```
const SHA256 = require("crypto-js/sha256");
class Block {
constructor(index, timestamp, data, previousHash = "") {
this.index = index;
this.timestamp = timestamp;
this.data = data;
this.previousHash = previousHash;
this.hash = this.calculateHash();
}
calculateHash() {
return SHA256(
this.index +
this.previousHash +
this.timestamp +
JSON.stringify(this.data)
).toString();
}
}
class Blockchain {
constructor() {
this.chain = [this.createGenesisBlock()];
}
createGenesisBlock() {
return new Block(0, "21/04/2023", "Genesis Block", "0");
}
getLatestBlock() {
return this.chain[this.chain.length - 1];
}
addBlock(newBlock) {
newBlock.previousHash = this.getLatestBlock().hash;

newBlock.hash = newBlock.calculateHash();
this.chain.push(newBlock);
}
isChainValid() {
for (let i = 1; i < this.chain.length; i +) {
const currentBlock = this.chain[i];
const previousBlock = this.chain[i - 1];
if (currentBlock.hash = currentBlock.calculateHash()) {
return false;
}
if (currentBlock.previousHash = previousBlock.hash) {
return false;
}
```

```
}
return true;
}
}
/Blockchain Implementation
let myCoin = new Blockchain();
myCoin.addBlock(new Block(1, "22/04/2023", { amount: 4 }));
myCoin.addBlock(new Block(2, "22/04/2023", { amount: 8 }));
/console.log('Is blockchain valid? ' + myCoin.isChainValid());
console.log(JSON.stringify(myCoin, null, 4));
```

Assembly

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
contract InlineAssembly {
// Defining function
function add(uint256 a) public view returns (uint256 b) {
assembly {
let c := add(a, 16)
mstore(0x80, c)
{
let d := add(sload(c), 12)
b := d
}
b := add(b, c)
}
}
}
```

using_library.sol Code
←**—---Exponent—-->**

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
import "contracts/myLIB.sol";
contract UseLib {
function getsum(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.sum(x, y);
}
```

```
function getexponent(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.exponent(x, y);
}
}
```

/— error handling —/

```
contract ErrorDemo {
function getSum(uint256 a, uint256 b) public pure returns (uint256) {
uint256 sum = a + b;
// require(sum < 255, "Invalid");
assert(sum<255);
return sum;
}
}
```

—event—--

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0;
// Creating a contract
contract eventExample {
// Declaring state variables
uint256 public value = 0;
// Declaring an event
event Increment(address owner);
// Defining a function for logging event
function getValue(uint256 _a, uint256 _b) public {
emit Increment(msg.sender);
value = _a + _b;
}
}
```

Abstract

```
pragma solidity ^0.5.17;
contract Calculator {
function getResult() external view returns (uint256);
}
contract Test is Calculator {
constructor() public {}
function getResult() external view returns (uint256) {
uint256 a = 1;
uint256 b = 2;
uint256 result = a + b;
return result;
```

```
}
}
```

C) Libraries, Assembly, Events, Error handling.

1) Libraries

myLib.sol Code

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
library myMathLib {
function sum(uint256 a, uint256 b) public pure returns (uint256) {
return a + b;
}
function exponent(uint256 a, uint256 b) public pure returns (uint256) {
return a**b;
}
}
```

using_library.sol Code

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
import "contracts/myLIB.sol";
contract UseLib {
function getsum(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.sum(x, y);
}
function getexponent(uint256 x, uint256 y) public pure returns (uint256) {
return myMathLib.exponent(x, y);
}
}
```

2) **Assembly**

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;
contract InlineAssembly {
// Defining function
function add(uint256 a) public view returns (uint256 b) {
assembly {
let c := add(a, 16)
mstore(0x80, c)
{
let d := add(sload(c), 12)
b := d
}
```

```
b := add(b, c)
}
}
}
```

3) Events

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0;
// Creating a contract
contract eventExample {
// Declaring state variables
uint256 public value = 0;
// Declaring an event
event Increment(address owner);
// Defining a function for logging event
function getValue(uint256 _a, uint256 _b) public {
emit Increment(msg.sender);
value = _a + _b;
}
}
```

Restricted Access

