Code Examples

All solidity practice codes

1. Hello World

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.0;
pragma abicoder v2;

contract HelloWorld{
    string public greet = "Hello World!";
}
```

2. First App

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Counter{
    uint public count = 0;
    function increaseCount() public{
        count += 1;
    }
    function decreaseCount() public{
        count -= 1;
    }
}
```

3. Primitives

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Primitives{
    //Data types
    /*
    1. unsigned integers (non-negative integers)
    2. signed integers(negative and positive integers)
    3. booleans
    4. address
    5. bytes
```

```
    */
    // 1. uint
    uint public x = 10;
    uint256 public y = 444;
    uint8 public w = 255;
    // uint8 public z = 256;   //error because it cant fit in uint8 range
i.e. 0 to 2^n - 1

    //2. int
    int public p = 55;
    int256 public q = 11;
    int public neg = -58;
    int8 public num = -128;
    // int8 public numb = -129;    //error because it cant fit in int8 range
i.e. -2^(n-1) to +2^(n-1) - 1

    //int and uint only themselves refer to int256 and uint256 respectively.

    //3. boolean
    bool on = true;
    bool off = false;

    //4. address
    address public addr = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4;
    // 40 characters, prefixed by 0x, takes 20 bytes or 160 bits or 40 hex
characters
    // 40 hex chars * 4 bits = 160 bits i.e. 20 bytes as 1 byte = 8 bits.
    // It corresponds to the last 20 bytes of the Keccak-256 hash of the
public key.

    // 5. bytes
    bytes32 public xyz =
0x6162636400000000000000000000000000000000000000000000000000000000;

    //notes
    uint umin = type(uint).min;
    uint umax = type(uint).max;

    int imin = type(int).min;
    int imax = type(int).max;

    //default values;
    uint public duint;  //0
```

```
    int public dint;    //0
    bool public dbool;  //false
    address public daddr;   //0x0000000000000000000000000000000000000000
    bytes32 public dbyte;
//0x0000000000000000000000000000000000000000000000000000000000000000

}
```

4. Variables

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Variables{
    /*
        1. State variables - stored in the blockchain, requires gas fee
        2. Local variables - local with the scope of a function, doesnot
require gas fee
        3. Global variables - exist in global workspace, provides
information about the blockchain and the transaction process



    */
    // state variables
    uint public state = 145;
    string public hello = "Hello";

    // local variables
    function setLocal() public pure returns(uint){
        uint local = 5;
        return local;
    }



    // global variables
    uint public timestamp = block.timestamp;
    uint public difficulty = block.difficulty;
    address public sender = msg.sender; // address of the calle

}
```

5. Constants

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.7;

contract Constants{
    uint public constant NUM = 13;  //convention to make constant uppercase
    //saves gas fee, hardcode
}
```

6. Immutables

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.7;

contract Immutables{
    address public immutable owner;   //saves gas
    //immutables are like constant but you can initialize them only one time
at the time of
    //deployment of the contract.
    constructor(){
        owner = msg.sender;
    }
}
```

7. Read state

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.7;

contract ReadState{
        //state variable
        uint public myNum;
        //writing or updating the state variable you have to send a
transacion
        //hence it requires gas.
        function setNum(uint _myNum) public{
            myNum = _myNum;
        }

        //Reading a state variable doesnot reqiure gas.
        function readNum() public view returns(uint){
            return myNum;
        }
}
```

8. Wei and Ether

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.7;

contract WeiEther{
    uint public constant OneWei = 1 wei;
    uint public constant OneEth = 1 ether;

    function checkOneWei() public pure returns(bool){
        return 1 wei == 1;
    }
    function checkOneEther() public pure returns(bool){
        return 1 ether == 1e18 wei;
    }
}
```

9. Gas

```
//SPDX-License-Identifier:MIT
pragma solidity ^0.8.7;

contract Gas{
    uint public i;
    function forever() public{
        while(true){
            i += 1;
        }
    }
}

//Above function is forever running function .
```

10. If Else

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract IfElse{
    function GreaterORLess(uint _x) public pure returns(uint){
        if(_x < 10){
            return 0;
        }
        else if(_x == 10){
            return 1;
        }
```

