Smart Contract Execution – Quick Guidelines

1. Open Remix IDE. (https://remix.ethereum.org)
2. Set Solidity compiler version to **^0.8.0**.
3. Create a new file and paste the smart contract code.
4. Compile the contract using the **Solidity Compiler**.
5. Use **JavaScript VM** as the environment for simple testing.
6. Click **Deploy** under the Deploy & Run section.
7. Test functions using the deployed contract interface.

✅3. Store and Retrieve Username

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

contract UsernameStorage {
    mapping(address => string) private usernames;

    function setUsername(string memory _username) public {
        usernames[msg.sender] = _username;
    }

    function getUsername() public view returns (string memory) {
        return usernames[msg.sender];
    }
}
```

✅4. Student Marks and Average

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

contract StudentMarks {
    uint[] private marks;

    function addMarks(uint _mark) public {
        marks.push(_mark);
    }
    function getAverageMarks() public view returns (uint) {
        uint sum = 0;
        for (uint i = 0; i < marks.length; i++) {
            sum += marks[i];
        }
        return marks.length > 0 ? sum / marks.length : 0;
    }
}
```

✅5. Basic Voting System

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

contract Voting {
    mapping(string => uint) public votes;
    mapping(address => bool) public hasVoted;
    string[] public candidates;

    constructor(string[] memory _candidates) {
        candidates = _candidates;
    }

    function vote(string memory _candidate) public {
        require(!hasVoted[msg.sender], "Already voted");
        votes[_candidate]++;
        hasVoted[msg.sender] = true;
    }
}
```

✅6. Bank Deposit and Withdrawal

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

contract SimpleBank {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint _amount) public {
        require(balances[msg.sender] >= _amount, "Insufficient funds");
        balances[msg.sender] -= _amount;
        payable(msg.sender).transfer(_amount);
    }
    function checkBalance() public view returns (uint) {
        return balances[msg.sender];
    }
}
```

✅7. Parking Slot Booking

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

contract ParkingBooking {
    mapping(uint => address) public slotOwner;

    function bookSlot(uint slotNumber) public {
        require(slotOwner[slotNumber] == address(0), "Already booked");
        slotOwner[slotNumber] = msg.sender;
    }
}
```

✅8. Timelock Contract

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

contract Timelock {
    uint public unlockTime;
    address public owner;

    constructor(uint _delay) payable {
        owner = msg.sender;
        unlockTime = block.timestamp + _delay;
    }

    function withdraw() public {
        require(msg.sender == owner, "Not owner");
        require(block.timestamp >= unlockTime, "Locked");
        payable(owner).transfer(address(this).balance);
    }
}
```

✅9. Event Registration

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

```
contract EventRegistration {
    address public owner;
    uint public fee;
    address[] public participants;

    constructor(uint _fee) {
        owner = msg.sender;
        fee = _fee;
    }

    function register() public payable {
        require(msg.value == fee, "Incorrect fee");
        participants.push(msg.sender);
    }

    function getParticipants() public view returns (address[] memory) {
        return participants;
    }
}
```

✅10. Employee Attendance

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

contract Attendance {
    mapping(address => uint) public lastMarked;

    function markAttendance() public {
        require(block.timestamp > lastMarked[msg.sender] + 1 days, "Already
marked");
        lastMarked[msg.sender] = block.timestamp;
    }
}
```

✅11. Library Management

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

contract LibrarySystem {
    struct Book {
        string title;
        bool isBorrowed;
    }

    Book[] public books;

    function addBook(string memory _title) public {
        books.push(Book(_title, false));
    }

    function borrowBook(uint index) public {
        require(!books[index].isBorrowed, "Already borrowed");
        books[index].isBorrowed = true;
    }

    function returnBook(uint index) public {
        books[index].isBorrowed = false;
    }
}
```

✅12. Subscription Access

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

contract SubscriptionAccess {
    mapping(address => uint) public subscriptions;
    uint public fee = 1 ether;

    function subscribe() public payable {
        require(msg.value == fee, "Incorrect fee");
        subscriptions[msg.sender] = block.timestamp + 30 days;
    }

    function isSubscribed() public view returns (bool) {
        return block.timestamp < subscriptions[msg.sender];
    }
}
```

✅13. Product Warranty

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

contract WarrantyManager {
    struct Product {
        uint purchaseDate;
        uint warrantyPeriod; // in days
    }

    mapping(uint => Product) public products;

    function registerProduct(uint id, uint _warrantyDays) public {
        products[id] = Product(block.timestamp, _warrantyDays);
    }
    function isValid(uint id) public view returns (bool) {
        Product memory p = products[id];
        return block.timestamp <= p.purchaseDate + (p.warrantyPeriod * 1
days);
    }
}
```

✓14. Lucky Draw

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

contract LuckyDraw {
    address[] public players;
    address public winner;

    function enterDraw() public payable {
        require(msg.value == 0.01 ether, "Entry fee is 0.01 ETH");
        players.push(msg.sender);
    }

    function pickWinner() public {
        require(players.length > 0, "No players");
        uint index = uint(keccak256(abi.encodePacked(block.timestamp,
block.difficulty))) % players.length;
        winner = players[index];
        payable(winner).transfer(address(this).balance);
        delete players;
    }
}
```

✓15. Doctor Appointment Booking

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

contract AppointmentBooking {
    mapping(uint => address) public slots; // slot ID to patient

    function bookSlot(uint slotId) public {
        require(slots[slotId] == address(0), "Slot booked");
        slots[slotId] = msg.sender;
    }

    function getSlotPatient(uint slotId) public view returns (address) {
        return slots[slotId];
    }
}
```