SOLIDITY

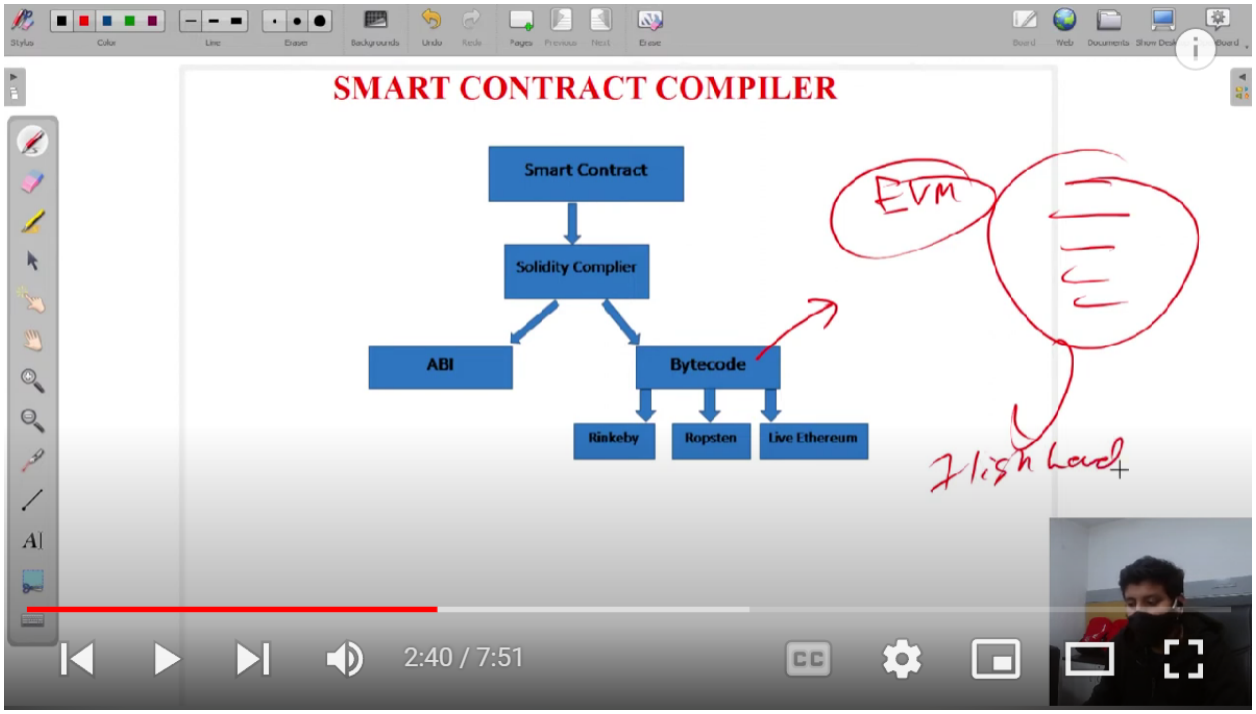
- Overview of Sample Smart Contract

Home | README.txt | sample.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ownable{
    address public owner;
    // es tarah se owner nam ka ak variable set kiya hai jiska datatype address hai

    constructor(){    251335 gas 226800 gas
        owner = msg.sender;
        // eske halp se owner variable me ak address store ho jayega, jo owner ke account ka address hoga

    }

    modifier onlyOwner(){
        require(msg.sender == owner, "not owner");
        _;

        // eske help se contract ka main power kewal owner ke pass hi aa jayega
    }

    function setOwner(address _newOwner) public onlyOwner{    26902 gas
        require(_newOwner != address(0),"Invalid address");
        owner = _newOwner;

        // eske help se owner apne contract ka owner kisi dusre ko bana sakta hai ya set kar sakta hai
    }

    function onlyOwnerCanAccess() public onlyOwner{    2622 gas
        // code
    }

    function anyOneCanAccess() public {    144 gas
        // code
    }
}
```

- 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

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// data types : values and references
// values: bool, int, address, bytes etc
// references: Array etc

contract valuesType{
    bool public boolTemp = true;
    uint public temp = 243; // eska use hamesha gas ke lenden me use hota hai, eska range: 0 to 2^256 - 1
    int public temp2 = 143; // eska range -2^255 to 2^255 - 1
    address public addr = 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2;
    // esme 20 bits tak data store ho sakta hai; esme data hexadecimal ke form me data store hota hai
    bytes32 public b3; // string ke jagah per eska use krte hai, ye bhi hexadecimal ko hi store krta hai

}

// 0xd9145CCE52D386f254917e481eB44e9943F39138 ye valueType contract ka address hai
```

- Function in Solidity

Home | Datatypes/1_sample.sol | functionIntro.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract functionIntro{
    uint age = 20;
    function add(uint _x, uint _y) public pure returns(uint){    infinite gas
        return _x + _y;
    }

    function changeAge() public{    infinite gas
        age+= 1;
    }

    function getage() public view returns(uint){    2459 gas
        return age;
    }