```
        else{
            return 2;
        }
    }
    //ternary operator
    function ternaryOP(uint _y) public pure returns(uint){
        return _y > 10 ? 1 : 2;
    }
}
```

11. Loops

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Loop{
    uint public count = 0;
    uint public clicks = 1;
    function loop() public {
        for(uint i = 0; i < 10; i++){
            if(i == 3)  continue;
            else if(i == 10) break;
            else count++;

        }
    }
    function While() public{
        uint x = 0;
        while(x < 5){
            clicks *= 2;
            x++;
        }
    }
}
```

12. Mapping

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

// // contract Mapping{
// //     mapping(address => uint) public balances;   //simple mapping
// //     mapping(address => mapping(address=>bool)) public isFriend;
//nested mapping
```

```
// //     function examples() external{
// //         balances[msg.sender] = 123;
// //         uint bal = balances[msg.sender];
// //         uint bal2 = balances[address(1)];

// //         balances[msg.sender] += 456; //123 + 456 = 579

// //         delete balances[msg.sender]; //reset to uint default 0
// //     }
// // }

// contract Mapping{
//     //mapping from addresses to uint
//     mapping(address => uint) public balances;
//     //get the balance of an address
//     function getbalance(address addr) public view returns(uint){
//         return balances[addr];

//     }
//     //set balance of an address
//     function setbalance(uint _balance, address _addr) public {
//         balances[_addr] = _balance;
//     }
//     //reset balance of an address
//     function resetbalance(address _addr) public{
//         delete balances[_addr];
//     }
// }

// contract NestedMapping{
//     //mapping to mapping
//     mapping(address => mapping(uint=>bool)) public nested;

//     //get
//     function Getnested(address _addr,uint _i) public view returns(bool){
//         return nested[_addr][_i];
//     }
//     //set
//     function Setnested(address _addr, uint _i,bool _boo) public {
//         nested[_addr][_i] = _boo;
//     }
// }
```

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

contract Mapping {
    // Mapping from address to uint
    mapping(address => uint) public myMap;

    function get(address _addr) public view returns (uint) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return myMap[_addr];
    }

    function set(address _addr, uint _i) public {
        // Update the value at this address
        myMap[_addr] = _i;
    }

    function remove(address _addr) public {
        // Reset the value to the default value.
        delete myMap[_addr];
    }
}

contract NestedMapping {
    // Nested mapping (mapping from address to another mapping)
    mapping(address => mapping(uint => bool)) public nested;

    function get(address _addr1, uint _i) public view returns (bool) {
        // You can get values from a nested mapping
        // even when it is not initialized
        return nested[_addr1][_i];
    }

    function set(
        address _addr1,
        uint _i,
        bool _boo
    ) public {
        nested[_addr1][_i] = _boo;
    }
```

```
    function remove(address _addr1, uint _i) public {
        delete nested[_addr1][_i];
    }
}
```

13. Array

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

contract Array{
    uint[] public arr;

    //insert into an array
    function set(uint _val) public{
        arr.push(_val);
    }

    //get array element
    function get(uint index) public view returns(uint){
        return arr[index];
    }

    //get whole array
    function getArr() public view returns(uint[] memory){
        return arr;
    }

    //get size of the array
    function getlength() public view returns(uint){
        return arr.length;
    }

    //remove element from the array
    function remove() public {
        arr.pop();
    }

    //delete an element at an index
    function remove(uint _i) public {
        delete arr[_i];     //resets default uint value of 0
                            //size of the array doesnot change when delete
is called
    }
```

```
    //create a fixed sized array in memory
    function fmarray() public{
        uint[] memory a = new uint[](5);
    }

}
```

14. Enum

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

contract Enum{
    enum Status{
        None,
        Pending,
        Shipped,
        Completed,
        Rejected,
        Canceled
    }

    Status public status;
    function get() public view returns(Status){
        return status;
    }
    function set(Status _status) public{
        status = _status;
    }
    function Pending() public{
        status = Status.Pending;
    }
    function Cancel() public{
        status = Status.Canceled;
    }
    function reset() public{
        delete status;
    }
}
```

