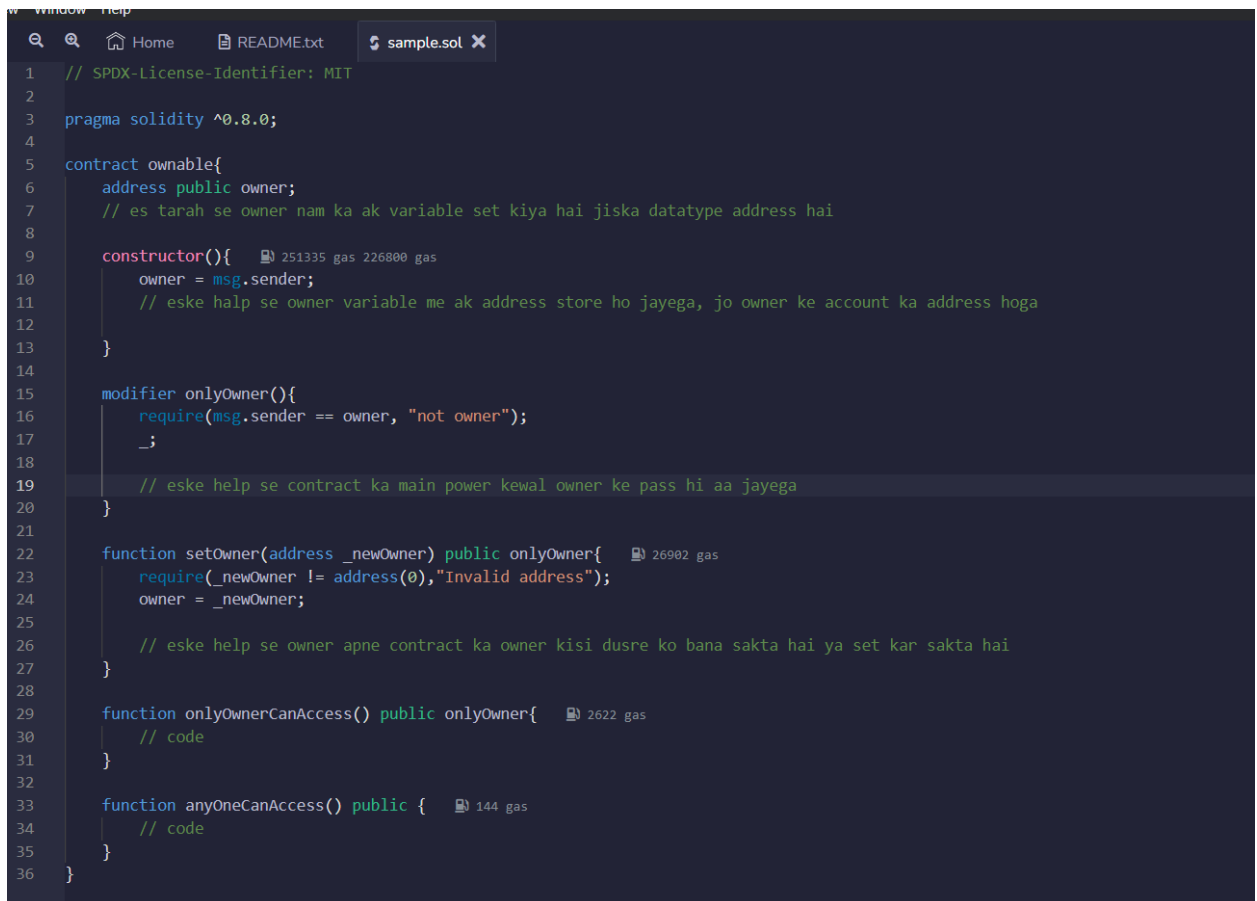


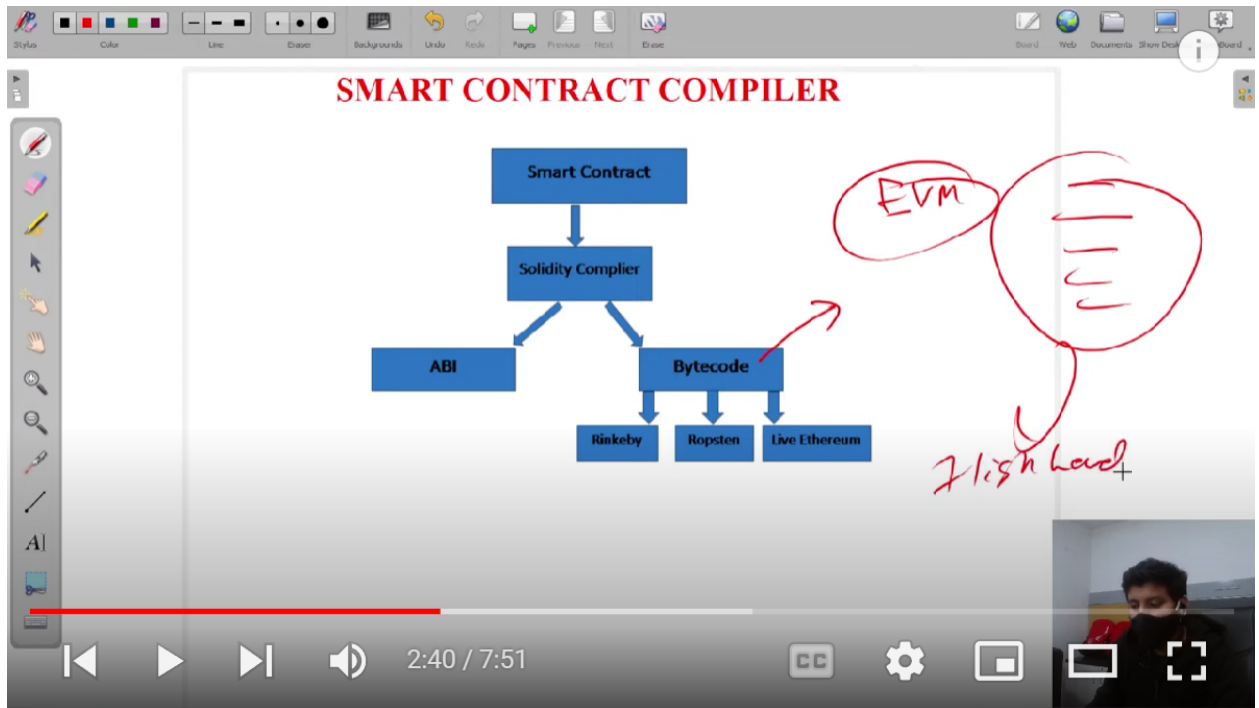
SOLIDITY

- Overview of Sample Smart Contract



```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 contract ownable{
6     address public owner;
7     // es tarah se owner nam ka ak variable set kiya hai jiska datatype address hai
8
9     constructor(){ 251335 gas 226800 gas
10         owner = msg.sender;
11         // eske help se owner variable me ak address store ho jayega, jo owner ke account ka address hoga
12     }
13
14     modifier onlyOwner(){
15         require(msg.sender == owner, "not owner");
16         _;
17     }
18     // eske help se contract ka main power kewal owner ke pass hi aa jayega
19 }
20
21
22 function setOwner(address _newOwner) public onlyOwner{ 26902 gas
23     require(_newOwner != address(0), "Invalid address");
24     owner = _newOwner;
25
26     // eske help se owner apne contract ka owner kisi dusre ko bana sakta hai ya set kar sakta hai
27 }
28
29 function onlyOwnerCanAccess() public onlyOwner{ 2622 gas
30     // code
31 }
32
33 function anyOneCanAccess() public { 144 gas
34     // code
35 }
36 }
```

- Smart Contract Compiler
 - ABI : eska use communication ke liye use kiya jata hai, ye Account se contract se communication karwata hai
 - Aur ak contract se dusre contract se communication karwata hai
 - Bytecode: ye code EVM per deploy hota hai



- DataTypes : values and references

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 // data types : values and references
6 // values: bool, int, address, bytes etc
7 // references: Array etc
8
9 contract valueType{
10     bool public boolTemp = true;
11     uint public temp = 243; // eska use hamesha gas ke lenden me use hota hai, eska range: 0 to 2^256 - 1
12     int public temp2 = 143; // eska range -2^255 to 2^255 - 1
13     address public addr = 0xAb8483F64d9C6d1EcF9b849Ae677d3315835cb2;
14     // esme 20 bits tak data store ho sakta hai; esme data hexadecimal ke form me data store hota hai
15     bytes32 public b3; // string ke jagah per eska use krte hai, ye bhi hexadecimal ko hi store krta hai
16 }
17
18
19 // 0xd9145CCE52D386f254917e481eB44e9943F39138 ye valueType contract ka address hai
```

- Function in Solidity

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract functionIntro{
6     uint age = 20;
7     function add(uint _x, uint _y) public pure returns(uint){ infinite gas
8         return _x + _y;
9     }
10
11     function changeAge() public{ infinite gas
12         age+= 1;
13     }
14
15     function getage() public view returns(uint){ 2459 gas
16         return age;
17     }
18
19     function fun() internal{ infinite gas
20         // code
21     }
22 }
23
24 function dummy(uint _x) pure returns(uint){
25     return _x*10;
26     // ye function contract ke bahr likha gya hai; eski visibility bydefault internal hoti hai
27     // jo function contract ke bahr likha rahta hi unki visibility change nahi kar sakte hai
28     // es function ka use hum apne contract me import karne ke liye use kar sakte hai
29 }
```