    function fun() internal{    infinite gas
        // code
    }
}

function dummy(uint _x) pure returns(uint){
    return _x*10;
    // ye function contract ke bahr likha gya hai; eski visibility bydefault internal hoti hai
    // jo function contract ke bahr likha rahta hi unki visibility change nahi kar sakte hai
    // es function ka use hum apne contract me import karne ke liye use kar sakte hai
}
```

- State Variable

Home | Datatypes/1_sample.sol | functionIntro.sol | stateVariable.sol | localVariable.sol

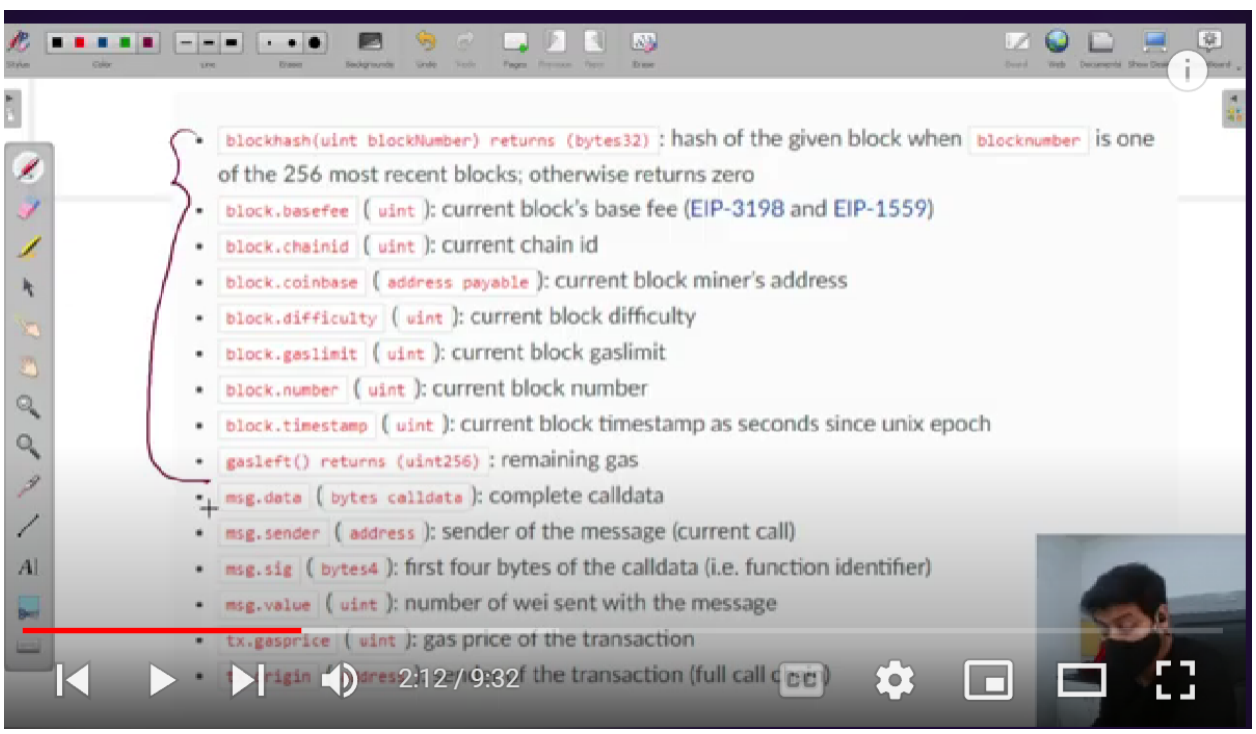
```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract stateVariable{
    // state variable contract ke ander aur function ke bahr define hota hai
    // state variable sidha blockchain and contract storage per store hota hai joki costly hai

    uint public salary;

    constructor(){    63807 gas 41600 gas
        salary=100;
        // constructor ak special function hai jo apne aap hi call ho jata hai
    }

    function setSalary() public{    22258 gas
        salary = 2000;
    }
}
```

- Local Variable

Home | Datatypes/1_sample.sol | functionIntro.sol | stateVariable.sol | localVariable.sol | README.txt

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract localVariable{
    // jo variable contract ke kisi function ke ander define hota hai wah local variable hota hai
    // local variable memory me store hota hai i.e RAM me store hota hai
    // local variable ke liye negilegible cost i.e gas lagta hai
    // local variable execution ke bad memory se delete ho jata hai

    uint public age;
    bool public b;
    address public newAdd;

    function fun(uint _x, bool _y, address _z) public returns(uint,bool){    infinite gas
        uint i = 29;
        bool b1 = true;
        i+= 29;
        b1 = false;


        age = _x;
        b = _y;
        newAdd = _z;

        return (i,b1);
    }
}
```

- Global Variable



Home | Datatypes/1_sample.sol | functionIntro.sol | stateVariable.sol | localVariable.sol | README.txt | global_Variable.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract global_Variable{
//   These are spacial variable which exist in global workspace and provide
//   information about the blockchain and transaction properties.
// ye predefine variable hota hai e.g msg.sender
// msg.sender --> ye account ko detail ko deta hai jis account se deploy huaa rahta hai
// msg.sender --> ye us account ke detail ko bhi deta hai jo mere smart contract se communicate karna chahta hai
// msg.sender --> jab koi smart contract ke function call hota us function ka bhi information deta hai

address public myadd = msg.sender;
uint public time = block.timestamp;
uint public diff = block.difficulty;
uint public gascost = tx.gasprice;
}
```

- View, Pure and Simple Function

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract view_pure_simple_function{
    // view --> ye read only karta hai;
    // ye blockchain ke ander, statevariable, global variable ko kewal read karta hai usko change nahi karta hai i.e write nahi karta hai

    // pure --> ye na to read karta hai aur nahi write karta hai

    // simple function --> ye state variable ko change kar sakte hai


    uint public age = 20;
    function viewFunc() public view returns(uint){    2481 gas
        return age;
    }

    function pureFunc1() public pure returns(uint){    315 gas
        return 1;
    }

    function pureFunc2(uint _x) public pure returns(uint){    infinite gas
        return _x;
    }

    function simpleFunc() public{    infinite gas
        age = age + 10;
        // yaha per age ka value read bhi ho raha hai aur age ka value write bhi ho raha hai
    }

    function dummy() public view returns(uint){    infinite gas
        return age+10;
        // yaha per kewal view type function age ko read kar raha hai , us age me koi change nahi kar raha hai;
        // ye kewal age ko read karke usme 10 ko add kiya hai kewal
    }
}
```