```
/SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
contract RestrictedAccess {
address public owner = msg.sender;
uint256 public creationTime = block.timestamp;
modifier onlyBy(address _account) {
require(msg.sender == _account, "Sender not authorized!");
_;
}
modifier onlyAfter(uint256 _time) {
require(block.timestamp >= _time, "Function was called too early!");
_;
}
modifier costs(uint256 _amount) {
require(msg.value >= _amount, "Not enough Ether provided!");
_;
}
function forceOwnerChange(address _newOwner)
```

```
public
payable
costs(200 ether)
{
owner = _newOwner;
}
function changeOwner(address _owner) public onlyBy(owner) {
owner = _owner;
}
function disown() public onlyBy(owner) onlyAfter(creationTime + 3 weeks) {
delete owner;
}
}
```

A) **Withdrawal Pattern, Restricted Access**

1) Withdrawal Pattern

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18;
contract WithdrawalPattern {
address public owner;
uint256 public lockedbalance;
uint256 public withdrawablebalance;
constructor() {
owner = msg.sender;
}
modifier onlyowner() {
require(msg.sender == owner, "Only the owner can call this function");
_;
}
function deposit(uint256 amount) public payable {
require(amount > 0, "Amount must be greater than zero");
lockedbalance += amount;
}
function withdraw(uint256 amount) public payable onlyowner {
require(
amount <= withdrawablebalance,
"Insufficient withdrawable balance"
);
withdrawablebalance -= amount;
payable(msg.sender).transfer(amount);
}
function unlock(uint256 amount) public onlyowner {
```

```
require(amount <= lockedbalance, "Insufficient locked balance");
lockedbalance -= amount;
withdrawablebalance += amount;
}
}
```

Function modifiers

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.5.0;
contract ExampleContract {
address public owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
uint256 public counter;
modifier onlyowner() {
require(msg.sender == owner, "Only the contract owner can call");
_;
}
function incrementcounter() public onlyowner {
counter++;
}
}
```

4. **Cryptographic Functions**

```
pragma solidity ^0.5.0;
contract Test{
function callKeccak256() public pure returns(bytes32 result){
return keccak256("BLOCKCHAIN");
}
function callsha256() public pure returns(bytes32 result){
return sha256("BLOCKCHAIN");
}
function callripemd() public pure returns (bytes20 result){
return ripemd160("BLOCKCHAIN");
}
}
```

2. Pure Functions

```
pragma solidity ^0.5.0;
contract pure_demo {
function getResult() public pure returns (uint256 product, uint256 sum) {
uint256 num1 = 2;
uint256 num2 = 4;
product = num1 * num2;
```

```
sum = num1 + num2;
}
}
```

11.Ether Units

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract SolidityTest {
function convert_Amount_to_Wei(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 wei;
}
function convert_Amount_To_Ether(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 ether;
}
function convert_Amount_To_Gwei(uint256 Amount)
public
pure
returns (uint256)
{
return Amount * 1 gwei;
}
function convert_seconds_To_mins(uint256 _seconds)
public
pure
returns (uint256)
{
return _seconds / 60;
}
function convert_seconds_To_Hours(uint256 _seconds)
public
pure
returns (uint256)
{
return _seconds / 3600;
```

```
}
function convert_Mins_To_Seconds(uint256 _mins)
public
pure
returns (uint256)
{
return _mins * 60;
}
}
```

9. Mappings

```
pragma solidity ^0.5.0;
contract LedgerBalance {
mapping(address => uint256) public balances;
function updateBalance(uint256 newBalance) public {
balances[msg.sender] = newBalance;
}
}
contract Updater {
function updateBalance() public returns (uint256) {
LedgerBalance ledgerBalance = new LedgerBalance();
return ledgerBalance.balances(address(this));
}
}
```

7. **Enums**

```
pragma solidity ^0.5.0;
contract enumdemo {
enum week_days {
Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday,
Sunday
}
week_days week;
week_days choice;
week_days constant default_value = week_days.Sunday;
function set_value() public {
choice = week_days.Tuesday;
```

```
}
function get_choice() public view returns (week_days) {
return choice;
}
function get_defaultvalue() public view returns (week_days) {
return default_value;
}
}
```