- State Variable

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract stateVariable{
6     // state variable contract ke ander aur function ke bahr define hota hai
7     // state variable sidha blockchain and contract storage per store hota hai joki costly hai
8
9     uint public salary;
10
11     constructor(){ 63807 gas 41600 gas
12         salary=100;
13         // constructor ak special function hai jo apne aap hi call ho jata hai
14     }
15
16     function setSalary() public{ 22258 gas
17         salary = 2000;
18     }
19 }
```

- Local Variable

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract localVariable{
6     // jo variable contract ke kisi function ke ander define hota hai wah local variable hota hai
7     // local variable memory me store hota hai i.e RAM me store hota hai
8     // local variable ke liye negligible cost i.e gas lagta hai
9     // local variable execution ke bad memory se delete ho jata hai
10
11     uint public age;
12     bool public b;
13     address public newAdd;
14
15     function fun(uint _x, bool _y, address _z) public returns(uint,bool){ infinite gas
16         uint i = 29;
17         bool b1 = true;
18         i+= 29;
19         b1 = false;
20
21
22         age = _x;
23         b = _y;
24         newAdd = _z;
25
26         return (i,b1);
27     }
28 }
```

- Global Variable

The video player shows a list of Solidity global variables and their descriptions. A red bracket highlights the first 10 items:

- `blockhash(uint blockNumber) returns (bytes32)` : hash of the given block when `blocknumber` is one of the 256 most recent blocks; otherwise returns zero
- `block.basefee (uint)` : current block's base fee (EIP-3198 and EIP-1559)
- `block.chainid (uint)` : current chain id
- `block.coinbase (address payable)` : current block miner's address
- `block.difficulty (uint)` : current block difficulty
- `block.gaslimit (uint)` : current block gaslimit
- `block.number (uint)` : current block number
- `block.timestamp (uint)` : current block timestamp as seconds since unix epoch
- `gasleft() returns (uint256)` : remaining gas
- `msg.data (bytes calldata)` : complete calldata
- `msg.sender (address)` : sender of the message (current call)
- `msg.sig (bytes4)` : first four bytes of the calldata (i.e. function identifier)
- `msg.value (uint)` : number of wei sent with the message
- `tx.gasprice (uint)` : gas price of the transaction
- `tx.origin (address)` : origin of the transaction (full call chain)

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract global_Variable{
6     // These are spacial variable which exist in global workspace and provide
7     // information about the blockchain and transaction properties.
8     // ye predefine variable hota hai e.g msg.sender
9     // msg.sender --> ye account ko detail ko deta hai jis account se deploy huaa rahta hai
10    // msg.sender --> ye us account ke detail ko bhi deta hai jo mere smart contract se communicate karna chahta hai
11    // msg.sender --> jab koi smart contract ke function call hota us function ka bhi information deta hai
12
13    address public myadd = msg.sender;
14    uint public time = block.timestamp;
15    uint public diff = block.difficulty;
16    uint public gascost = tx.gasprice;
17 }
```

- View, Pure and Simple Function

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract view_pure_simple_function{
6     // view --> ye read only karta hai;
7     // ye blockchain ke ander, statevariable, global variable ko kewal read karta hai usko change nahi karta hai i.e write nahi karta hai
8
9     // pure --> ye na to read karta hai aur nahi write karta hai
10
11    // simple function --> ye state variable ko change kar sakte hai
12
13
14    uint public age = 20;
15    function viewFunc() public view returns(uint){ 2481 gas
16        return age;
17    }
18
19    function pureFunc1() public pure returns(uint){ 315 gas
20        return 1;
21    }
22
23    function pureFunc2(uint _x) public pure returns(uint){ infinite gas
24        return _x;
25    }
26
27    function simpleFunc() public{ infinite gas
28        age = age + 10;
29        // yaha per age ka value read bhi ho raha hai aur age ka value write bhi ho raha hai
30    }
31
32    function dummy() public view returns(uint){ infinite gas
33        return age+10;
34        // yaha per kewal view type function age ko read kar raha hai , us age me koi change nahi kar raha hai;
35        // ye kewal age ko read karke usme 10 ko add kiya hai kewal
36    }
37 }
```

- Default Values

```
stateVariable.sol  localVar.sol  README.txt  global_Variable.sol  view_pure_simple_function.sol  default_Values.sol X
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract default_Values{
6     uint public x;
7     bool public y;
8     address public z;
9     bytes32 public a;
10    string public str;
11 }
```

- Strings

```
e.sol  localVar.sol  README.txt  global_Variable.sol  view_pure_simple_function.sol  default_Values.sol  string.sol 1 X
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract strings{
6     // String by default blockchain i.e contract storage me store hota hai , chahe local variable ho ya state variable ho
7     // esliye string data type ko jab local variable banate hai to memory keyword ka use karna padta hai
8
9     string public str = "gaurav"; // str state variable hai
10    function checkstring() public pure{ 213 gas
11        string memory name = "aman";
12
13        // es function ko return nahi kar rahe hai esliye button click karne per dikh nahi raha hai
14    }
15
16    function checkstring2() public pure returns(string memory){ infinite gas
17        string memory name2 = "Nirbhay Gupta";
18        return name2;
19
20        // es function ko return kar rahe hai esliye button per click karne per dikh raha hai
21    }
22
23    function checkstring3(string memory _str) public pure returns(string memory){ infinite gas
24        string memory name3 = _str;
25        return name3;
26    }
27 }
28
```

- Constants

```
new Window Help  ionIntro.sol  stateVariable.sol  localVar.sol  README.txt  global_Variable.sol  view_pure_simple_function.sol  default_Values.sol  string.sol  constants.sol X
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract constants{
6     // jab kisi variable ka value change n karna ho to constant keyword ka use karte hai
7     // jab kisi contract ke owner ko change nahi karne ho to us case me constant keyword ka use kar sakte hai
8     // jab state variable bnate hai to uska gas cost jyada hota hai usi gas cost ko kam karne aur efficiency ko badhane ke liye constant keyword ka use karte hai
9     address public constant owneradd = 0x5B38D0a6a701c568545dcFcb03FcB875f56beddC4;
10    address public owneradd2 = 0x5B38D0a6a701c568545dcFcb03FcB875f56beddC4;
11
12 }
```

- Constructor in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract blockchainconstructor{
6     // ak contract me kewal ak hi constructor ho sakta hai
7     // constructor ak tarah ka function hota hai, ye aisa function hai jab smart contract create hota hai to ye sabse pahel call hone wala function hai
8     // eska use smart contract ke owner ko set karne ke liye kiya jata hai ya state variable ko initialize karne ke liye use kiya jata hai
9     // state variable ko initialize karne ke liye tin tarike hote hai
10    // pahle state variable banate time hi initialize kar sakte hai, dusra constructor ke doura, tisra function ke doura
11
12    uint public age;
13    address public owner;
14    string public name;
15
16    constructor(string memory _name, uint _age, address _owner){
17        // age = 20;
18        age = _age;
19        owner = _owner;
20        name = _name;
21
22        // jab constructor me as argument koi value lete hai to ye deploy ke time hi es value ko dena padega as input, nahi to smart contract deploy nahi hoga
23    }
24 }
25
26
```

- If-else in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract if_elseBlockchain{
6
7     // if else ka use contract level per use nahi kar sakte hai
8
9     function fun(uint _x) public pure returns(string memory){
10         string memory val;
11
12         if(_x>100){
13             val = "greater than 100";
14         }
15         else if(_x<100){
16             val = "smaller than 100";
17         }
18         else{
19             val = "equal to 100";
20         }
21
22         return val;
23     }
24 }
25
```

- Ternary Operator in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract ternaryOperator{
6     function fun(uint _x) public pure returns(string memory){
7         string memory val;
8
9         val = _x>100 ? "greater than 100" : "equal or smaller than 100";
10        return val;
11    }
12 }
```

- Loops in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract loopBlockchain{
6     // loop bhi contract level per kam nahi karte hai, ye function level per kam karte hai
7
8     function loop() public pure returns(uint){
9         uint count = 0;
10        for(uint i=0; i<10; i++){
11            count = count+5;
12        }
13        // return count;
14
15        uint j=0;
16        while(j<10){
17            count = count+10;
18            j++;
19        }
20
21        // return count;
22
23        do{
24            count = count+10;
25            j++;
26        }while(j<10);
27
28        return count;
29    }
30
31 }
```