- Default Values

stateVariable.sol | localVariable.sol | README.txt | global_Variable.sol | view_pure_simple_function.sol | default_Values.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract default_Values{
    uint public x;
    bool public y;
    address public z;
    bytes32 public a;
    string public str;
}
```

- Strings

e.sol | localVariable.sol | README.txt | global_Variable.sol | view_pure_simple_function.sol | default_Values.sol | string.sol 1

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract strings{
    // String bydefault blockchain i.e contract storage me store hota hai , chahe local variable ho ya state variable ho
    // esliye string data type ko jab local variable banate hai to memory keyword ka use karna padta hai

    string public str = "gaurav"; // str state variable hai
    function checkstring() public pure{    213 gas
        string memory name = "aman";

        // es function ko return nahi kar rahe hai esliye button click karne per dikh nahi raha hai
    }

    function checkstring2() public pure returns(string memory){    infinite gas
        string memory name2 = "Nirbhay Gupta";
        return name2;

        // es function ko return  kar rahe hai esliye button per click karne per dikh  raha hai
    }

    function checkstring3(string memory _str) public pure returns(string memory){    infinite gas
        string memory name3 = _str;
        return name3;
    }
}
```

- Constants

ionIntro.sol | stateVariable.sol | localVariable.sol | README.txt | global_Variable.sol | view_pure_simple_function.sol | default_Values.sol | string.sol | constants.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract constants{
    // jab kisi variable ka value change n karna ho to constant keyword ka use karte hai
    // jab kisi contract ke owner ko change nahi karne ho to us case me constant keyword ka use kar sakte hai
    // jab state variable bnate hai to uska gast cost jyada hota hai usi gas cost ko kam karne aur efficiency ko badhane ke liye constant keyword ka use karte hai
    address public constant owneradd = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    address public owneradd2 = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;

}
```

- Constructor in Blockchain

```
:ol  localVariable.sol  README.txt  global_Variable.sol  view_pure_simple_function.sol  default_Values.sol  string.sol  constants.sol  blockchainConstructor.sol
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract blockchainConstructor{
    // ak contract me kewal ak hi constructor ho sakta hai
    // 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
    // eska use smart conract ke owner ko set karne ke liye kiya jata hai ya state variable ko initialize karne ke liye use kiya jata hai
    // state variable ko initialize karne ke liye tin tarike hote hai
    // pahle state variable banate time hi initialize kar sakte hai, dusra constructor ke doura, tisra function ke doura

    uint public age;
    address public owner;
    string public name;

    constructor(string memory _name, uint _age, address _owner){    infinite gas 163600 gas
        //  age = 20;
        age = _age;
        owner = _owner;
        name = _name;

        // jab constructor me as argument koi value lete hai to ye deploy ke time hi es value ko dena padega as input, nahi to samart contract deploy nahi hoga
    }
}
```

- If-else in Blockchain

```
view_pure_simple_function.sol  default_Values.sol  string.sol  constants.sol  blockchainConstructor.sol  if_elseBlockchain.sol
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract if_elseBlockchain{

    // if else ka use contract level per use nahi kar sakte hai

    function fun(uint _x) public pure returns(string memory){    infinite gas
        string memory val;

        if(_x>100){
            val = "greater than 100";
        }
        else if(_x<100){
            val = "smaller than 100";
        }
        else{
            val = "equal to 100";
        }

        return val;

    }
}
```

- Ternary Operator in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract ternaryOperator{
    function fun(uint _x) public pure returns(string memory){
        string memory val;

        val = _x>100 ? "greater than 100" : "equal or smaller than 100";
        return val;
    }
}
```

- Loops in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract loopBlockchain{
    // loop bhi contract level per kam nahi karte hai, ye function level per kam karte hai

    function loop() public pure returns(uint){
        uint count = 0;
        for(uint i=0; i<10; i++){
            count = count+5;
        }
        // return count;

        uint j=0;
        while(j<10){
            count = count+10;
            j++;
        }

        // return count;

        do{
            count = count+10;
            j++;
        }while(j<10);

        return count;
    }

}
```

- Continue and Break Keyword in Blockchain

onstants.sol | blockchainConstructor.sol | if_elseBlockchain.sol | ternaryOperator.sol | loopBlockchain.sol | continueBreakKeyword.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract continueBreakKeyword{
    function loop() public pure returns(uint){    infinite gas
        uint count = 0;
        // continue and break ka use if else ke bahr bhi use kar sakte hai

        for(uint i=0; i<10; i++){
            if(i==8){
                continue;
            }
            count = count+5;
        }

        for(uint i=0; i<10; i++){
            if(i==6){
                break;
            }
            count = count+5;
        }


        return count;
    }
}
```

