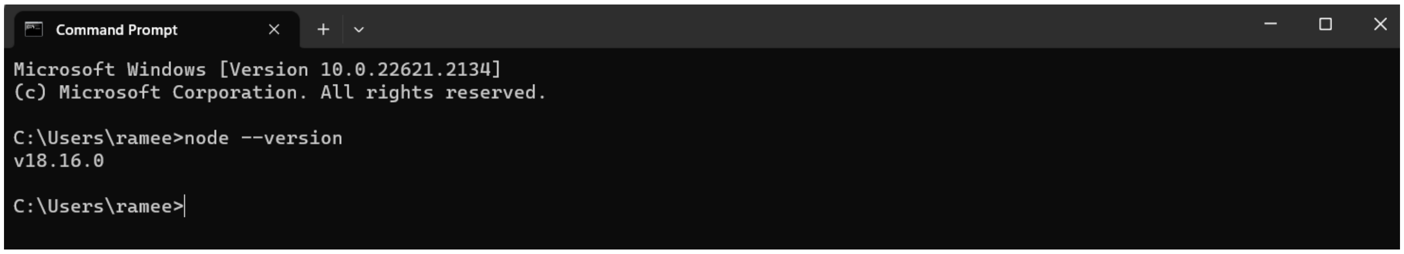
You do not have enough ETH in your account to pay for transaction fees on Ethereum Mainnet network. Buy ETH or deposit from another account.

WEEK-2

Installation and Configuration of Node.js and Web3.js

Procedure :

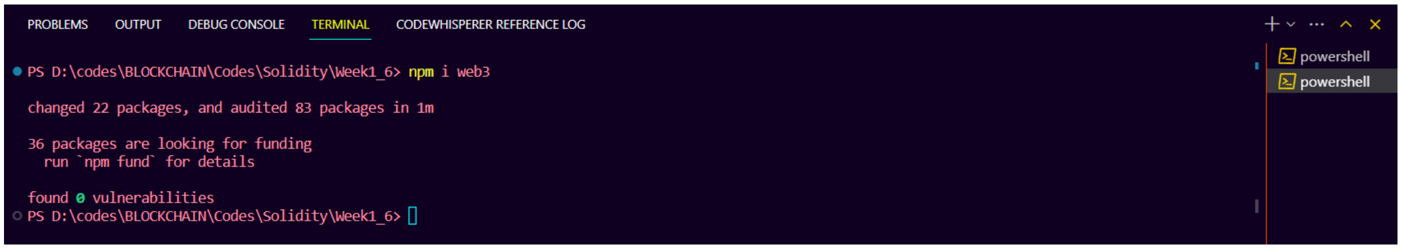
- **Install Node.js:** Download and install Node.js from https://nodejs.org/.
- **Open Terminal:** Access your terminal or command prompt.
- **Install Web3.js:** Run npm install web3 to install the Web3.js library.
- **Create File:** Create a JavaScript file, e.g., ethereum.js, in a text editor.
- **Initialize Web3.js:** In ethereum.js, use const Web3 = require('web3'); const web3 = new Web3('YOUR_NODE_URL'); to set up Web3.js.
- **Interact with Ethereum:** Write code to interact with Ethereum, like web3.eth.getBlockNumber().then(blockNumber => console.log('Block:', blockNumber));.
- **Save File:** Save ethereum.js.
- **Terminal:** Navigate to the file's directory in the terminal.
- **Run Script:** Execute node ethereum.js to run the script.
- **Explore Ethereum:** Your script now interacts with Ethereum using Node.js and Web3.js.



```
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ramee>node --version
v18.16.0

C:\Users\ramee>
```



```
PS D:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6> npm i web3

changed 22 packages, and audited 83 packages in 1m

36 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6>
```

WEEK–3

Using Web3.js to Transfer Ether from one account to another .

Code:

```
const { Web3 } = require("web3");
const web3 = new Web3("HTTP://127.0.0.1:7545");

// Private key of the sender's Ethereum account
const senderPrivateKey =
    "0x28c5f4f67ed61e5ef7c6c712b9e6b62deb94dd5f3684b27b81e9d3883a04206f";
// Sender's Ethereum address
const senderAddress = "0xD2D473Ac80Df9c59B7F8817920Ee8D06e1a6f581";

// Recipient's Ethereum address
const recipientAddress = "0x0474c3D5241D703f573F764A0Ee94A335e582e01";
// Replace with the recipient"s Ethereum address
// Transfer Ether function
async function transferEther() {
    // Get the current gas price
    const gasPrice = await web3.eth.getGasPrice();
    // Estimate the gas required for the transaction
    const gasEstimate = await web3.eth.estimateGas({
        from: senderAddress,
        to: recipientAddress,
        value: web3.utils.toWei("0.1", "ether"), // Amount of Ether to send (in this
example, 0.1 Ether)
    });
    // Create the raw transaction object
    const rawTx = {
        from: senderAddress,
        to: recipientAddress,
        value: web3.utils.toHex(web3.utils.toWei("0.1", "ether")), // Amount of
        // Ether to send, converted to Wei
        gasPrice: web3.utils.toHex(gasPrice),
        gasLimit: web3.utils.toHex(gasEstimate),
    };
    // Sign the transaction with the sender"s private key
    const signedTx = await web3.eth.accounts.signTransaction(
        rawTx,
        senderPrivateKey
    );
    // Send the signed transaction to the Ethereum network
    const txReceipt = await web3.eth.sendSignedTransaction(
        signedTx.rawTransaction
    );
    console.log(
        "Transaction successful. Transaction hash:",
        txReceipt.transactionHash
    );
}

// Call the transferEther function to initiate the transaction
transferEther();
```