- Continue and Break Keyword in Blockchain


```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract continueBreakKeyword{
6     function loop() public pure returns(uint){ infinite gas
7         uint count = 0;
8         // continue and break ka use if else ke bahr bhi use kar sakte hai
9
10        for(uint i=0; i<10; i++){
11            if(i==8){
12                continue;
13            }
14            count = count+5;
15        }
16
17        for(uint i=0; i<10; i++){
18            if(i==6){
19                break;
20            }
21            count = count+5;
22        }
23
24
25        return count;
26    }
27 }
```

- Fixed Size Array in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract fixedSizeArray{
6     uint[5] public arr = [10,20,30,40,50];
7     uint[5] public arr1;
8
9     uint[5] arr2;
10    constructor(){ infinite gas 294400 gas
11        arr2 = [10,20,30,40,50];
12    }
13
14    function returnArray() public view returns(uint[5] memory){ infinite gas
15        return arr2;
16        // es tarah se kisi function se kisi array ko pass kar sakte hai
17        // array return karne per gas kaphi jyada lagta hai esliye array ko return nahi karte hai
18    }
19
20    // yadi hame es array ka length aur es array ke kisi element ko update karna ho ya delete karna ho ya get karna ho ye sab kisi function ke ander hi kar sakte hai,
21    // ese contract ke ander aur function ke bahr nahi kar sakte hai
22    // contract ke ander kewal state variable, function, and modifier hi bana sakte hai
23    // array bhi string ki tarah by default storage me store hota hai
24
25 }
```

```

26 function array() public view returns(uint){ 2580 gas
27     // get
28     uint temp;
29     temp = arr[3];
30
31     // update
32     // arr[2] = 3000;
33
34     // delete
35     // delete arr[4];
36
37     // length
38     uint len = arr.length;
39     return len;
40
41 }
42
43 function arrayWithLoop(uint _x) public { infinite gas
44     for(uint i=0; i<arr.length; i++){
45         arr[i] = _x+1;
46     }
47 }
48
49 function createArray() public pure returns(uint){ infinite gas
50     uint[] memory arrMem = new uint[](3);
51     // memory ke ander kewal fixed size array hi bana sakte hai, Dynamic array nahi bana sakte hai
52     // es tarah se kisi function ke ander array create karte hai
53
54     arrMem[0] = 33;
55     arrMem[1] = 35;
56     arrMem[2] = 40;
57
58     return arrMem[0];
59     // array return karne per gas kafi jyada lagta hai esliye array ko return nahi karte hai
60     // return arrMem;
61 }

```

- Dynamic Size Array in Blockchain

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract dynamicSizeArray{
6     // Dynamic array memory ke ander nahi ho sakte hai
7     uint[] public arr = [10,299,3432,434,34,32,1];
8
9     function returnArray() public view returns(uint[] memory){ infinite gas
10         return arr;
11     }
12
13     function fun() public{ 113787 gas
14         // get
15         uint temp = arr[3];
16
17         // update
18         arr[4] = 222; // [10,299,3432,434,222,32,1]
19
20         // delete
21         delete arr[1]; // [10,0,3432,434,222,32,1]
22
23         // length
24         uint len = arr.length;
25
26         // push
27         arr.push(111); // [10,0,3432,434,34,32,1,111]
28
29         // pop
30         arr.pop(); // ye array ke last element ko pop krta hai
31     }
32 }
33

```

- Bytes in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract bytesInBlockchain{
6     // 1 byte = 8bits
7     // 1 hexadecimaldigit = 4bit
8     // 1 byte = 2 hexadecimaldigit
9     bytes5 public temp1;
10    bytes7 public temp2;
11
12    // bytes ke value ko bhi tin tarah se set kar sakte hai
13
14    function setValue() public{ 48706 gas
15        temp1 = "abcde";
16        temp2 = "12abcde";
17    }
18
19    function getdigit() public view returns(bytes1){ 2580 gas
20        return temp1[2];
21    }
22
23    function getlen() public view returns(uint){ 2519 gas
24        return temp1.length;
25    }
26 }
```

- Dynamic Bytes in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract dynamicBytesInBlockchain{
6     bytes public temp;
7
8     constructor(){ infinite gas 303400 gas
9         temp = "123abc45";
10    }
11
12    function pushElement() public { 74794 gas
13        temp.push('c');
14    }
15
16    function popElement() public{ 50617 gas
17        temp.pop();
18    }
19
20    function getlength() public view returns(uint){ 2636 gas
21        return temp.length;
22    }
23
24    function getElement(uint _idx) public view returns(bytes1){ 7187 gas
25        return temp[_idx];
26    }
27
28 }
```

- Enums in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract enumBlockchain{
6     // enum ak user define data types hai
7     // enum ki help se smart contract ke maintences easy ho jati hai
8     //enum ki help se readability badh jati hai
9     // enum ke use se error hone ki chance kam ho jati hai
10
11     enum Status{
12         Pending,
13         Shipped,
14         Accepted,
15         Rejected,
16         Cancel
17     }
18
19     Status public status;
20
21     function getStatus() public view returns(Status){ 2500 gas
22         return status;
23     }
24
25     function setStatus(Status _status) public{ undefined gas
26         status = _status;
27     }
28
29     function reset() public{ 24430 gas
30         delete status;
31     }
32 }
33
34 }
```

- Struct in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 struct Emp{
6     string name;
7     uint age;
8     address acc;
9 }
10
11 contract structBlockchain{
12     // struct keyword ka use hum contract ke bahr use kar sakte hai
13     // struct ke help se khud ka data types bana sakte hai
14     // struct data type me storage me hi store hota hai esliye jab local variable bnayenge to memory keyword ka use karna padega
15
16     // struct Emp{
17     //     string name;
18     //     uint age;
19     //     address acc;
20     // }
21
22     Emp public emp;
23     // emp ak variable hai jo jiska data type Emp hai
24     // emp ke ander ak address hai ye address us data ka hai jaha per name, age, acc hai
25
26 }
```

```
26
27
28 constructor(string memory _name,uint _age,address _acc){  Infinite gas 671600 gas
29     emp.name = _name;
30     emp.age = _age;
31     emp.acc = _acc;
32 }
33
34
35
36
37 Emp[] public emps;
38 // emps ak array hai jiska datatype Emp hai
39
40 function setvalues() public {  Infinite gas
41     Emp memory emp1 = Emp("Nirbhay Gupta",21,0x58380a6a701c568545dcfc803fc8875f56beddc4);
42     // emp = emp1;
43     Emp memory emp2 = Emp({acc:msg.sender,name:"Mohit Gupta",age:23});
44     Emp memory emp3;
45     emp3.name = "Rohit Gupta";
46     emp3.age = 24;
47     emp3.acc = 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB;
48
49     emps.push(emp1);
50     emps.push(emp2);
51     emps.push(emp3);
52
53     emps.push(Emp("Ritik",99,msg.sender));
54 }
```

```
54
55 // Data update in Array
56 Emp storage emp_temp1 = emps[1];
57 emp_temp1.age = 31;
58 delete emp_temp1.acc;
59
60 delete emps[2];
61
62
63
64 // es tarah se hum emp variable ke data ko update kar sakte hai
65 Emp storage emp_temp = emp;
66 // jab storage ke jagah per yadi memory rakhta to emp ka data update nahi ho pata
67 // emp ke ander ak address hai ye address us data ka hai jaha per name, age, acc hai
68 // ab emp ka address emp_temp me aa jayega, ye address blockchain ke ander rakhe data ka address hita hai esliye hum emp ke data ko update kar pate hai
69
70
71 emp_temp.name = "Harsh Gupta";
72
73 // es tarah se bhi emp ke data ko update kar sakte hai
74 // emp.name = "Harsh Gupta";
75
76 }
77
78 }
```

- Mapping in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract mappingBlockchain{
6     // mapping ko contract se bahr nahi define kar sakte hai
7     // mapping ko function ke ander bhi define nahi kar sakte hai i.e mapping ko function level per define nahi kar sakte hai
8     // mapping ko contract level per define kar sakte hai
9
10    mapping(uint=>string) public emp_id;
11    // emp_id ka mapping ke wajah se ak table ban jayega jisme key aur value hoga
12    function setId() public { infinite gas
13        emp_id[31] = "Nirbhay Gupta";
14        emp_id[27] = "Mohit Gupta";
15        emp_id[33] = "Rohit Gupta";
16        emp_id[28] = "Aryan Gupta";
17        emp_id[14] = "Harsh Gupta";
18    }
19
20
21    function getId(uint _id) public view returns(string memory){ infinite gas
22        return emp_id[_id];
23    }
24
25
26 }
```

- Advance Mapping in Blockchain