- Fixed Size Array in Blockchain

View Window Help

string.sol | constants.sol | blockchainConstructor.sol | if_elseBlockchain.sol | ternaryOperator.sol | loopBlockchain.sol | continueBreakKeyword.sol | fixedSizeArray.sol

```
SPDX-License-Identifier: MIT

agma solidity ^0.8.8;

ntract fixedSizeArray{
  uint[5] public arr = [10,20,30,40,50];
  uint[5] public arr1;

  uint[5] arr2;
  constructor(){    infinite gas 294400 gas
      arr2 = [10,20,30,40,50];
  }

  function returnArray() public view returns(uint[5] memory){    infinite gas
      return arr2;
      // es tarah se kisi function se kisi array ko pass kar sakte hai
      // array return karne per gas kaphi jyada lagta hai esliye array ko return nahi karte hai
  }


  // 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,
  // ese contract ke ander aur function ke bahr nahi kar sakte hai
  // contract ke ander kewal state variable, function, and modifier hi bana sakte hai
  // array bhi string ki tarah bydefault storage me store hota hai
```

```
function array() public view returns(uint){    2580 gas
    // get
    uint temp;
    temp = arr[3];

    // update
    // arr[2] = 3000;

    // delete
    // delete arr[4];

    // length
    uint len = arr.length;
    return len;

}

function arrayWithLoop(uint _x) public {    infinite gas
    for(uint i=0; i<arr.length; i++){
        arr1[i] = _x+i;
    }
}

function createArray() public pure returns(uint){    infinite gas
    uint[] memory arrMem = new uint[](3);
    // memory ke ander kewal fixed size array hi bana sakte hai, Dynamic array nahi bana sakte hai
    // es tarah se kisi function ke ander array create karte hai

    arrMem[0] = 33;
    arrMem[1] = 35;
    arrMem[2] = 40;

    return arrMem[0];
    // array return karne per gas kaphi jyada lagta hai esliye array ko return nahi karte hai
    // return arrMem;
}
```

- Dynamic Size Array in Blockchain

```
:eBlockchain.sol   ternaryOperator.sol   loopBlockchain.sol   continueBreakKeyword.sol   fixedSizeArray.sol   dynamicSizeArray.sol 2

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract dynamicSizeArray{
    // Dynamic array memory ke ander nahi ho sakte hai
    uint[] public arr = [10,299,3432,434,34,32,1];

    function returnArray() public view returns(uint[] memory){    infinite gas
        return arr;
    }

    function fun() public{    113787 gas
        // get
        uint temp = arr[3];

        // update
        arr[4] = 222; // [10,299,3432,434,222,32,1]

        // delete
        delete arr[1]; // [10,0,3432,434,222,32,1]

        // length
        uint len = arr.length;

        // push
        arr.push(111); // [10,0,3432,434,34,32,1,111]


        // pop
        arr.pop(); // ye array ke last element ko pop krta hai
    }
}
```

- Bytes in Blockchain

naryOperator.sol | loopBlockchain.sol | continueBreakKeyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract bytesInBlockchain{
    // 1 byte = 8bits
    // 1 hexadecimaldigit = 4bit
    // 1 byte = 2 hexadecimaldigit
    bytes5 public temp1;
    bytes7 public temp2;

    // bytes ke value ko bhi tin tarah se set kar sakte hai

    function setvalue() public{    48706 gas
        temp1 = "abcde";
        temp2 = "12abcde";
    }

    function getdigit() public view returns(bytes1){    2580 gas
        return temp1[2];
    }

    function getlen() public view returns(uint){    2519 gas
        return temp1.length;
    }
}
```

- Dynamic Bytes in Blockchain

ı.sol | continueBreakKeyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract dynamicBytesInBlockchain{
    bytes public temp;

    constructor(){    infinite gas 303400 gas
        temp = "123abc45";
    }

    function pushElement() public {    74794 gas
        temp.push('c');
    }

    function popElement() public{    50617 gas
        temp.pop();
    }

    function getlength() public view returns(uint){    2636 gas
        return temp.length;
    }

    function getElement(uint _idx) public view returns(bytes1){    7187 gas
        return temp[_idx];
    }

}
```

- Enums in Blockchain

eyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract enumBlockchain{
    // enum ak user define data types hai
    // enum ki help se smart contract ke maintences easy ho jati hai
    //enum ki help se readability badh jati hai
    // enum ke use se error hone ki chance kam ho jati hai

    enum Status{
        Pending,
        Shipped,
        Accepted,
        Rejected,
        Cancel

    }

    Status public status;

    function getStatus() public view returns(Status){    2590 gas
        return status;
    }

    function setStatus(Status _status) public{    undefined gas
        status = _status;
    }

    function reset() public{    24430 gas
        delete status;
    }

}
```

- Struct in Blockchain

continueBreakKeyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

struct Emp{
        string name;
        uint age;
        address acc;
    }

contract structBlockchain{
    // struct keyword ka use hum contract ke bahr use kar sakte hai
    // struct ke help se khud ka data types bana sakte hai
    // struct data type me storage me hi store hota hai esliye jab local variable bnayenge to memory keyword ka use karna padega

    // struct Emp{
    //     string name;
    //     uint age;
    //     address acc;
    // }

    Emp public emp;
    // emp ak variable hai jo jiska data type Emp hai
    // emp ke ander ak address hai ye address us data ka hai jaha per name, age, acc hai
```

continueBreakKeyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol

```
constructor(string memory _name,uint _age,address _acc){    infinite gas 671600 gas
    emp.name = _name;
    emp.age = _age;
    emp.acc = _acc;
}



Emp[] public emps;
// emps ak array hai jiska datatype Emp hai

function setvalues() public {    infinite gas
    Emp memory emp1 = Emp("Nirbhay Gupta",21,0x5B38Da6a701c568545dCfcB03FcB875f56beddC4);
    // emp = emp1;
    Emp memory emp2 = Emp({acc:msg.sender,name:"Mohit Gupta",age:23});
    Emp memory emp3;
    emp3.name = "Rohit Gupta";
    emp3.age = 24;
    emp3.acc = 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB;

    emps.push(emp1);
    emps.push(emp2);
    emps.push(emp3);

    emps.push(Emp("Ritik",99,msg.sender));
```

continueBreakKeyword.sol | fixedSizeArray.sol | dynamicSizeArray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol

```
    // Data update in Array
    Emp storage emp_temp1 = emps[1];
    emp_temp1.age = 31;
    delete emp_temp1.acc;

    delete emps[2];


    // es tarah se hum emp variable ke data ko update kar sakte hai
    Emp storage emp_temp = emp;
    // jab storage ke jagah per yadi memory rahta to emp ka data update nahi ho pata
    // emp ke ander ak address hai ye address us data ka hai jaha per name, age, acc hai
    // 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


    emp_temp.name = "Harsh Gupta";

    // es tarah se bhi emp ke data ko update kar sakte hai
    // emp.name = "Harsh Gupta";

 }

}
```