15. Struct

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

contract Struct{
    struct Car{
        string model;
        uint year;
        address owner;
    }
    //single instance
    Car public car;
    //multiple instances - array
    Car[] public cars;
    //mapping carsbyowners
    mapping(address => Car[]) public carsByOwners;

    //examples
    function examples() external{
        //initializing structs
        Car memory toyota = Car("Toyota",1990,msg.sender);  //parameters
need to follow the order
        //initialization using key value pairs
        Car memory lambo =
Car({model:"Lamborghini",owner:msg.sender,year:1980});    //this way we dont
have to follow the order
        //struct will have default value if not given explicitly
        Car memory tesla;   //defaults - string "",uint 0,address
0x00112....
        tesla.model = "Tesla";
        tesla.year = 2010;
        tesla.owner = msg.sender;

        //storing above cars into cars array
        cars.push(toyota);
        cars.push(lambo);
        cars.push(tesla);

        //we dont always have to first create instance in memory and then
push into the array
        cars.push(Car("Ferrari",1985,msg.sender));

        //getting structs
        Car memory _car = cars[0];
```

```
        _car.model;
        _car.year;
        _car.owner;
        //modifying struct members' data
        Car storage _car_ = cars[1];
        _car_.model = "Tata";
        _car_.year = 1960;
        //we can use delete to reset data in struct
        delete _car_.owner;

        //detele a member in a struct
        //delete cars[2];
    }

}
```

16. Error

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

contract Errorr{
    //require
    function testrequire(uint _i) public pure{
        require(_i <= 10, "Number is greater than 10!");
    }
    //revert
    function testrevert(uint _i) public pure{
        if(_i > 10){
            revert("Number is greater than 10");
        }
    }
    //assert
    uint public num = 123;
    function testassert() public view{
        //accidental update of state variable num.
        assert(num == 123);
    }
    function foo(uint _i) public{
        num += 1;
        require(_i < 10);
    }
    //custom error
    error myError(uint _p);
```

```
    function testcustomerror(uint _p) public pure{
        if(_p > 10){
            revert myError(_p);
        }
    }

}
```

17. Modifiers

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

contract Modifier{
    uint public count;
    bool public paused;

    function setPause(bool _pause) public {
        paused = _pause;
    }

    modifier whenNotPaused(){
        require(!paused,"Paused");
        _;
    }

    function inc() public whenNotPaused{
        count += 1;
    }
    function dec() public whenNotPaused{
        count -= 1;
    }
}
```

18. Events

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

contract Event {
    // Event declaration
    // Up to 3 parameters can be indexed.
    // Indexed parameters helps you filter the logs by the indexed parameter
```

```
    event Log(address indexed sender, string message);
    event AnotherLog();

    function test() public {
        emit Log(msg.sender, "Hello World!");
        emit Log(msg.sender, "Hello EVM!");
        emit AnotherLog();
    }
}
```

19. Constructor

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

contract Constructor{
    uint public x;
    address public owner;

    constructor(uint _x){
        owner = msg.sender;
        x = _x;
    }
}
//constructor initializes the state variables
```

20. Inheritance

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


/*
        A
       / \
      B   C
     /\   /\
    D  E F  G


*/


contract A{
```

```
    function foo() public pure returns(string memory){
        return "foo - A";
    }

    function bar() public pure virtual returns(string memory){
        return "bar - A";
    }
}

//lets inherit A to contract B

contract B is A{
    //override bar function of contract A
    function bar() public pure virtual override returns(string memory){
        return "bar - B";
    }
}

//virtual keyword specifies that the function is inheritable and can be
customized by the
//other contract

contract C is B{
    function bar() public pure virtual override returns(string memory){
        return "bar - C";
    }
}

contract D is B,C{
    function bar() public pure virtual override(B,C) returns(string memory){
        return super.bar();     //returns bar - C
    }
}



/*in case of multiple inheritance like
        B
        |
    ----|
    |   |
    C   |
    |   |
    ----D
```

```
      |
      |
      |
      E

in contract E while inheriting C and D, we most consider the order
here C is the most base like constructor as it inherits only B while
D inherits 2 contracts C and B. So C should be inherited first and is done
by
mentioning it first after "is" keyword.
*/
contract E is C,D{
    function bar() public pure override(C,D) returns(string memory){
        return super.bar();     //bar - C is coming from the execution of
bar function in D that return bar - C.
    }
}
```