```
chain.sol  dynamicBytesIn_Blockchain.sol  enumBlockchain.sol  structBlockchain.sol  mappingBlockchain.sol  advanceMapping.sol X
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 struct donor_dts{
6     string name;
7     uint age;
8     string add;
9     uint don;
10 }
11
12 contract advanceMapping{
13     mapping(address=>donor_dts) public acc_info;
14     function set(string memory _name, uint _age, string memory _add, uint _don) public {  Infinite gas
15         acc_info[msg.sender] = donor_dts(_name,_age,_add, acc_info[msg.sender].don+_don);
16     }
17
18     function delete_info() public{  Infinite gas
19         delete acc_info[msg.sender];
20     }
21 }
```

- Visibility in Blockchain

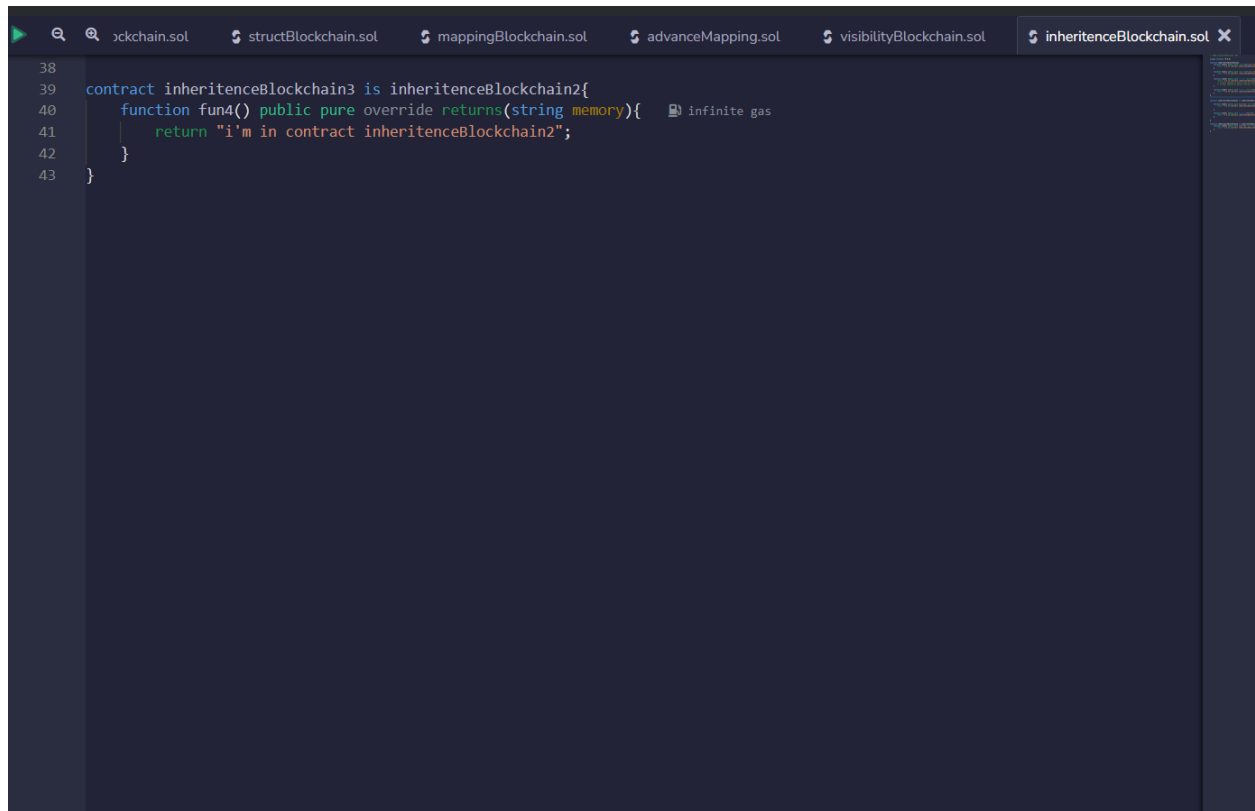
```
bytesInBlockchain.sol  dynamicBytesIn_Blockchain.sol  enumBlockchain.sol  structBlockchain.sol  mappingBlockchain.sol  advanceMapping.sol  visibilityBlockchain.sol 1 X
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract visibilityBlockchain1{
6     // visibility state variable and function ke liye define hoti hai
7     // Visibility char tarah ka hota hai
8     // 1. public
9     // 2. private
10    // 3. Internal
11    // 4. External
12
13    // Gas cost in increasing order --> private > Internal > External > Public
14
15    // Potential call char tarah ka hota hai
16    // 1. Mycontract --> jab ak contract ke ander bane state variable and function jab usi contract me use hota hai to wah Mycontract ke ander ata hai
17
18    // 2. DerivedContract is Mycontract --> jab ak contract se koi dusra contract derived hota hai to wah child contract kahlata hai,
19    //     esme parent contract ke state variable and function ka use child contract me kar sakte hai
20
21    // 3. Another Contract --> jab ak hi file me 2 contract bana aur es dono contract ka koi aapas me relation nahi hai,
22    //     lekin jab ak contract ka state variable and function kisi dusre contract me use karte hai
23
24    // 4. outside world -->
25
26    // private visibility --> eska use kewal Mycontract ke liye hi ho sakta hai i.e
27    // jis variable aur function per private ko use karte hai to varriable aur function ka use usi contract me ho sakta hi jis contract me ye define hai
28
29    // Internal visibility --> eska use Mycontract and DerivedContract is Mycontract me hi kewal use kar sakte hai
30
31    // external visibility --> eska use kabhi bhi state variable banane ke liye use nahi karte hai,
32    //     esaka use kewal AnotherContract and outside world me hi kiya ja sakta hai
33
34    // public visibility --> eska use sabhi tarah ke potential calling me use kiya ja sakta hai
35
36    // public and external visibility ke liye button banta hai
37 }
```

```
new Window Help
bytesInBlockchain.sol dynamicBytesIn_Blockchain.sol enumBlockchain.sol structBlockchain.sol mappingBlockchain.sol advanceMapping.sol visibilityBlockchain.sol 1 X
37
38 // Koi bhi state variable By default internal hota hai aur function By default public hota hai
39
40
41 uint private x = 10; // only within the contract
42 uint internal y = 100; // within the contract and derived contract
43 uint public z = 1000; // everywhere
44
45 function check1() private pure returns(string memory){ infinite gas
46     return "private";
47 }
48
49 function check2() internal pure returns(string memory){ infinite gas
50     return "internal";
51 }
52
53 function check3() external pure returns(string memory){ infinite gas
54     return "external";
55 }
56
57 function check4() public pure returns(string memory){ infinite gas
58     return "public";
59 }
60
61 function checkAll() public view returns(string memory){ infinite gas
62     // return x;
63     // return y;
64     // return z;
65     // return check1();
66     // return check2();
67     // return check3(); // show Error
68     return check4();
69 }
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668


```



```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract inheritanceBlockchain1{
6     function fun1() public pure returns(string memory){ infinite gas
7         return "i'm in contract inheritanceBlockchain1";
8     }
9
10    function fun2() public pure returns(string memory){ infinite gas
11        return "i'm in contract inheritanceBlockchain1";
12    }
13
14    function fun3() public pure virtual returns(string memory){ infinite gas
15        return "i'm in contract inheritanceBlockchain1";
16        // virtual keyword ki help se hum es function ko change kar sakte hai child contract me
17        // virtual keyword se parent contract se permission mil gya hai ki es function ko child contract me edit kar sakte hai
18    }
19
20    function fun4() public pure virtual returns(string memory){ infinite gas
21        return "i'm in contract inheritanceBlockchain1";
22    }
23 }
24
25
26 contract inheritanceBlockchain2 is inheritanceBlockchain1{
27
28     function fun3() public pure override returns(string memory){ infinite gas
29         return "i'm in contract inheritanceBlockchain2";
30     }
31
32     function fun4() public pure virtual override returns(string memory){ infinite gas
33         return "i'm in contract inheritanceBlockchain3";
34     }
35 }
36
37 }
```



The image shows a screenshot of a Solidity code editor with a dark theme. The editor has several tabs at the top: 'blockchain.sol', 'structBlockchain.sol', 'mappingBlockchain.sol', 'advanceMapping.sol', 'visibilityBlockchain.sol', and 'inheritanceBlockchain.sol'. The active tab is 'inheritanceBlockchain.sol'. The code in the editor is as follows:

```
38
39 contract inheritanceBlockchain3 is inheritanceBlockchain2{
40     function fun4() public pure override returns(string memory){  infinite gas
41         return "i'm in contract inheritanceBlockchain2";
42     }
43 }
```

The code defines a contract 'inheritanceBlockchain3' that inherits from 'inheritanceBlockchain2'. It contains a function 'fun4()' which is public, pure, and overrides the function in the parent contract. The function returns a string 'i'm in contract inheritanceBlockchain2'. The editor also shows a gas icon and the text 'infinite gas' next to the function definition.

- Events In Blockchain

```
n.sol  enumBlockchain.sol  structBlockchain.sol  mappingBlockchain.sol  advanceMapping.sol  visibilityBlockchain.sol  inheritanceBlockchain.sol
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract eventsBlockchain{
6
7     // event ka use tab karte hai jab data ko blockchain me dalna ho
8     event balance(address account, string message, uint val);
9
10    function setData(uint _val) public{  infinite gas
11        // jo simple type function ya transaction function hota hai wah kuchh bhi return nahi karta hai
12        emit balance(msg.sender, "has value", _val);
13        // jab koi setData button per click karega to data blockchain ke ander chala jayega,
14        //us data me ether ka value hoga aur ye ether kiske doura send kiya ja raha hai uska account ka detail hoga aur hash value
15    }
16 }
17
18
19
20 contract chatApp{
21     event chat(address indexed _from, address _to, string message);
22
23     function sendMess(address to_, string memory message_) public{  infinite gas
24         emit chat(msg.sender,to_, message_);
25     }
26 }
```

- Require In Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract requireBlockchain{
6     // require ka use Error handling me kiya jata hai
7     // require input validation me kam ata hai
8     // require access control karta hai
9
10    address public owner = msg.sender;
11    uint public age = 25;
12
13    function CheckRequire(uint _x) public{  infinite gas
14        age = age + 5; // yaha per age ka value tab tak change nahi jab tak require wala condition true nahi hoga
15        require(_x>2,"Value of x is less than 2"); // input validation
16        // jab require wala condition false ho jata hai to gas ko return kar diya jata hai
17    }
18
19    function onlyOwner() public{  infinite gas
20        require(owner == msg.sender,"You are note the owner");
21        age = age-2;
22    }
23
24 }
```

- Revert and Assert in Blockchain

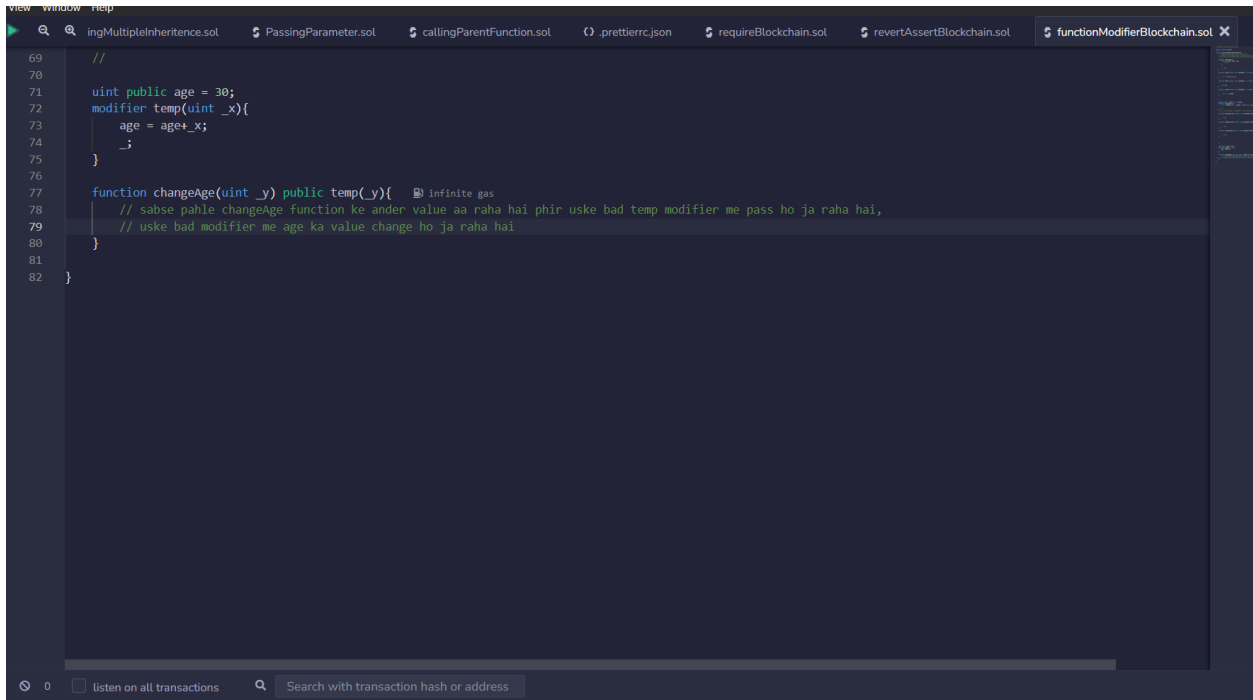
```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract revertAssertBlockchain{
6
7     // Revert bhi require ki tarah hi kam karta hai, esme ak extra function hai ki esme custom error likh sakte hai
8     // assert ka use bug check karne ke liye kiya jata hai aur security ke liye kiya jata hai
9     address public owner = msg.sender;
10    uint public age = 25;
11
12    function CheckRequire(uint _x) public {    infinite gas
13        age = age + 5; // yaha per age ka value tab tak change nahi jab tak require wala condition true nahi hoga
14        require(_x>2,"Value of x is less than 2"); // input validation
15        // jab require wala condition false ho jata hai to gas ko return kar diya jata hai
16    }
17
18    error throwError(string,address); // custom error
19
20    function CheckRevert(uint _x) public {    infinite gas
21        age = age +5;
22        if(_x<2){
23            // revert("value of x is less than 2");
24            revert throwError("value of x is less than 2",msg.sender);
25        }
26    }
27
28    function onlyOwner() public {    infinite gas
29        // require(owner == msg.sender,"You are note the owner");
30
31        if(owner!=msg.sender){
32            revert("You are note the owner");
33        }
34        age = age-2;
35    }
36
37 }
```

```
37
38 // Assert
39 function checkOwnership() public view{    2341 gas
40     assert(owner != 0x5B38Da6a701c568545dCfCB03FcB875f56beddc4);
41 }
42
43
44 }
```

- Function Modifier in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract functionModifierBlockchain{
6     // Modifier ak special type function hai
7     // modifier code reuse karne ki facility deta hai
8     // ak contract ke ander bahut sara modifier ho sakte hai lekin constructor ak hi ho sakta hai
9
10    modifier sameCode(){
11        for(uint i=0; i<10; i++){
12            //code
13        }
14        _;
15        // code
16    }
17
18    function fun1() public pure sameCode returns(string memory){ infinite gas
19
20        return "fun1 say hi";
21    }
22
23    function fun2() public pure sameCode returns(uint _x){ infinite gas
24
25        _x = 20;
26    }
27
28    function fun3() public view sameCode returns(address){ infinite gas
29
30        return msg.sender;
31    }
32
33
34
35
36
37
```

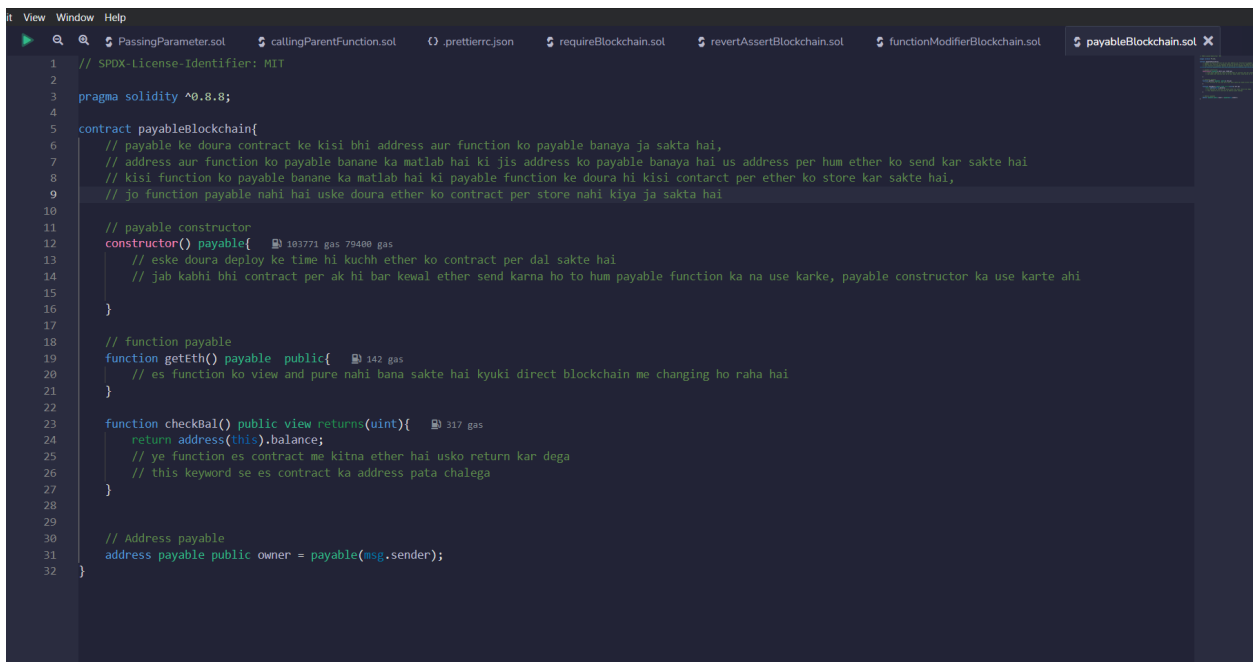
```
38 //
39
40 address public owner = msg.sender;
41 modifier onlyOwner(){
42     require(owner == msg.sender, "Ypu are not the owner");
43     _;
44 }
45 // jis jis function me onlyOwner ka use karenge us function ko kewal owner hi access kar sakta hai
46
47 function startAuction() public view onlyOwner{ 2622 gas
48
49     // code
50 }
51
52 function stopAuction() public view onlyOwner{ 2645 gas
53
54     // code
55 }
56
57 function checkState() public view onlyOwner{ 2688 gas
58
59     // code
60 }
61
62
63
64
65
66
67
68
```



The screenshot shows a Solidity IDE with a file named `functionModifierBlockchain.sol`. The code defines a variable `age` and a modifier `temp` that increments it. A function `changeAge` uses this modifier to update the `age` variable. Comments explain the flow of execution and the purpose of the modifier.