- Mapping in Blockchain

rray.sol | bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol | mappingBlockchain.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract mappingBlockchain{
    // mapping ko contract se bahr nahi define kar sakte hai
    // mapping ko function ke ander bhi define nahi kar sakte hai i.e mapping ko function level per define nahi kar sakte hai
    // mapping ko contract level per define kar sakte hai

    mapping(uint=>string) public emp_id;
    // emp_id ka mapping ke wajah se ak table ban jayega jisme key aur value hoga
    function setId() public{    infinite gas
        emp_id[31] = "Nirbhay Gupta";
        emp_id[27] = "Mohit Gupta";
        emp_id[33] = "Rohit Gupta";
        emp_id[28] = "Aryan Gupta";
        emp_id[14] = "Harsh Gupta";

    }

    function getId(uint _id) public view returns(string memory){    infinite gas
        return emp_id[_id];
    }


}
```

- Advance Mapping in Blockchain

:hain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol | mappingBlockchain.sol | advanceMapping.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

struct donor_dts{
    string name;
    uint age;
    string add;
    uint don;
}

contract advanceMapping{
    mapping(address=>donor_dts) public acc_info;
    function set(string memory _name, uint _age, string memory _add, uint _don) public {    infinite gas
        acc_info[msg.sender] = donor_dts(_name,_age,_add, acc_info[msg.sender].don+_don);
    }

    function delete_info() public{    infinite gas
        delete acc_info[msg.sender];
    }
}
```

- Visibility in Blockchain

bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol | mappingBlockchain.sol | advanceMapping.sol | visibilityBlockchain.sol 1

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract visibilityBlockchain1{
    // visibility state variable and function ke liye define hoti hai
    // Visibility char tarah ka hota hai
    // 1. public
    // 2. private
    // 3. Internal
    // 4. External

    // Gas cost in increasing order --> private > Internal > External > Public

    // Potential call char tarah ka hota hai
    // 1. Mycontract --> jab ak contarct ke ander bane state variable and function jab usi contract me use hota hai to wah Mycontract ke ander ata hai

    // 2. DerivedContract is Mycontract --> jab ak contarct se koi dusra contract derived hota hai to wah child contract kahlata hai,
    //    esme parent contract ke state variable and function ka use child contract me kar sakte hai

    // 3. Another Contract --> jab ak hi file me 2 contract bana aur es dono contract ka koi aapas me relation nahi hai,
    //    lekin jab ak contract ka state variable and function kisi dusre contract me use karte hai

    // 4. outside world -->

    // private visibility --> eska use kewal Mycontract ke liye hi ho sakta hai i.e
    //  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

    // internal visibility --> eska use Mycontract and DerivedContract is Mycontract me hi kewal use kar sakte hai

    // external visibility --> eska use kabhi bhi state variable banane ke liye use nahi karte hai,
    //                         esaka use kewal AnotherContract and outside world me hi kiya ja sakta hai

    // public visibility --> eska use sabhi tarah ke potential calling me use kiya ja sakta hai

    // public and external visibility ke liye button banta hai

```

bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol | mappingBlockchain.sol | advanceMapping.sol | visibilityBlockchain.sol 1

```
    // Koi bhi state variable By default internal hota hai aur function By default public hota hai


    uint private x = 10; // only within the contract
    uint internal y = 100; // within the contract and derived contract
    uint public z = 1000; // everywhere

    function check1() private pure returns(string memory){    infinite gas
        return "private";
    }

    function check2() internal pure returns(string memory){    infinite gas
        return "internal";
    }

    function check3() external pure returns(string memory){    infinite gas
        return "external";
    }

    function check4() public pure returns(string memory){    infinite gas
        return "public";
    }

    function checkAll() public view returns(string memory){    infinite gas
        // return x;
        // return y;
        // return z;

        // return check1();
        // return check2();
        // return check3(); // show Error
        return check4();


    }
```

bytesInBlockchain.sol | dynamicBytesIn_Blockchain.sol | enumBlockchain.sol | structBlockchain.sol | mappingBlockchain.sol | advanceMapping.sol | visibilityBlockchain.sol 1

```
}


contract visibilityBlockchain2 is visibilityBlockchain1{

    // visibilityBlockchain2 child contract hai visibilityBlockchain1 ka

    // uint a = x; // show Error
    // uint a = y;
    // uint a = z;

    // string b = check1() // Show Error
    // string b = check2();
    // string b = check3(); // Show Error
    string b = check4();



}

contract visibilityBlockchain3{

    visibilityBlockchain1 obj = new visibilityBlockchain1();
    // uint public a = obj.y(); // Show Error
    // uint public a = obj.x(); // Show Error
    uint public a = obj.z();

    function anotherCaller() public view returns(string memory){    infinite gas
        return obj.check3();
    }
}
```

- Inheritance In Blockchain

ockchain.sol | structBlockchain.sol | mappingBlockchain.sol | advanceMapping.sol | visibilityBlockchain.sol | inheritenceBlockchain.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract inheritenceBlockchain1{
    function fun1() public pure returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain1";
    }

    function fun2() public pure returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain1";
    }

    function fun3() public pure virtual returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain1";
        // virtual keyword ki help se hum es function ko change kar sakte hai child contract me
        // virtual keyword se parent contract  se permission mil gya hai ki es function ko child contract me edit kar sakte hai
    }

    function fun4() public pure virtual returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain1";
    }
}


contract inheritenceBlockchain2 is inheritenceBlockchain1{

    function fun3() public pure override returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain2";

    }

    function fun4() public pure virtual override returns(string memory){   infinite gas
        return "i'm in contract inheritenceBlockchain3";
    }

}
```

```
contract inheritenceBlockchain3 is inheritenceBlockchain2{
    function fun4() public pure override returns(string memory){
        return "i'm in contract inheritenceBlockchain2";
    }
}
```