Output :

```
NOTE: web3.js is running without provider. You need to pass a provider in order to interact with the network!
d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\utils\get_transaction_error.js:48
            error = new web3_errors_1.TransactionRevertInstructionError(_reason, undefined, transactionReceiptFormatted);
TransactionRevertInstructionError: Transaction has been reverted by the EVM
    at d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\utils\get_transaction_error.js:48:21
    at Generator.next (<anonymous>)
    at d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\utils\get_transaction_error.js:24:71
    at new Promise (<anonymous>)
    at __awaiter (d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\utils\get_transaction_error.js:20:12)
    at getTransactionError (d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\utils\get_transaction_error.js:33:12)
    at Object.<anonymous> (d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\rpc_method_wrappers.js:441:100)
    at Generator.next (<anonymous>)
    at fulfilled (d:\codes\BLOCKCHAIN\Codes\Solidity\Week1_6\node_modules\web3-eth\lib\commonjs\rpc_method_wrappers.js:21:58)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {
  innerError: undefined,
  reason: 'VM Exception while processing transaction: insufficient balance',
  signature: undefined,
  receipt: undefined,
  data: undefined,
  code: 402
}
Node.js v18.16.0
```

WEEK-4

Create a Private Ethereum Blockchain network.

Procedure :

- **Install Ganache:** Download and install Ganache from https://www.trufflesuite.com/ganache.
- **Open Ganache:** Launch Ganache and note the local blockchain URL (e.g., http://127.0.0.1:7545).
- **Create File:** Open a text editor and create transfer.js.
- **Install Web3.js:** In the terminal, navigate to the file's directory and run npm install web3.
- **Write Transfer Code:** In transfer.js, use Web3.js to transfer Ether.
- **Run Script:** In the terminal, navigate to the directory and run node transfer.js.
- **Transaction Execution:** Observe the transaction hash output.

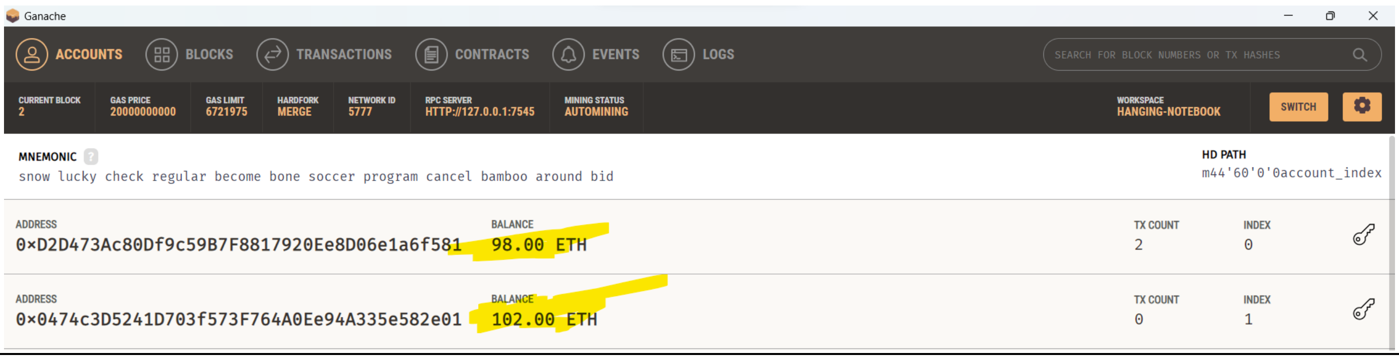
```javascript
const {Web3} = require('web3');
const web3 = new Web3('http://127.0.0.1:7545'); // Ganache default URL

async function sendEther() {
    try {
        const accounts = await web3.eth.getAccounts();
        const sender = accounts[0];
        const receiver = accounts[1];
        const amountToSend = web3.utils.toWei('1', 'ether'); // Convert 1 Ether to Wei
        const tx = {
            from: sender,
            to: receiver,
            value: amountToSend
        };

        const transactionHash = await web3.eth.sendTransaction(tx);
        console.log(`Transaction successful. Transaction hash: ${transactionHash}`);
    } catch (error) {
        console.error(`Error sending transaction: ${error}`);
    }
}

sendEther();
```