```
//  
69  
70  
71 uint public age = 30;  
72 modifier temp(uint _x){  
73     age = age+_x;  
74     _;  
75 }  
76  
77 function changeAge(uint _y) public temp(_y){ // infinite gas  
78     // sabse pahle changeAge function ke ander value aa raha hai phir uske bad temp modifier me pass ho ja raha hai,  
79     // uske bad modifier me age ka value change ho ja raha hai  
80 }  
81  
82 }
```

- Payable in Blockchain



The screenshot shows a Solidity IDE with a file named `payableBlockchain.sol`. The code defines a contract `payableBlockchain` with a constructor, a `getEth` function, and a `checkBal` function. Comments explain the purpose of each function and the use of the `payable` keyword.

```
1 // SPDX-License-Identifier: MIT  
2  
3 pragma solidity ^0.8.8;  
4  
5 contract payableBlockchain{  
6     // payable ke doura contract ke kisi bhi address aur function ko payable banaya ja sakta hai,  
7     // address aur function ko payable banane ka matlab hai ki jis address ko payable banaya hai us address per hum ether ko send kar sakte hai  
8     // kisi function ko payable banane ka matlab hai ki payable function ke doura hi kisi contract per ether ko store kar sakte hai,  
9     // jo function payable nahi hai uske doura ether ko contract per store nahi kiya ja sakta hai  
10  
11     // payable constructor  
12     constructor() payable{ // 183771 gas 79408 gas  
13         // eske doura deploy ke time hi kuchh ether ko contract per dal sakte hai  
14         // jab kabhi bhi contract per ak hi bar kewal ether send karna ho to hum payable function ka na use karke, payable constructor ka use karte ahi  
15     }  
16  
17  
18     // function payable  
19     function getEth() payable public{ // 142 gas  
20         // es function ko view and pure nahi bana sakte hai kyuki direct blockchain me changing ho raha hai  
21     }  
22  
23     function checkBal() public view returns(uint){ // 317 gas  
24         return address(this).balance;  
25         // ye function es contract me kitna ether hai usko return kar dega  
26         // this keyword se es contract ka address pata chalega  
27     }  
28  
29  
30     // Address payable  
31     address payable public owner = payable(msg.sender);  
32 }
```

- Fallback and Receive in Blockchain

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5
6 /*
7  1. It is executed when a non-existent function is called on the contract.
8  2. It is required to be marked external.
9  3. It has no name.
10 4. It has no arguments
11 5. It can not return any thing.
12 6. It can be define one per contract.
13 7. If not marked payable, It will throw exception if contract receives ether.
14 8. It's main use is to directly send the ETH to contract
15 */
16
17
18 contract fallbackReceiveBlockchain{
19
20     event log(string _fun,address _sender,uint _val,bytes _data);
21
22     fallback() external payable{ undefined gas
23         // ye ether or data dono ko le sakta hai
24         // receive kewal ether ko hi le sakta hai
25         // es function ke help se direct hi es contract per ether ko get kiya ja sakta hai aur data bhi
26
27         emit log("fallback", msg.sender, msg.value, msg.data);
28
29         /*
30          msg.sender, msg.value, msg.data ye predefine global variable hai
31         */
32     }
33
34 }
```

```
35 receive() external payable{ undefined gas
36     // es function ke doura hamesha ether ko get hi kiya ja sakta hai es contract per esliye eske liye payable use hamesha karna padta hai
37     // receive kabhi bhi data nahi le sakta ahi
38
39     emit log("fallback", msg.sender, msg.value, "");
40 }
41
42 /*
43  jab kabhi bhi fallback and receive dono ko ak hi sath define kar dete hai to us case me
44  jab data+ether dono sath me send karenge to fallback ke doura accept kiya jayega
45  lekin jab kewal ether ko send kiya jayega to receive de doura accept kiya jayega
46 */
47
48
49 function checkBal() public view returns(uint){ 317 gas
50     return address(this).balance;
51 }
52
53 }
```