- Events In Blockchain

n.sol enumBlockchain.sol structBlockchain.sol mappingBlockchain.sol advanceMapping.sol visibilityBlockchain.sol inheritenceBlock

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract eventsBlockchain{

    // event ka use tab karte hai jab data ko blockchain me dalna ho
    event balance(address account, string message, uint val);

    function setData(uint _val) public{    infinite gas
        // jo simple type function ya transaction function hota hai wah kuchh bhi return nahi karta hai
        emit balance(msg.sender, "has value", _val);
        // jab koi setData button per click karega to data blockchain ke ander chala jayega,
        //us data me ether ka value hoga aur ye ether kiske doura send kiya ja raha hai uska account ka detail hoga aur hash valu

    }
}


contract chatApp{
    event chat(address indexed  _from, address _to, string message);

    function sendMess(address to_, string memory message_) public{    infinite gas
        emit chat(msg.sender,to_, message_);
    }
}
```

- Require In Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract requireBlockchain{
    // require ka use Error handling me kiya jata hai
    // require input validation me kam ata hai
    // require access control karta hai

    address public owner = msg.sender;
    uint public age = 25;

    function CheckRequire(uint _x) public{    infinite gas
        age = age + 5; // yaha per age ka value tab tak change nahi jab tak require wala condition true nahi hoga
        require(_x>2,"Value of x is less than 2"); // input validation
        // jab require wala condition false ho jata hai to gas ko return kar diya jata hai

    }
    function onlyOwner() public{    infinite gas
        require(owner == msg.sender,"You are note the owner");
        age = age-2;
    }

}
```

- Revert and Assert in Blockchain

nheritence.sol | PassingParameter.sol | callingParentFunction.sol | .prettierrc.json | requireBlockchain.sol | revertAssertBlockchain.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract revertAssertBlockchain{

    // Revert bhi require ki tarah hi kam karta hai, esme ak extra function hai ki esme custom error likh sakte hai
    // assert ka use bug check karne ke liye kiya jata hai aur security ke liye kiya jata hai
    address public owner = msg.sender;
    uint public age = 25;

    function CheckRequire(uint _x) public{    infinite gas
        age = age + 5; // yaha per age ka value tab tak change nahi jab tak require wala condition true nahi hoga
        require(_x>2,"Value of x is less than 2"); // input validation
        // jab require wala condition false ho jata hai to gas ko return kar diya jata hai

    }

    error throwError(string,address); // custom error

    function CheckRevert(uint _x) public {    infinite gas
        age = age +5;
        if(_x<2){
            // revert("value of x is less than 2");
            revert throwError("value of x is less than 2",msg.sender);
        }
    }

    function onlyOwner() public{    infinite gas
        // require(owner == msg.sender,"You are note the owner");

        if(owner!=msg.sender){
            revert("You are note the owner");
        }
        age = age-2;
    }

```

nheritence.sol | PassingParameter.sol | callingParentFunction.sol | .prettierrc.json | requireBlockchain.sol | revertAssertBlockchain.sol

```

    // Assert
    function checkOwnership() public view{    2341 gas
        assert(owner != 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4);
    }


}
```

0 listen on all transactions Search with transaction hash or address

- Function Modifier in Blockchain

```
ew  Window  Help
ingMultipleInheritence.sol   PassingParameter.sol   callingParentFunction.sol   .prettierrc.json   requireBlockchain.sol   revertAssertBlockchain.sol   functionModifierBlockchain.sol
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract functionModifierBlockchain{
     // Modifier ak special type function hai
     // modifier code reuse karne ki facility deta hai
    // ak contract ke ander bahut sara modifier ho sakte hai lekin constructor ak hi ho sakta hai

    modifier sameCode(){
        for(uint i=0; i<10; i++){
            //code
        }
        _;

        // code
    }

    function fun1() public pure sameCode returns(string memory){    infinite gas


        return "fun1 say hi";
    }

    function fun2() public pure sameCode returns(uint _x){    infinite gas


        _x = 20;
    }

    function fun3() public view sameCode returns(address){    infinite gas


        return msg.sender;
    }
```

```
ew  Window  Help
ingMultipleInheritence.sol   PassingParameter.sol   callingParentFunction.sol   .prettierrc.json   requireBlockchain.sol   revertAssertBlockchain.sol   functionModifierBlockchain.sol
    //

    address public owner = msg.sender;
    modifier onlyOwner(){
        require(owner == msg.sender, "Ypu are not the owner");

        _;
    }
    // jis jis function me onlyOwner ka use karenge us function ko kewal owner hi access kar sakta hai

    function startAuction() public view onlyOwner{    2622 gas


        // code
    }

    function stopAuction() public view onlyOwner{    2645 gas


        // code
    }

    function checkState() public view onlyOwner{    2688 gas


        // code
    }
```

```
    //

    uint public age = 30;
    modifier temp(uint _x){
        age = age+_x;
        _;
    }

    function changeAge(uint _y) public temp(_y){    infinite gas
        // sabse pahle changeAge function ke ander value aa raha hai phir uske bad temp modifier me pass ho ja raha hai,
        // uske bad modifier me age ka value change ho ja raha hai
    }

