

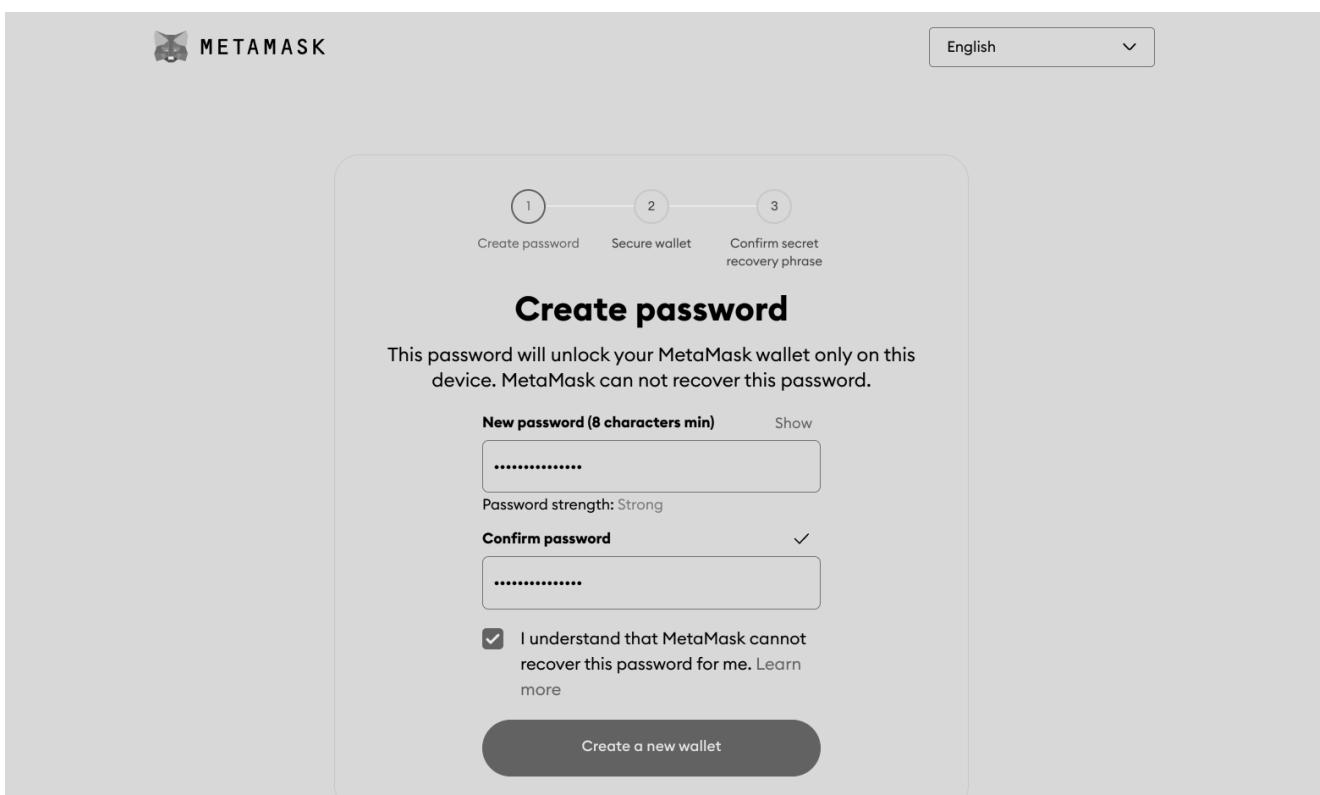
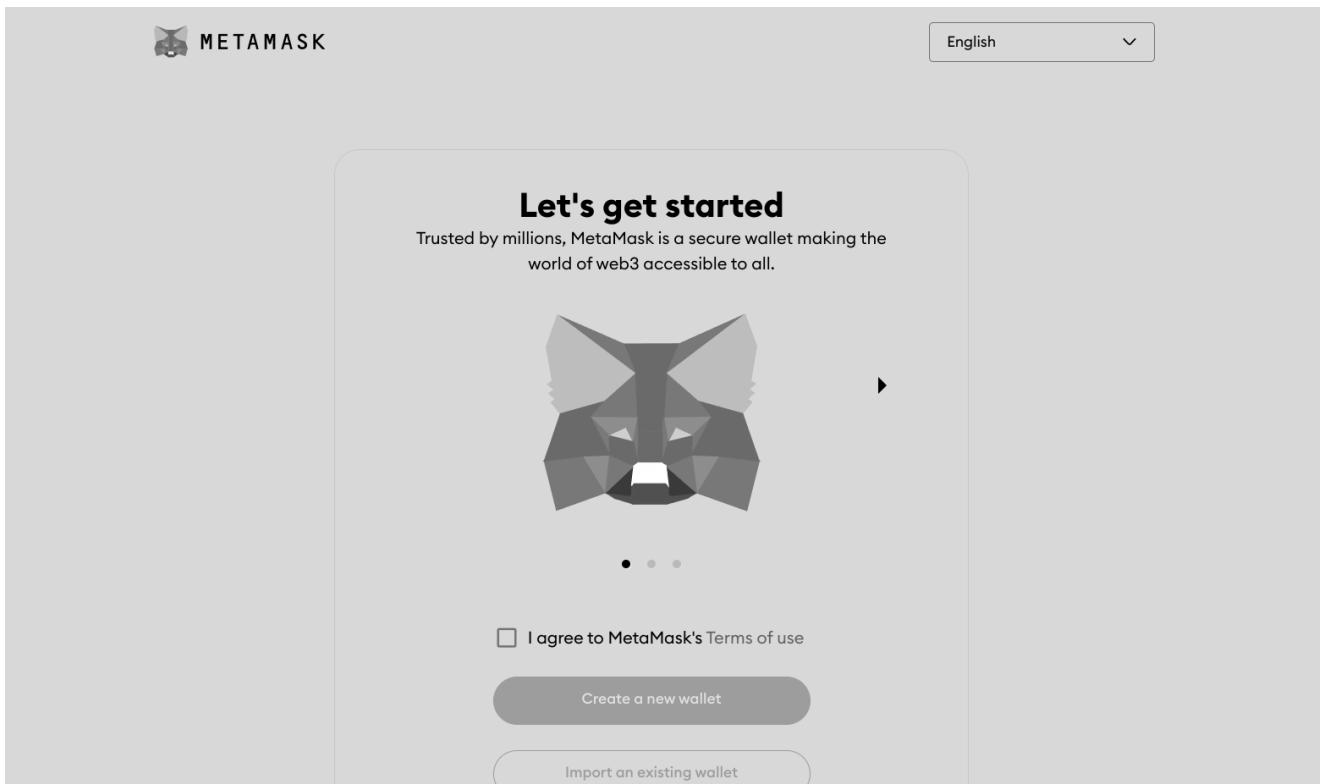
BCT Practical 1: Metamask Homepage