21. Shadowing Inherited

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

contract A{
    string public message = "state variable";
    function get() public view returns(string memory){
        return message;
    }
}

contract B is A{
    /*
    string public message = "new from B";
    above line of code generates error because it is not allowed to override
the state variable
    */

    //correct way
    constructor(){
        message = "Correct overriding of state variable";

    }
}
```

22. Visibility

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

contract Base{
    string private privateVar = "private variable";
    string internal internalVar = "internal variable";
    string public publicVar = "public variable";

    function privateFunc() private pure returns(string memory){
        return "Private function test";
    }

    function publicFunc() public pure returns(string memory){
        return "Public function test";
    }

    function internalFunc() internal pure returns(string memory){
        return "Internal function test";
    }

    function externalFunc() external pure returns(string memory){
        return "External function test";
    }

    // function testExternalFunc() public view returns(string memory){
    //    return externalFunc();
    // }
}

contract Child is Base{
    function example() public view returns(string memory){
        internalFunc();
        publicFunc();
        return internalVar;
    }
}

// pragma solidity ^0.8.13;

// contract Base {
//     // Private function can only be called
```

```
//     // - inside this contract
//     // Contracts that inherit this contract cannot call this function.
//     function privateFunc() private pure returns (string memory) {
//         return "private function called";
//     }

//     function testPrivateFunc() public pure returns (string memory) {
//         return privateFunc();
//     }

//     // Internal function can be called
//     // - inside this contract
//     // - inside contracts that inherit this contract
//     function internalFunc() internal pure returns (string memory) {
//         return "internal function called";
//     }

//     function testInternalFunc() public pure virtual returns (string
memory) {
//         return internalFunc();
//     }

//     // Public functions can be called
//     // - inside this contract
//     // - inside contracts that inherit this contract
//     // - by other contracts and accounts
//     function publicFunc() public pure returns (string memory) {
//         return "public function called";
//     }

//     // External functions can only be called
//     // - by other contracts and accounts
//     function externalFunc() external pure returns (string memory) {
//         return "external function called";
//     }

//     // This function will not compile since we're trying to call
//     // an external function here.
//     // function testExternalFunc() public pure returns (string memory) {
//     //     return externalFunc();
//     // }

//     // State variables
```

```
//     string private privateVar = "my private variable";
//     string internal internalVar = "my internal variable";
//     string public publicVar = "my public variable";
//     // State variables cannot be external so this code won't compile.
//     // string external externalVar = "my external variable";
// }

// contract Child is Base {
//     // Inherited contracts do not have access to private functions
//     // and state variables.
//     // function testPrivateFunc() public pure returns (string memory) {
//     //     return privateFunc();
//     // }

//     // Internal function call be called inside child contracts.
//     function testInternalFunc() public pure override returns (string
memory) {
//         return internalFunc();
//     }
// }
```

23. Interface

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

interface ICounter{
    function count() external view returns(uint);
    function increaseCount() external;
    function decreaseCount() external;
}

contract Interface{
    function Increment(address _counter) external{
        ICounter(_counter).increaseCount();
    }
    function getCount(address _counter) external view returns(uint){
        return ICounter(_counter).count();
    }
    function Decrement(address _counter) external{
        ICounter(_counter).decreaseCount();
    }
}
```

24. Payable

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

contract Payable{
    address payable public owner;

    constructor(){
        owner = payable(msg.sender);    //we need to typecast the address to
address payable.
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
    function deposit() public payable {}

    }




// pragma solidity ^0.8.13;

// contract Payable {
//     // Payable address can receive Ether
//     address payable public owner;

//     // Payable constructor can receive Ether
//     constructor() payable {
//         owner = payable(msg.sender);
//     }

//     // Function to deposit Ether into this contract.
//     // Call this function along with some Ether.
//     // The balance of this contract will be automatically updated.
//     function deposit() public payable {}

//     // Call this function along with some Ether.
//     // The function will throw an error since this function is not
payable.
//     function notPayable() public {}
```

```
//     // Function to withdraw all Ether from this contract.
//     function withdraw() public {
//         // get the amount of Ether stored in this contract
//         uint amount = address(this).balance;

//         // send all Ether to owner
//         // Owner can receive Ether since the address of owner is payable
//         (bool success, ) = owner.call{value: amount}("");
//         require(success, "Failed to send Ether");
//     }

//     // Function to transfer Ether from this contract to address from
input
//     function transfer(address payable _to, uint _amount) public {
//         // Note that "to" is declared as payable
//         (bool success, ) = _to.call{value: _amount}("");
//         require(success, "Failed to send Ether");
//     }
// }
```

