Blockchain Experiment 6

Aim: Creating Smart Contracts and performing transactions using Solidity and Remix IDE

Theory:

Smart Contract:

A smart contract is a self-executing digital contract with the terms of the agreement directly written into code. These contracts run on blockchain technology and automatically execute and enforce the terms and conditions agreed upon by the parties involved. Smart contracts eliminate the need for intermediaries, such as lawyers or notaries, as the contract's code enforces the rules, and the blockchain ensures transparency, security, and immutability of the contract.

Smart contracts have the following key characteristics:

1. Self-executing: Once certain conditions are met, the contract executes automatically without the need for third-party involvement.

2. Tamper-proof: Once deployed on a blockchain, a smart contract's code and execution are immutable, making it resistant to fraud or manipulation.

3. Transparency: Smart contract code and transaction history are publicly accessible on the blockchain, enhancing transparency.

4. Security: The decentralized nature of blockchain provides security against single points of failure or attacks.

5. Efficiency: Smart contracts can streamline processes by automating tasks and removing intermediaries.

Significance of Smart Contracts in Ethereum Blockchain:

1. Decentralization: Ethereum's blockchain, which supports smart contracts, is decentralized, meaning no single entity has control. This reduces the risk of manipulation, censorship, or fraud.

2. Security: Ethereum's smart contracts are highly secure due to cryptographic principles and the immutability of the blockchain. This security is particularly significant in financial, legal, and other critical applications.

3. Trustless Transactions: Smart contracts enable trustless interactions. Parties do not need to trust each other, as the contract's code enforces the agreement.

4. Automated Execution: Smart contracts execute automatically when predefined conditions are met. This automation reduces the risk of human error and ensures that the terms of the contract are followed precisely.

5. Cost Reduction:By eliminating intermediaries, smart contracts can significantly reduce transaction costs. This is especially relevant in sectors like finance, where fees for intermediaries can be high.

6. Accessibility:Ethereum's smart contracts are accessible to anyone with an internet connection, democratizing access to financial and legal services.

7. Diverse Use Cases: Smart contracts have applications beyond finance, including supply chain management, voting systems, real estate transactions, and more. Ethereum's flexibility allows developers to create a wide range of smart contract-based applications.

8. Innovation: Ethereum's smart contract platform has spurred innovation in the blockchain and decentralized application (DApp) space. Many new projects and solutions have been built on the Ethereum platform, leading to a vibrant ecosystem.

Smart contracts are a fundamental building block of Ethereum's ecosystem, and their significance lies in their ability to automate, secure, and transparently enforce a wide range of agreements and transactions. They have the potential to revolutionize various industries and the way we conduct business.

Implementation:

The case study topic i.e Impact of Blockchain on Freelance and gig economy is chosen .

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;

contract FreelanceContract {
    address public client;
    address public freelancer;
    uint public totalPayment;
    uint public projectCompletionDate;
```

```
bool public projectCompleted;
string public projectDescription;

struct WorkRecord {
    address client;
    address freelancer;
    string description;
    bool completed;
}

WorkRecord[] public workRecords;

struct FreelancerProfile {
    address freelancer;
    string profileData;
}

mapping(address => FreelancerProfile) public freelancerProfiles;
mapping(address => mapping(address => uint)) public clientReviews;
mapping(address => uint) public intellectualAssets;

constructor(address _freelancer, string memory _description) {
    client = msg.sender;
    freelancer = _freelancer;
    totalPayment = 0;
    projectCompletionDate = 0;
    projectCompleted = false;
    projectDescription = _description;
}

modifier onlyClient() {
    require(msg.sender == client, "Only the client can call this function");
    _;
}

modifier onlyFreelancer() {
    require(msg.sender == freelancer, "Only the freelancer can call this function");
    _;
}
```

```
function fundContract() public payable onlyClient {
    totalPayment += msg.value;
}

function startProject(uint _daysToComplete) public onlyClient {
    require(!projectCompleted, "Project is already completed.");
    projectCompletionDate = block.timestamp + _daysToComplete * 1 days;
}

function completeProject() public onlyFreelancer {
    require(!projectCompleted, "Project is already completed.");
    require(block.timestamp >= projectCompletionDate, "Project not yet due for completion.");
    projectCompleted = true;

    // Record the completed work
    workRecords.push(WorkRecord({
        client: client,
        freelancer: freelancer,
        description: projectDescription,
        completed: true
    }));
}

function releasePayment() public onlyClient {
    require(projectCompleted, "Project is not completed yet.");
    require(totalPayment > 0, "No funds available for release.");
    payable(freelancer).transfer(totalPayment);
    totalPayment = 0;
}

function addFreelancerProfile(string memory _profileData) public onlyFreelancer {
    freelancerProfiles[freelancer] = FreelancerProfile({
        freelancer: freelancer,
        profileData: _profileData
    });
}

function addClientReview(address _freelancer, uint _rating) public onlyClient {
    require(_rating >= 1 && _rating <= 5, "Rating must be between 1 and 5.");
    clientReviews[_freelancer][client] = _rating;
```

```
    }

    function addIntellectualAsset(uint _value) public {
        require(msg.sender == client || msg.sender == freelancer, "Only client or freelancer can add
            intellectual assets.");
        intellectualAssets[msg.sender] += _value;
    }
}
```