}
```

- Payable in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract payableBlockchain{
    // payable ke doura contract ke kisi bhi address aur function ko payable banaya ja sakta hai,
    // 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
    // kisi function ko payable banane ka matlab hai ki payable function ke doura hi kisi contarct per ether ko store kar sakte hai,
    // jo function payable nahi hai uske doura ether ko contract per store nahi kiya ja sakta hai

    // payable constructor
    constructor() payable{    103771 gas 79400 gas
        // eske doura deploy ke time hi kuchh ether ko contract per dal sakte hai
        // 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

    }

    // function payable
    function getEth() payable  public{    142 gas
        // es function ko view and pure nahi bana sakte hai kyuki direct blockchain me changing ho raha hai
    }

    function checkBal() public view returns(uint){    317 gas
        return address(this).balance;
        // ye function es contract me kitna ether hai usko return kar dega
        // this keyword se es contract ka address pata chalega
    }


    // Address payable
    address payable public owner = payable(msg.sender);
}
```

- Fallback and Receive in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;


/*

 1. It is executed when a non-existant function is called on the contract.
 2. It is required to be marked external.
 3. It has no name.
 4. It has No arguments
 5. It can not return any thing.
 6. It can be define one per contract.
 7. If not marked payable, it will throw exception if contract receives ether.
 8. It's main use is to directly send the ETH to contract

*/
contract fallbackReceiveBlockchain{

    event log(string _fun,address _sender,uint _val,bytes _data);

    fallback() external payable{
        // ye ether or data dono ko le sakta hai
        // receive kewal ether ko hi le sakta hai
        // es function ke help se direct hi es contract per ether ko get kiya ja sakta hai aur data bhi

        emit log("fallback", msg.sender, msg.value, msg.data);

        /*
            msg.sender, msg.value, msg.data ye predefine global variable hai
        */

    }
```

```
    receive() external payable{
        // 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
        // receive kabhi bhi data nahi le sakta ahi

        emit log("fallback", msg.sender, msg.value, "");
    }

    /*
        jab kabhi bhi fallback and receive dono ko ak hi sath define kar dete hai to us case me
        jab data+ether dono sath me send karenge to fallback ke doura accept kiya jayega
        lekin jab kewal ether ko send kiya jayega to receive de doura accept kiya jayega
    */


    function chechBal() public view returns(uint){
        return address(this).balance;
    }

}
```

- Send, Transfer, Call in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;


/*
      1. payable function, payable contructor,  fallback and receive ki help se hum kisi contract per ether ko store kar sakte hai
      2. Send, Transfer, Call ki help se hum ak contract se kisi dusre contract me ether ko send kar sakte hai
         aur eske help se ak contract se kisi dusre ke account me ether ko bhej sakte hai
      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
          jab operation fail ho jata hai to ye gas ko return bhi nahi karta hai esliye Transfer aur Call ko use kiya jata ahi
          jab opertaion fail ho jata hai to es operation ke douran jo bhi state variable me change hota hai to wah revert nahi ho pata hai
          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
           ye hamesha true ya false ko return karta hai; true --> success, false --> fail
       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
           hai. ye sara kam khud se hi kar deta hai

       5.  Call --> esme hum gas limit ko decide kar sakte hai.ye bool aur sath hi sath byte data ko return karta hai.
           eske sath bhi require ka use krna padta hi kyuki Send wala hi problem esme bhi hai

*/

contract SendTransferCallBlockchain{
    address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);

    receive() external payable{}    undefined gas

    function checkBal() public view returns(uint){    361 gas
        return address(this).balance;
    }

    function SEND() public{    infinite gas
       bool sent =  getter.send(1000000000000000000);
       require(sent,"tran is fail");
    }
```

```
    function TRANSFER() public{    infinite gas
        getter.transfer(1000000000000000000);
    }

    function CALL() public {    infinite gas
        (bool sent, ) = getter.call{value:1000000000000000000}("");

        require(sent,"tran is failed");
    }
}
```

- Send, Transfer, Call
  → address provided by user

lbackReceiveBlockchain.sol | Send_Transfer_Call_in_Blockchain/sendTransferCallBlockchain.sol | Send_Transfer_Call_in_Blockchain_2/sendTransferCallBlockchain.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;


contract sendTransferCall{
    // address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);

    receive() external payable{}    undefined gas

    function checkBal() public view returns(uint){    317 gas
        return address(this).balance;
    }

    function SEND(address payable getter) public{    undefined gas
       bool sent =  getter.send(1000000000000000000);
       require(sent,"tran is fail");
    }

    function TRANSFER(address payable getter) public{    undefined gas
        getter.transfer(1000000000000000000);
    }

    function CALL(address payable getter) public {    undefined gas
        (bool sent, ) = getter.call{value:1000000000000000000}("");

        require(sent,"tran is failed");
    }
}
```

0 listen on all transactions Search with transaction hash or address

ew Window Help

fallbackReceiveBlockchain.sol | Send_Transfer_Call_in_Blockchain/sendTransferCallBlockchain.sol | Send_Transfer_Call_in_Blockchain_2/sendTransferCallBlockchain.sol | sendTransferCall.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;


contract sendTransferCall{
    // address payable public getter = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);

    receive() external payable{}    undefined gas

    function checkBal() public view returns(uint){    317 gas
        return address(this).balance;
    }

    event log(uint value);

    function SEND(address payable getter) public payable{    undefined gas
        // es tarah se ether ak account se dusre account me sath hi sath ja raha ahi
       emit log(msg.value);
       bool sent =  getter.send(msg.value);
       require(sent,"tran is fail");
    }

    function TRANSFER(address payable getter) public payable{    undefined gas
        getter.transfer(msg.value);
    }

    function CALL(address payable getter) public payable{    undefined gas
        emit log(msg.value);
        (bool sent, ) = getter.call{value:msg.value}("");

        require(sent,"tran is failed");
    }
}
```

```
contract GetETH{
    receive() external payable{}    undefined gas

    function checkBal() public view returns(uint){    317 gas
        return address(this).balance;
    }
}
```