25. SendEther

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



contract ReceiveEther{
    receive() external payable {}   //Function to receive Ether msg.data
must be empty
    fallback() external payable {}  //Fallback function is called when
msg.data is not empty

    //get balance of this address
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
}

contract SendEther{
    function sendViaTransfer(address payable _to) public payable{
        _to.transfer(msg.value);    //no longer recommended for sending
ether
    }
```

```
    function sendViaSend(address payable _to) public payable{
        bool sent = _to.send(msg.value);
        require(sent,"Failed to send Ether");
    }
    function sendViaCall(address payable _to) public payable{
        (bool sent,bytes memory data) = _to.call{value:msg.value}("");
        require(sent,"Failed to send Ether");
    }
}
```

26. Fallback

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

contract Fallback{
    event Log(string func,uint gas);
    //Fallback function must be declared as external.
    fallback() external payable {
        emit Log("fallback",gasleft());
    }
    receive() external payable {
        emit Log("receive",gasleft());
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
    }
}

contract SendToFallback{
    function transferToFallback(address payable _to) public payable{
        _to.transfer(msg.value);
    }
    function callFallback(address payable _to) public payable{
        (bool sent,) = _to.call{value:msg.value}("");
        require(sent,"Failed to send Ether");
    }
}
```

27. Call

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract TestCall{
```

```
    string public message;
    uint public x;

    event Log(string message);

    fallback() external payable {
        emit Log("Fallback was called");

    }
    receive() external payable{}

    function foo(string memory _message, uint _x) external payable
returns(bool, uint){
        x = _x;
        message = _message;
        return (true,999);
    }
}

contract Call{
    bytes public data;
    function callFoo(address _test) external payable{
        (bool success,bytes memory _data) = _test.call{value:111}(
            abi.encodeWithSignature("foo(string,uint256)","call foo",123)
        );
        require(success,"Call Failed");
        data = _data;
    }
    function callDoesnotExist(address _test) external {
       (bool success,) =
_test.call(abi.encodeWithSignature("doesnotexist()"));
       require(success,"Call Failed");
    }
}




contract Receiver {
    event Received(address caller, uint amount, string message);

    fallback() external payable {
        emit Received(msg.sender, msg.value, "Fallback was called");
```

```
    }

    receive() external payable{}
    function foo(string memory _message, uint _x) public payable returns
(uint) {
        emit Received(msg.sender, msg.value, _message);

        return _x + 1;
    }
}

contract Caller {
    event Response(bool success, bytes data);

    // Let's imagine that contract Caller does not have the source code for
the
    // contract Receiver, but we do know the address of contract Receiver
and the function to call.
    function testCallFoo(address payable _addr) public payable {
        // You can send ether and specify a custom gas amount
        (bool success, bytes memory data) = _addr.call{value: msg.value,
gas: 5000}(
            abi.encodeWithSignature("foo(string,uint256)", "call foo", 123)
        );

        emit Response(success, data);
    }

    // Calling a function that does not exist triggers the fallback
function.
    function testCallDoesNotExist(address _addr) public {
        (bool success, bytes memory data) = _addr.call(
            abi.encodeWithSignature("doesNotExist()")
        );

        emit Response(success, data);
    }
}
```

28. Delegate Call

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

