Basic

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Hello {

string public greet;

constructor(){

greet = "Hey there!";

}

// memory use to store temp data

function setGreet (string memory newGreet) public {

greet = newGreet;

}

}

Prime number

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract PrimeNumber {

// memory use to store temp data

// pure defines that function will not modify code

// function checkIsPrimeNumber (int256 number) public pure returns (bool) {

// if (number <= 1) {

// return false;

// }

// for (int256 i = 2; i <= number/2; ++i) {

// if (number % i == 0) {

// return false;

// }

// }

// return true;

// }

function checkIsPrimeNumber (int256 number) public pure returns (string memory) {

if (number <= 1) {

return "Not Prime";

}

for (int256 i = 2; i <= number/2; ++i) {

if (number % i == 0) {

return "Not Prime";

}

}

return "Is Prime";

}

}

Variables

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Variables {

int256 public signNum;

bool public isTrue;

address public walletAddress;

uint256[3] public fixArray;

string public message;

enum STATUS { pending, approved } // return index of enums instead of actual value, starting from 0.

STATUS public currentStatus;

constructor() {

signNum = -45;

isTrue = true;

walletAddress = msg.sender; // wallet address

fixArray = [1, 2, 3];

currentStatus = STATUS.pending; // return index of emum

message = "Initial message!";

}

function updateVariables() public {

signNum = -20;

isTrue = false;

walletAddress = msg.sender; // wallet address

fixArray[2] = 4; // update value of 3rd element in an array

currentStatus = STATUS.approved; // return index of emum

message = "Updated message!";

}

}

Operations

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Maths {

uint256 public num1;

uint256 public num2;

uint256 public result;

function setNumber(uint256 \_num1, uint256 \_num2) public {

num1 = \_num1;

num2 = \_num2;

}

function add() public returns (uint256) {

result = num1 + num2;

return result;

}

function divide() public returns (uint256) {

require(num2 != 0, "Oops! Can't divide a number by 0.");

result = num1 / num2;

return result;

}

}

Decision making

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Variables {

uint256 public num;

function setNumber(uint256 \_num) public {

num = \_num;

}

function check() public view returns (string memory) {

if(num % 2 == 0) {

return string.concat("The number entered ", num, " is even.");

return string(abi.encodePacked("The number entered ", num, " is even."));

}

return string(abi.encodePacked("The number entered ", num, " is odd."));

}

}

// convert number to string in retrun pending

While loop

Mod

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Mod {

function addMod() public pure returns (uint256) {

return addmod(4, 5, 3);

}

function mulMod() public pure returns (uint256) {

return mulmod(4, 5, 3);

}

}

Generate keccak hash

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract GenerateKeccakHash {

bytes32 private message;

function setHash(string memory \_message) public {

message = keccak256(bytes(\_message));

}

function getHash() public view returns (bytes32) {

return message;

}

}

Random number

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract RandomNumber {

uint randomNumber = 0;

function setRandomNumber() public {

randomNumber = uint (keccak256(abi.encodePacked(msg.sender, randomNumber)));

}

function getRandomNumber() public view returns (uint) {

return randomNumber;

}

}

FallBack

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract FallBbackFunction {

// fallback declaration as external always its declared with out name, parameter and does not return any value.

string public call;

fallback() external payable {

call = "I am a fallback function!";

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

contract Sender {

function transfer() public payable {

require(msg.value >= 2 ether, "Insufficient ether");

(bool sent,) = payable (0xd16B472C1b3AB8bc40C1321D7b33dB857e823f01).call{value: 2 ether}("Transaction completed!");

require(sent, "Transaction failed!");

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

}

Function Overloading

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract FunctionOverloading {

function getSum(uint x, uint y) public pure returns (uint256) {

return x + y;

}

function getSum(uint x, uint y, uint z) public pure returns (uint256) {

return x + y + z;

}

}

Contracts

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract GetLoan{

// no use, use get or set to interact with other contract

function TakeLoan() external payable {

GiveLoan loan = new GiveLoan(1000);

}

}

contract GiveLoan{

uint public money;

constructor (uint \_amount) public {

money = \_amount;

}

}

Single inheritance

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract Parent {

uint internal sum;

function setVal() external {

uint a = 50;

uint b = 20;

uint c = 20;

sum = a + b + c;

}

}

contract child is Parent {

function getVal() external view returns(uint){

return sum;

}

}

contract caller {

child cc = new child();

function Inherit() public returns(uint){

cc.setVal();

return cc.getVal();

}

}

multiple inheritance

// SPDX-License-Identifier: MIT

pragma solidity 0.8.24;

contract A {

string internal x;

string a = "Hey " ;

string b = "there";

function getA() external{

x = string(abi.encodePacked(a, b));

}

}

contract B is A {

string public y;

string c = "harshal!";

function getB() external payable returns (

string memory){

y = string(abi.encodePacked(x, c));

}

}

contract C is B {

function getC() external view returns(

string memory){

return y;

}

}

contract caller {

C cc = new C();

function testInheritance(

) public returns (

string memory) {

cc.getA();

cc.getB();

return cc.getC();

}

}

Abstract: It is a contract that works in a class without implementation

is – relationship

has – association

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;

contract A {

function getResult() public view returns(uint);

}

contract B is A {

function getResult() public view returns (uint) {

uint a = 100;

uint b = 210;

uint result = a \* b;

return result;

}

}

Interface

* We can't use constructor in interface
* Only external function

// SPDX-License-Identifier: MIT

pragma solidity ^0.5.17;

interface A {

function getResult() external view returns(uint);

}

contract B is A {

function getResult() external view returns (uint) {

uint a = 200;

uint b = 100;

uint result = a - b;

return result;

}

}

Library

* Reuse certain code
* Stateless entity, never alter or modify state of contract
* Reduces Gas burn

Util.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

library Util {

function concat(string memory x,string memory y) public pure returns (string memory) {

return string.concat(x, y);

}

function add(uint x, uint y) public pure returns (uint) {

return x + y;

}

}

Test.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

import "util.sol";

contract ImportHere {

using Util for uint;

using Util for string;

function sum(uint a, uint b) public pure returns (uint) {

return a.add(b);

}

function concat(string memory x,string memory y) public pure returns (string memory) {

return x.concat(y);

}

}

Error Handling

a.Require:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract requireStatement {

function checkInput(uint \_input) public pure returns(string memory){

require(\_input >= 0, "invalid uint8");

require(\_input <= 255, "invalid uint8");

return "Input is Uint8";

}

function odd(uint \_input) public pure returns(bool){

require(\_input % 2 != 0);

return true;

}

}

b.Assert:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract assertStatement {

bool result;

function checkOverflow(uint \_num1, uint \_num2) public {

uint sum = \_num1 + \_num2;

assert(sum<=255);

result = true;

}

function getResult() public view returns(string memory){

if(result == true){

return "No Overflow";

}

else{

return "Overflow exist";

}

}

}

c.Revert:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract revertStatement {

function checkOverflow(uint \_num1, uint \_num2) public pure returns(string memory, uint){

uint sum = \_num1 + \_num2;

if(sum < 0 || sum > 255){

return ("Overflow", sum);

revert(" Overflow Exist");

}

else{

return ("No Overflow", sum);

}

}

}

2. Mapping

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.24;

contract LedgerBalance{

mapping(address => uint) balance;

mapping(address => string) name;

function updateBalance() public returns(uint){

balance[msg.sender]=20;

return balance[msg.sender];

}

function senderInfo() public returns(string memory){

name[msg.sender] = "Sameera";

return name[msg.sender];

}

function printSender() public view returns(address){

return msg.sender;

}

}