Output :



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER             MINING STATUS   WORKSPACE
2              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545  AUTOMINING      HANGING-NOTEBOOK

MNEMONIC                                                                          HD PATH
snow lucky check regular become bone soccer program cancel bamboo around bid     m44'60'0'0account_index

ADDRESS                                       BALANCE                          TX COUNT   INDEX
0×D2D473Ac80Df9c59B7F8817920Ee8D06e1a6f581    98.00 ETH                        2          0

ADDRESS                                       BALANCE                          TX COUNT   INDEX
0×0474c3D5241D703f573F764A0Ee94A335e582e01    102.00 ETH                       0          1
```

WEEK-9

A. Write a program to display the statements.

Code:

```rust
fn main() {
    // Print a single line statement
    println!("Hello, world!");

    // Print multiple statements
    println!("This is a Rust program.");
    println!("It demonstrates printing statements.");
}
```

Output:

```
Hello, world!

This is a Rust program.

It demonstrates printing statements.
```

B. Write a program to demonstrate the basic data types in Rust.

Code:

```rust
fn main() {
    let integer: i32 = 42;
    let float: f32 = 3.14;
    let character: char = 'A';
    let boolean: bool = true;
    let name: &str = "Rameez";  // Use &str for string literals

    println!("Integer: {}", integer);
    println!("Float: {}", float);
    println!("Character: {}", character);
    println!("Boolean: {}", boolean);
    println!("Name: {}", name);  // Corrected the println! format string
}
```

Output:

```
Integer: 42

Float: 3.14

Character: A

Boolean: true

Name: Rameez
```

C. Write a program to format strings and numbers.

Code:

```rust
fn main() {
    let integer: i32 = 42;
    let float: f64 = 3.14159265;
    let character: char = 'A';
    let boolean: bool = true;

    // Format integer with leading zeros and specify minimum width
    println!("Integer (with leading zeros and width 5): {:05}", integer);

    // Format float with precision
    println!("Float (with precision 2): {:.2}", float);

    // Format character as is
    println!("Character: {}", character);

    // Format boolean as lowercase
    println!("Boolean (in lowercase): {:?}", boolean);


    println!(
        "Integer: {:05}, Float: {:.2}, Character: {}, Boolean: {}",
        integer, float, character, boolean
    );

    let my_string = "Hello, Rust!";

    // Using indexing
    let slice = &my_string[0..5]; // Slice from index 0 (inclusive) to 5 (exclusive)
    println!("Slice: {}", slice);

    // You can also omit the start index
    let slice_from_start = &my_string[..5]; // Slice from the beginning to 5 (exclusive)
    println!("Slice from start: {}", slice_from_start);

    // Omitting the end index slices to the end
    let slice_to_end = &my_string[7..]; // Slice from index 7 (inclusive) to the end
    println!("Slice to end: {}", slice_to_end);
}



fn again() {
    // Format a string with placeholders
    let name = "Alice";
    let age = 30;
    let formatted_string = format!("Hello, {}! You are {} years old.", name, age);
    println!("{}", formatted_string);

    // Format numbers
    let pi = 3.14159265;
    let formatted_float = format!("Pi: {:.2}", pi); // Format as a floating-point number with 2 decimal places
    println!("{}", formatted_float);

    let number = 42;
    let formatted_binary = format!("Binary: {:b}", number); // Format as binary
    let formatted_hex = format!("Hexadecimal: {:x}", number); // Format as hexadecimal
    println!("{}", formatted_binary);
    println!("{}", formatted_hex);
}
```

Output:

```
Integer (with leading zeros and width 5): 00042
Float (with precision 2): 3.14
Character: A
Boolean (in lowercase): true
Integer: 00042, Float: 3.14, Character: A, Boolean: true
Slice: Hello
Slice from start: Hello
Slice to end: Rust!
```

D. Write a program to compute arithmetic operations taking input from the user and display the result.

Code:

```
use std::io;

fn main() {
    println!("Enter two numbers: ");

    let mut input = String::new();

    io::stdin().read_line(&mut input).expect("Failed to read line");

    let numbers: Vec<i32> = input
        .split_whitespace()
        .map(|num| num.parse().unwrap())
        .collect();
    let sum = numbers[0] + numbers[1];
    let difference = numbers[0] - numbers[1];
    let product = numbers[0] * numbers[1];
    let quotient = numbers[0] as f64 / numbers[1] as f64;

    println!("Sum: {}", sum);
    println!("Difference: {}", difference);
    println!("Product: {}", product);
    println!("Quotient: {:.2}", quotient);
}
```

Output:

```
Enter two numbers:

12 12

Sum: 24

Difference: 0

Product: 144

Quotient: 1.00
```

E. Write a program to demonstrate bitwise and logical operators.

Code:

```rust
fn main() {
    let a = 0b1010; // Binary representation of 10
    let b = 0b1100; // Binary representation of 12

    let bitwise_and = a & b;
    let bitwise_or = a | b;
    let bitwise_xor = a ^ b;
    let logical_and = a > 0 && b > 0;
    let logical_or = a > 0 || b > 0;

    println!("Bitwise AND: {:04b}", bitwise_and);
    println!("Bitwise OR: {:04b}", bitwise_or);
    println!("Bitwise XOR: {:04b}", bitwise_xor);
    println!("Logical AND: {}", logical_and);
    println!("Logical OR: {}", logical_or);
}
```

Output:

```
Bitwise AND: 1000

Bitwise OR: 1110

Bitwise XOR: 0110

Logical AND: true

Logical OR: true
```

F. Write a program to swap two numbers without using a temporary variable.

Code:

```rust
fn main() {
    let mut num1 = 5;
    let mut num2 = 10;

    println!("Before swapping: num1 = {}, num2 = {}", num1, num2);

    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    println!("After swapping: num1 = {}, num2 = {}", num1, num2);
}
```

Output:

```
Before swapping: num1 = 5, num2 = 10
After swapping: num1 = 10, num2 = 5
```

WEEK-10

Write programs that demonstrate the Compound Data Types in Rust (Arrays, Tuples)

Code:

```rust
fn main() {
    // Arrays
    let numbers: [i32; 5] = [1, 2, 3, 4, 5];

    // Tuples
    let person: (String, i32, f64) = ("Alice".to_string(), 25, 5.8);

    println!("Array: {:?}", numbers);
    println!("Tuple: {:?}", person);
}
```

Output:

```
Array: [1, 2, 3, 4, 5]
Tuple: ("Alice", 25, 5.8)
```

WEEK-11

Write programs that can demonstrate the working of loops and Conditional Loops in Rust

Code:

```rust
fn main() {
    // For Loop
    for i in 1..=5 {
        println!("For loop: {}", i);
    }

    // While Loop
    let mut counter = 0;
    while counter < 5 {
        println!("While loop: {}", counter);
        counter += 1;
    }

    // Conditional loop (loop with break)
    let mut value = 0;
    loop {
        println!("Value: {}", value);
        if value >= 5 {
            break;
        }
        value += 1;
    }
}
```

Output:

```
For loop: 1
For loop: 2
For loop: 3
For loop: 4
For loop: 5
While loop: 0
While loop: 1
While loop: 2
While loop: 3
While loop: 4
Value: 0
Value: 1
Value: 2
Value: 3
Value: 4
Value: 5
```

WEEK-12

Write a program that can demonstrate

A)Assigning value of one variable to another variable.

B)Passing value to a function.

C)Returning value from a function.

Code:

```
// Function that returns the square of a number
fn square(x: i32) -> i32 {
    return x * x
}

fn main() {
    // Assigning value of one variable to another
    let num1 = 5;
    let num2 = num1;

    // Passing value to a function
    let result = square(num2);

    // Returning value from a function
    println!("Square of {} is {}", num2, result);
}
```

Output:

```
Square of 5 is 25
```

WEEK-13

Write a program to generate a Random number using Rust

Code:

```rust
use rand::Rng;
use std::io;


fn week_13() {
    let random_number = rand::thread_rng().gen_range(1..=100);
    println!("Random number: {}", random_number);
}

fn main() {
    week_13();
}
```

Output:

```
Random number: 37
```

WEEK-14

Write a program that can compare the guessed number with a Secret generated number

Code:

```rust
use rand::Rng;
use std::io;
fn week_14() {
    let secret_number = rand::thread_rng().gen_range(1..=100);

    loop {
        println!("Guess the number (1-100): ");
        let mut guess = String::new();
        io::stdin().read_line(&mut guess).expect("Failed to read line");


        let guess: i32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => {
                println!("Please enter a valid number.");
                continue;
            }
        };


        if guess == secret_number {
            println!("Congratulations! You guessed the number.");
            break;
        } else if guess < secret_number {
            println!("Try a higher number.");
        } else {
            println!("Try a lower number.");
        }
    }
}


fn main() {
    week_14();
}
```

Output:

```
Guess the number (1-100):
50
Try a lower number.
Guess the number (1-100):
25
Try a higher number.
Guess the number (1-100):
37
Try a higher number.
```

```
Guess the number (1-100):
45
Try a lower number.
Guess the number (1-100):
40
Try a lower number.
Guess the number (1-100):
38
Congratulations! You guessed the number.
```