```
contract B{
    uint public num;
    address public sender;
    uint public value;

    function setVars(uint _num) public payable{
        num = _num;
        sender = msg.sender;
        value = msg.value;
    }
}

contract A{
    uint public num;
    address public sender;
    uint public value;

    function setVars(address payable _addr,uint _num) public payable{
        // (bool success,bytes memory data) = _addr.delegatecall(
        //     abi.encodeWithSignature("setVars(uint256)",_num)
        // );
        (bool success,bytes memory data) = _addr.delegatecall(
            abi.encodeWithSelector(B.setVars.selector,_num)
        );
        require(success,"Delegate call failed");
    }


}
```

29. Function Selector

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

contract FunctionSelector{
    function getSelector(string calldata _func) external pure
returns(bytes4){
        return bytes4(keccak256(bytes(_func)));
    }
}

contract Receiver{
    event Log(bytes data);
```

```
    function transfer(address _to,uint amount) external {
        emit Log(msg.data);
    }
}

contract SayHello{
    event Log(bytes data);
    function sayHelloTo(string memory _name) external{
        emit Log(msg.data);
    }
    //
0x6c0ff76800000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000075072616a77616c0000000000000000000000000000000000000000000000000000000000
}

// 0xa9059cbb
// 0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4
// 000000000000000000000000000000000000000000000000000000000000007b
```

30. Import

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

import "./30_Foo.sol";

contract Bar{
    Foo public foo = new Foo();
    function getSum(uint x,uint y) public view returns(uint){
        return foo.sum(x,y);
    }
}

// // we can import via link as well
// import "github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol";

// contract MyToken is ERC20 {}
```

31. Library

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

```
library Math{
    function max(uint x, uint y) internal pure returns(uint){
        return x >= y ? x : y;
    }
}

library Arraylib{
    function find(uint[] storage arr,uint element_) internal view
returns(uint){
        for(uint i = 0; i < arr.length; i++){
            if(arr[i] == element_){
                return i;
            }
        }
        revert("Element not found in the array");
    }
}

contract TestLibrary{
    function findMax(uint x, uint y) public pure returns(uint){
        return Math.max(x,y);
    }
}

contract TestArraylib{
    using Arraylib for uint[];
    uint[] public myarr = [1,5,3,6];
    function Search(uint _x) public view returns(uint){
    //    return Arraylib.find(myarr,_x);
            return myarr.find(_x);
    }
}
```

32. ABI Decode

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

contract AbiDecode{
    struct MyStruct{
        string name;
        uint[2] nums;
```

```
    }
    function encode(
        uint x,
        address addr,
        uint[] calldata arr,
        MyStruct calldata mystruct
    ) external pure returns(bytes memory){
        return abi.encode(x,addr,arr,mystruct);
    }

    function decode(bytes calldata data) external pure returns(
        uint x,
        address addr,
        uint[] memory arr,
        MyStruct memory mystruct

    ){
        (x,addr,arr,mystruct) = abi.decode(data,
(uint,address,uint[],MyStruct));
    }
}
```

33. Hashing

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

contract Hash{
    function hash(uint x,string memory text,address addr) public pure
returns(bytes32){
        return keccak256(abi.encodePacked(x,text,addr));
    }
}

contract HashFunction {
    function hash(
        string memory _text,
        uint _num,
        address _addr
    ) public pure returns (bytes32) {
        return keccak256(abi.encodePacked(_text, _num, _addr));
    }
```

```
// Example of hash collision
// Hash collision can occur when you pass more than one dynamic data type
// to abi.encodePacked. In such case, you should use abi.encode instead.
function collision(string memory _text, string memory _anotherText)
    public
    pure
    returns (bytes32)
{
    // encodePacked(AAA, BBB) -> AAABBB
    // encodePacked(AA, ABBB) -> AAABBB
    return keccak256(abi.encodePacked(_text, _anotherText));
}

}
```

contract GuessTheMagicWord {
bytes32 public answer =
0x60298f78cc0b47170ba79c10aa3851d7648bd96f2f8e46a19dbc777c36fb0c00;

```
// Magic word is "Solidity"
function guess(string memory _word) public view returns (bool) {
    return keccak256(abi.encodePacked(_word)) == answer;
}

}
```