- Send, Transfer, Call in Blockchain

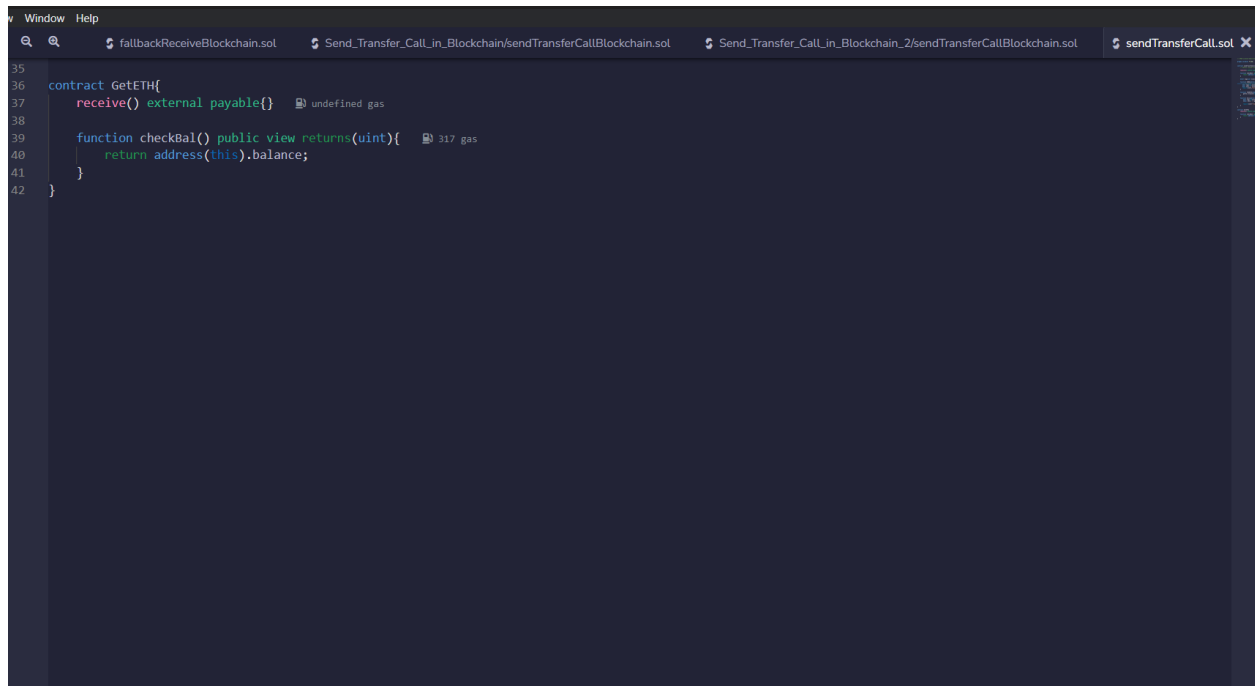

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5
6 /*
7  1. payable function, payable constructor, fallback and receive ki help se hum kisi contract per ether ko store kar sakte hai
8  2. Send, Transfer, Call ki help se hum ak contract se kisi dusre contract me ether ko send kar sakte hai
9  aur eske help se ak contract se kisi dusre ke account me ether ko bhej sakte hai
10 3. Send --> Send ka gas limit 2300 hota hai, yadi es gas limit se jdaya ka operation raha to us case me ye fail ho jata hai
11 jab operation fail ho jata hai to ye gas ko return bhi nahi karta hai esliye Transfer aur Call ko use kiya jata ahi
12 jab operation fail ho jata hai to es operation ke douran jo bhi state variable me change hota hai to wah revert nahi ho pata hai
13 esliye eska use require ke sath use karte hai, jab send ke sath require ka use karte hai to jab operation fail ho jata hai to bachha gas wapas aa jata hai
14 ye hamesha true ya false ko return karta hai; true --> success, false --> fail
15 4. Transfer --> eska bhi gas limit 2300 hota hai, lekin ye state variable ke change ko phir se pahle jaisa kar deta hai aur bache gas ko bhi return kar deta
16 hai, ye sara kam khud se hi kar deta hai
17
18 5. Call --> esme hum gas limit ko decide kar sakte hai. ye bool aur sath hi sath byte data ko return karta hai.
19 eske sath bhi require ka use krna padta hi kyuki Send wala hi problem esme bhi hai
20
21 */
22
23 contract SendTransferCallBlockchain{
24     address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677d03315835cb2);
25
26     receive() external payable{} // undefined gas
27
28     function checkBal() public view returns(uint){ // 361 gas
29         return address(this).balance;
30     }
31
32     function SEND() public{ // infinite gas
33         bool sent = getter.send(1000000000000000000);
34         require(sent,"tran is fail");
35     }
36 }
```

```
36
37     function TRANSFER() public{ // infinite gas
38         getter.transfer(1000000000000000000);
39     }
40
41     function CALL() public { // infinite gas
42         (bool sent, ) = getter.call{value:1000000000000000000}("");
43         require(sent,"tran is failed");
44     }
45
46 }
```

- Send, Transfer, Call
→ address provided by user

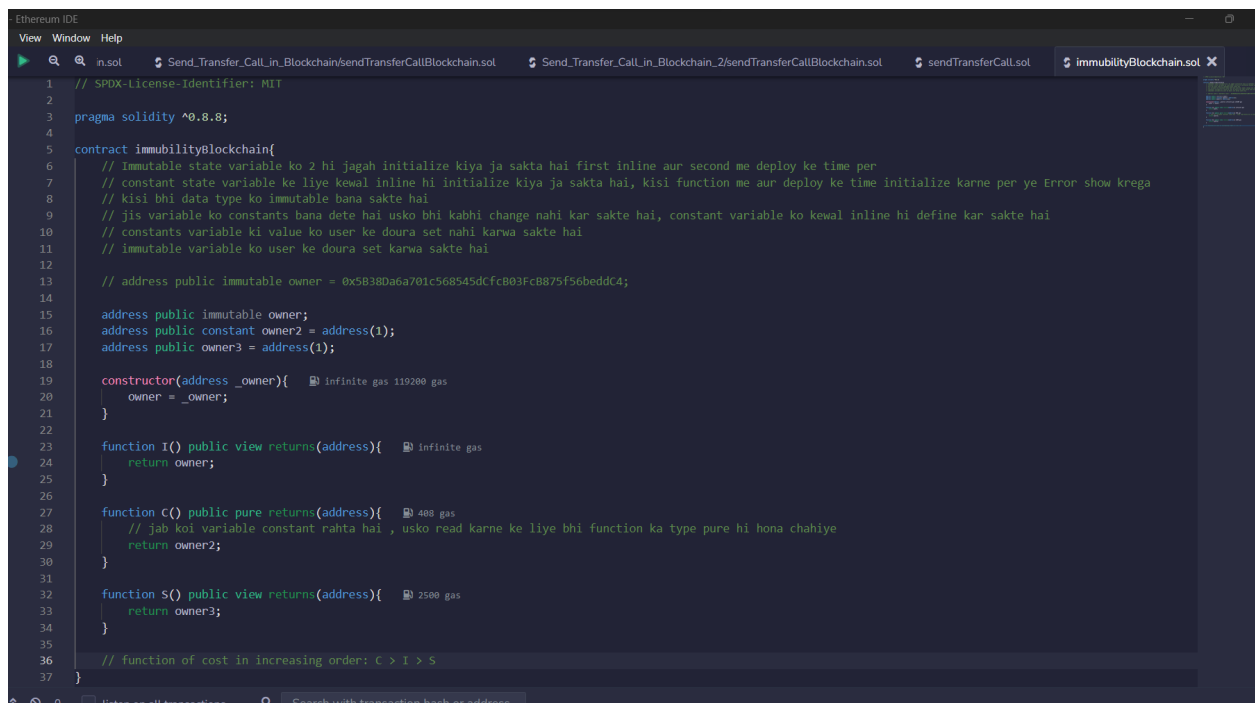
```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5
6 contract sendTransferCall{
7     // address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);
8
9     receive() external payable{} // undefined gas
10
11     function checkBal() public view returns(uint){ // 317 gas
12         return address(this).balance;
13     }
14
15     function SEND(address payable getter) public{ // undefined gas
16         bool sent = getter.send(1000000000000000000);
17         require(sent,"tran is fail");
18     }
19
20     function TRANSFER(address payable getter) public{ // undefined gas
21         getter.transfer(1000000000000000000);
22     }
23
24     function CALL(address payable getter) public { // undefined gas
25         (bool sent, ) = getter.call{value:1000000000000000000}("");
26
27         require(sent,"tran is failed");
28     }
29 }
```

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5
6 contract sendTransferCall{
7     // address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);
8
9     receive() external payable{} // undefined gas
10
11     function checkBal() public view returns(uint){ // 317 gas
12         return address(this).balance;
13     }
14
15     event log(uint value);
16
17     function SEND(address payable getter) public payable{ // undefined gas
18         // es tarah se ether ak account se dusre account me sath hi sath ja raha ahi
19         emit log(msg.value);
20         bool sent = getter.send(msg.value);
21         require(sent,"tran is fail");
22     }
23
24     function TRANSFER(address payable getter) public payable{ // undefined gas
25         getter.transfer(msg.value);
26     }
27
28     function CALL(address payable getter) public payable{ // undefined gas
29         emit log(msg.value);
30         (bool sent, ) = getter.call{value:msg.value}("");
31
32         require(sent,"tran is failed");
33     }
34 }
35
```



```
35
36 contract GetETH{
37     receive() external payable{}  undefined gas
38
39     function checkBal() public view returns(uint){  317 gas
40         return address(this).balance;
41     }
42 }
```

- Immutability in Blockchain



```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract immubilityBlockchain{
6     // Immutable state variable ko 2 hi jagah initialize kiya ja sakta hai first inline aur second me deploy ke time per
7     // constant state variable ke liye kewal inline hi initialize kiya ja sakta hai, kisi function me aur deploy ke time initialize karne per ye Error show krega
8     // kisi bhi data type ko immutable bana sakte hai
9     // jis variable ko constants bana dete hai usko bhi kabhi change nahi kar sakte hai, constant variable ko kewal inline hi define kar sakte hai
10    // constants variable ki value ko user ke doura set nahi karwa sakte hai
11    // immutable variable ko user ke doura set karwa sakte hai
12
13    // address public immutable owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
14
15    address public immutable owner;
16    address public constant owner2 = address(1);
17    address public owner3 = address(1);
18
19    constructor(address _owner){  Infinite gas 119280 gas
20        owner = _owner;
21    }
22
23    function I() public view returns(address){  Infinite gas
24        return owner;
25    }
26
27    function C() public pure returns(address){  400 gas
28        // jab koi variable constant rahta hai , usko read karne ke liye bhi function ka type pure hi hona chahiye
29        return owner2;
30    }
31
32    function S() public view returns(address){  2500 gas
33        return owner3;
34    }
35
36    // function of cost in increasing order: C > I > S
37 }
```