- Immutability in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract immubilityBlockchain{
    // Immutable state variable ko 2 hi jagah initialize kiya ja sakta hai first inline aur second me deploy ke time per
    // 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
    // kisi bhi data type ko immutable bana sakte hai
    // 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
    // constants variable ki value ko user ke doura set nahi karwa sakte hai
    // immutable variable ko user ke doura set karwa sakte hai

    // address public immutable owner = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;

    address public immutable owner;
    address public constant owner2 = address(1);
    address public owner3 = address(1);

    constructor(address _owner){    infinite gas 119200 gas
        owner = _owner;
    }

    function I() public view returns(address){    infinite gas
        return owner;
    }

    function C() public pure returns(address){    408 gas
        // jab koi variable constant rahta hai , usko read karne ke liye bhi function ka type pure hi hona chahiye
        return owner2;
    }

    function S() public view returns(address){    2500 gas
        return owner3;
    }

    // function of cost in increasing order: C > I > S
}
```

- Data Location - Storage, Memory and Calldata in Blockchain

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract storageMemoryCalldata{
    // storage ak box hai jo state variable ko store karta hai, aur ye storage khud blockchain per hota hai
    // 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
    /*
        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
        'memory' keyword ka use karna padta hai
    */
    // Calldata bhi RAM per hi hota hai
    /*
        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
        kyuki eska gas cost bahut kam hota hai
    */
    // jis data ko 'Calldata' me dal dete hai to us data ko change nahi kar sakte ahi
    // Gas cost in increasing order : Calldata > Memory > Storage

    uint[] public arr = [1,3,7,9,22];


    function Storage() public{    24465 gas
        uint[] storage arrs = arr;
        // arrs me arr ko photocopy nahi aya hai, arrs ke data ko change kerne per arr ka data change ho jayega
        arrs[3] = 99;
    }

    function Memeory() public view{    infinite gas
        uint[] memory arrm = arr;
        // arrm me arr ke data ka photocopy aya hai, arrm me data ko change karne se arr ke data chnge nahi hoga
        arrm[1] = 192;
    }

}
```

For Calldata

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract storageMemoryCalldata{
    function Memory(string memory str,uint[] memory arr) public{    infinite gas
        getInMem(arr);
        // getInCall(arr); // show Error
    }

    function Calldata(string calldata str, uint[] calldata arr) public{    infinite gas
        getInMem(arr);
        getInCall(arr);
        // Calldata se memory me pass ho sakta hai lekin memory se Calldata me pass nahi ho sakta hai
        // storage se storage pass ho sakta hai data
        // storage se memory me bhi pass ho sakta hai data
        // Calldata se Calldata me pass ho sakta hai data,
        // 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
        // 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
    }

    function getInMem(uint[] memory arr) public{    infinite gas

    }

    function getInCall(uint[] calldata arr) public{    474 gas

    }
}
```

- Inbuilt Cryptography hash function

sendTransferCall.sol | immubilityBlockchain.sol | storeageMemoryCalldata.sol | storageMemoryCalldata.sol | inbultCryptographicHashFunction.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract inbuiltCryptographicHashFunction{
    function hash1Keccak256(uint _x, string memory name, address _add) public pure returns(bytes32){    infinite gas
        return keccak256(abi.encodePacked(_x,name,_add));
    }

    function hash2Keccak256(uint _x, string memory name, address _add) public pure returns(bytes32){    infinite gas
        return keccak256(abi.encode(_x,name,_add));
    }

    function hashsha256(uint _x, string memory name, address _add) public pure returns(bytes32){    infinite gas
        return sha256(abi.encodePacked(_x,name,_add));
    }

    function hashRipemd160(uint _x, string memory name, address _add) public pure returns(bytes32){    infinite gas
        return ripemd160(abi.encodePacked(_x,name,_add));
    }
}
```

- Lottery Smart Contract

lockchain.sol | storeageMemoryCalldata.sol | storageMemoryCalldata.sol | inbultCryptographicHashFunction.sol | lotterySmartContract.sol

```python
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.8;

contract lotterySmartContract{
    address public manager;
    address payable[] public players;

    constructor(){    764039 gas 739000 gas
        manager = msg.sender;
    }

    function alreadyEntered() view private returns(bool){    infinite gas
        for(uint i=0; i<players.length; i++){
            if(players[i] == msg.sender)
            return true;
        }
        return false;
    }

    function enter() payable public{    infinite gas
        require(msg.sender!=manager,"Manager cannot enter");
        require(alreadyEntered() == false,"Player already entered");
        require(msg.value>=1 ether, "Minimun amount must be payed");

        players.push(payable(msg.sender));
    }

    function random() view private returns(uint){    infinite gas
        return uint(sha256(abi.encodePacked(block.difficulty,block.number,players)));
    }
```

```
    function pickwinner() public{    infinite gas
        require(msg.sender == manager,"Only manager can pick the winner");
        uint index = random()%players.length; // winner index
        address contractAddress = address(this);
        players[index].transfer(contractAddress.balance);
        players = new address payable[](0); // eske help se lottery reset ho gya hai; ab lottry phir se nikal sakte hai
    }

    function getPlayers() view public returns(address payable[] memory){    infinite gas
        return players;
    }
}
```

- Auction Smart Contract