The screenshot shows the official Metamask website. At the top, there's a navigation bar with the Metamask logo, a search bar, and links for "Features", "Build", "Resources", and "Learn". There's also a language switcher set to "EN" and a toggle switch for dark mode. The main content area features a large heading "Install MetaMask for your browser" above a wireframe-style visualization of the Metamask UI. This visualization includes a central window showing an account balance of "0.1655 ETH" and a value of "\$317.26 USD", with buttons for "Buy", "Send", "Swap", "Bridge", and "Portfolio". Below this are tabs for "Tokens", "NFTs", and "Activity", and a summary section for "Ethereum" showing "0.1655 ETH" and "\$317.26 USD". At the bottom of the visualization is a prominent "Install MetaMask for Chrome" button.

Metamask Installation and Create Wallet Screen

The screenshot shows the "Let's get started" screen of the Metamask application. At the top left is the Metamask logo and a language selector set to "English". The main title "Let's get started" is displayed in bold, followed by a subtitle: "Trusted by millions, MetaMask is a secure wallet making the world of web3 accessible to all." Below the text is a large, stylized fox head graphic. A progress bar at the bottom indicates the user is on step 1 of 3. At the bottom of the screen are two buttons: "Create a new wallet" and "Import an existing wallet".

BCT Practical 2:



1 2 3

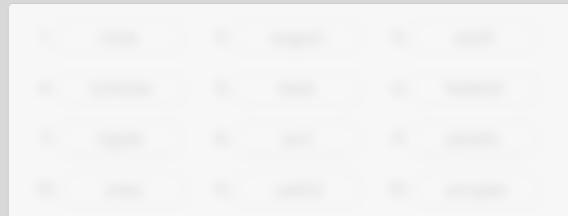
Create password Secure wallet Confirm secret recovery phrase

Write down your Secret Recovery Phrase

Write down this 12-word Secret Recovery Phrase and save it in a place that you trust and only you can access.