- Data Location - Storage, Memory and Calldata in Blockchain

```
Sheruum IDE
View Window Help
Blockchain/sendTransferCallBlockchain.sol Send_Transfer_Call_in_Blockchain_2/sendTransferCallBlockchain.sol sendTransferCall.sol immubilityBlockchain.sol storageMemoryCalldata.sol X

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract storageMemoryCalldata{
6     // storage ak box hai jo state variable ko store karta hai, aur ye storage khud blockchain per hota hai
7     // memory ke ander local variable store hote hai, ye RAM me aur stack me hota hai, ye blockchain per nahi jata hai esliye eska cost bahut kam hota hai
8     /*
9     | kuchh data types By default storage me hi store hota hai jab es tarah ke data types ke variable ko local variable banana ho to
10    | 'memory' keyword ka use karna padta hai
11    */
12    // Calldata bhi RAM per hi hota hai
13    /*
14    | Calldata ka use function input me hota hai, jaha jaha function input me 'memory' ka use karte hai waha waha per Calldata ka use kar sakte hai
15    | kyuki eska gas cost bahut kam hota hai
16    */
17    // jis data ko 'Calldata' me dal dete hai to us data ko change nahi kar sakte ahi
18    // Gas cost in increasing order : Calldata > Memory > Storage
19
20    uint[] public arr = [1,3,7,9,22];
21
22
23    function Storage() public{ 24465 gas
24        uint[] storage arrs = arr;
25        // arrs me arr ko photocopy nahi aya hai, arrs ke data ko change kerne per arr ka data change ho jayega
26        arrs[3] = 99;
27    }
28
29    function Memeory() public view{ infinite gas
30        uint[] memory arrm = arr;
31        // arrm me arr ke data ka photocopy aya hai, arrm me data ko change karne se arr ke data chnge nahi hoga
32        arrm[1] = 192;
33    }
34
35 }
```

For Calldata

```
View Window Help
sol Send_Transfer_Call_in_Blockchain_2/sendTransferCallBlockchain.sol sendTransferCall.sol immubilityBlockchain.sol storageMemoryCalldata.sol storageMemoryCalldata.sol 2 X

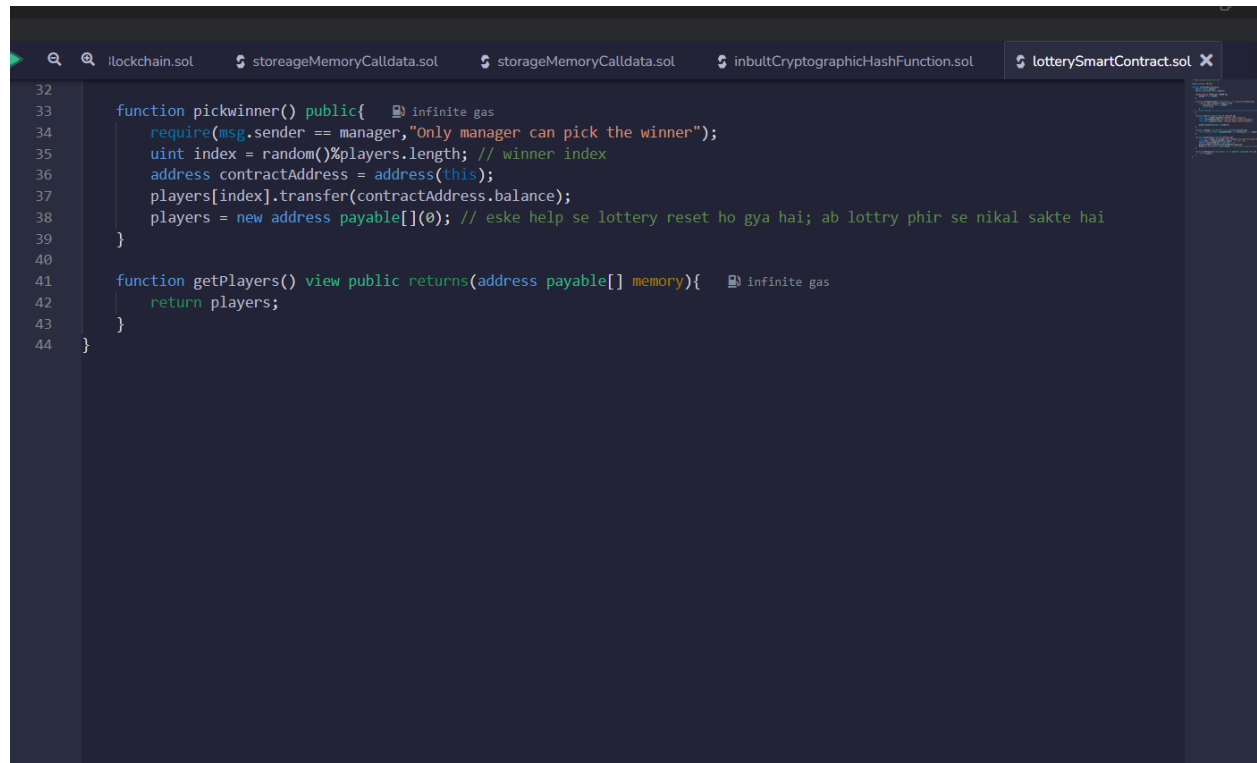
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract storageMemoryCalldata{
6     function Memory(string memory_str, uint[] memory arr) public{ infinite gas
7         getInMem(arr);
8         // getInCall(arr); // show Error
9     }
10
11     function Calldata(string calldata_str, uint[] calldata arr) public{ infinite gas
12         getInMem(arr);
13         getInCall(arr);
14         // Calldata se memory me pass ho sakta hai lekin memory se Calldata me pass nahi ho sakta hai
15         // storage se storage pass ho sakta hai data
16         // storage se memory me bhi pass ho sakta hai data
17         // Calldata se Calldata me pass ho sakta hai data,
18         // Calldata se Calldata me jab array pass ho raha hai to koi new array create nahi ho raha hai wahi same to same array hai
19         // jab Calldata se Memory me Array pass ho raha ahi to Memory me ak new array create ho raha hai esliye eska gas cost calldata se jyada hota hai
20     }
21
22     function getInMem(uint[] memory arr) public{ infinite gas
23     }
24
25
26     function getInCall(uint[] calldata arr) public{ 474 gas
27     }
28
29 }
```

- Inbuilt Cryptography hash function



```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract inbuiltCryptographicHashFunction{
6     function hash1Keccak256(uint _x, string memory name, address _add) public pure returns(bytes32){ infinite gas
7         return keccak256(abi.encodePacked(_x,name,_add));
8     }
9
10    function hash2Keccak256(uint _x, string memory name, address _add) public pure returns(bytes32){ infinite gas
11        return keccak256(abi.encode(_x,name,_add));
12    }
13
14    function hashsha256(uint _x, string memory name, address _add) public pure returns(bytes32){ infinite gas
15        return sha256(abi.encodePacked(_x,name,_add));
16    }
17
18    function hashRipemd160(uint _x, string memory name, address _add) public pure returns(bytes32){ infinite gas
19        return ripemd160(abi.encodePacked(_x,name,_add));
20    }
21 }
```

- Lottery Smart Contract

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.8;
4
5 contract lotterySmartContract{
6     address public manager;
7     address payable[] public players;
8
9     constructor(){ 764039 gas 739000 gas
10         manager = msg.sender;
11     }
12
13     function alreadyEntered() view private returns(bool){ infinite gas
14         for(uint i=0; i<players.length; i++){
15             if(players[i] == msg.sender)
16                 return true;
17         }
18         return false;
19     }
20
21     function enter() payable public{ infinite gas
22         require(msg.sender!=manager,"Manager cannot enter");
23         require(alreadyEntered() == false,"Player already entered");
24         require(msg.value>=1 ether, "Minimun amount must be payed");
25
26         players.push(payable(msg.sender));
27     }
28
29     function random() view private returns(uint){ infinite gas
30         return uint(sha256(abi.encodePacked(block.difficulty,block.number,players)));
31     }
32 }
```



The image shows a screenshot of a Solidity code editor with a dark theme. The editor has several tabs at the top: 'lockchain.sol', 'storeageMemoryCalldata.sol', 'storageMemoryCalldata.sol', 'inbuiltCryptographicHashFunction.sol', and 'lotterySmartContract.sol'. The 'lotterySmartContract.sol' tab is active. The code is written in Solidity and includes two functions: 'pickwinner()' and 'getPlayers()'. The 'pickwinner()' function is public and takes no arguments. It starts with a 'require' statement to ensure the sender is the manager. It then generates a random index, sets the contract address, transfers the balance to the player at that index, and resets the players array. The 'getPlayers()' function is view public and returns the players array. The code is annotated with line numbers 32 through 44. The 'pickwinner()' function has a comment in Hindi: '// eske help se lottery reset ho gya hai; ab lottry phir se nikal sakte hai'.

```
32
33 function pickwinner() public {  infinite gas
34     require(msg.sender == manager, "Only manager can pick the winner");
35     uint index = random()%players.length; // winner index
36     address contractAddress = address(this);
37     players[index].transfer(contractAddress.balance);
38     players = new address payable[](0); // eske help se lottery reset ho gya hai; ab lottry phir se nikal sakte hai
39 }
40
41 function getPlayers() view public returns(address payable[] memory) {  infinite gas
42     return players;
43 }
44 }
```

- Auction Smart Contract