Tips:

- Write down and store in multiple secret places
- Store in a safe deposit box



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](#)

BCT Practical 3: bank.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract Bank {
    string public name = "Deno Bank";
    string public symbol = "DB";
    uint256 public totalSupply;

    // Mapping to store the balances of each address
    mapping(address => uint256) public balanceOf;

    // Mapping to keep track of allowances
    mapping(address => mapping(address => uint256)) public allowance;

    // Constructor to initialize the total supply and assign it to the contract
    // deployer
    constructor(uint256 initialSupply) {
        totalSupply = initialSupply;
        balanceOf[msg.sender] = initialSupply;
    }

    // Function to transfer tokens from the sender to another address
    function transfer(address toWhom, uint256 value) public returns (bool
success) {
        require(balanceOf[msg.sender] >= value, "Insufficient Balance");
        balanceOf[msg.sender] = balanceOf[msg.sender] - value;
        balanceOf[toWhom] = balanceOf[toWhom] + value;
        return true;
    }

    // Function to approve an address to spend a certain amount on behalf of
    // the owner
    function approve(address spender, uint256 value) public returns (bool
success) {
        allowance[msg.sender][spender] = value;
        return true;
    }
}
```

```

// Function to transfer tokens from one account to another using an allowance
function transferFrom(address from, address to, uint256 value) public
returns (bool success) {
    require(balanceOf[from] >= value, "Insufficient Balance");
    require(allowance[from][msg.sender] >= value, "Allowance exceeded");

    // Adjust balances
    balanceOf[from] -= value;
    balanceOf[to] += value;

    // Adjust the allowance
    allowance[from][msg.sender] -= value;

    return true;
}

// Rename this function to avoid conflict with the `allowance` mapping
function checkAllowance(address owner, address spender) public view returns
(uint256 remaining) {
    return allowance[owner][spender];
}

```

Output:

Deployed Contracts 1

BANK AT 0X878...0154F (MEMORY)

Balance: 0 ETH

approve	address spender, uint256 value
transfer	address toWhom, uint256 value
transferFrom	address from, address to, uint256 value
allowance	address , address
balanceOf	address
checkAllowance	address owner, address spender
name	
symbol	
totalSupply	

BCT Practical 4 : bankcontract.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract BankContract {
    struct ClientAccount {
        int256 clientId; // Use int256 for better compatibility
        address clientAddress;
        uint256 clientBalanceInEther;
    }

    ClientAccount[] public clients; // Public visibility for client list
    int256 public clientCounter; // Make this public to allow external access
    address payable public manager; // Declare manager as payable
    mapping(address => uint256) public interestDate; // Keep track of interest
dates

    modifier onlyManager() {
        require(msg.sender == manager, "Only manager can call this!");
        _;
    }

    modifier onlyClients() {
        bool isClient = false;
        for (uint256 i = 0; i < clients.length; i++) {
            if (clients[i].clientAddress == msg.sender) {
                isClient = true;
                break;
            }
        }
        require(isClient, "Only clients can call this!");
        _;
    }

    constructor() {
        clientCounter = 0; // Initialize the counter
        manager = payable(msg.sender); // Set the creator as manager
    }

    receive() external payable {} // Allow contract to receive Ether

    function setManager(address payable managerAddress) public onlyManager
returns (string memory) {
        manager = managerAddress;
        return "Manager updated successfully!";
    }
}
```

```

function joinAsClient() public payable returns (string memory) {
    require(msg.value > 0, "You must send some Ether to join."); // Ensure
a deposit
    interestDate[msg.sender] = block.timestamp; // Store the current time
    clients.push(ClientAccount(clientCounter++, msg.sender, msg.value));
    return "You have successfully joined as a client.";
}

function deposit() public payable onlyClients {
    require(msg.value > 0, "You must send some Ether to deposit.");
    // Update the client's balance in the client array
    for (uint256 i = 0; i < clients.length; i++) {
        if (clients[i].clientAddress == msg.sender) {
            clients[i].clientBalanceInEther += msg.value;
            break;
        }
    }
}

function withdraw(uint256 amount) public onlyClients {
    require(amount <= address(this).balance, "Insufficient contract
balance.");
    require(amount <= getClientBalance(msg.sender), "Withdraw amount
exceeds client balance.");
    payable(msg.sender).transfer(amount); // Transfer the requested amount
to the client
}

function sendInterest() public onlyManager {
    for (uint256 i = 0; i < clients.length; i++) {
        address initialAddress = clients[i].clientAddress;
        uint256 lastInterestDate = interestDate[initialAddress];

        require(block.timestamp >= lastInterestDate + 10 seconds, "It's
just been less than 10 seconds since the last interest was sent!");

        payable(initialAddress).transfer(1 ether); // Send 1 Ether as
interest
        interestDate[initialAddress] = block.timestamp; // Update the last
interest date
    }
}

```

```
function getClientBalance(address client) public view returns (uint256) {
    for (uint256 i = 0; i < clients.length; i++) {
        if (clients[i].clientAddress == client) {
            return clients[i].clientBalanceInEther;
        }
    }
    return 0; // Return 0 if client not found
}

function getContractBalance() public view returns (uint256) {
    return address(this).balance; // Return the contract's balance
}
```