Solidity

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec1 {

/\*

변수 : 값을 지속적으로 변경 할 수 있음.

상수 : 한번 넣은 값은 변경 불가.

자료형 가시성 지정자 변수명 = 값

uint public a = 3;

a=4;

uint public constant b=3; -> 상수

b=4 -> 에러남

자료형 타입

값 타입 Vs 참조 타입

값 타입 : uint, int, bool , address (고정 길이), bytes1...bytes32 (고정 길이)

참조 타입 : bytes(동적 길이),string (동적 길이), array(배열), mapping (매핑), struct (구조체)

int : 기호있는 integer

int8 : -2^7 ~ 2^7-1

int16: -2^15~2^15-1

int32: -2^31~2^31-1

int64: -2^63~2^63-1

int128 : -2^127~2^127-1

int256 (=int): -2^255~2^255-1

uint: 기호없는 integer

uint8 : 0~2^8-1

uint16: -0~2^16-1

uint32: -0~2^32-1

uint64: -0~2^64-1

uint128 : -0~2^128-1

uint256 (=uint): 0~2^256-1

bool : true or false

산술연산자 : + - \* / \*\*

논리 연산자 : &&, ||

비교연산자: <,>,!=,==, >= , <=

\*/

uint public a = 1;

int public b = -1;

bool public c = true;

bytes1 public d = hex"11"; //1 바이트는 8비트고 하나의 문자는 4비트이다

string public e = "abcd";

//산술 연산자 : +,-,\*,/

uint public f = 1+2;

uint public g = 3-2;

uint public h = 4\*2;

uint public i = 4/2;

uint public j = 4\*\*2;

//논리 연산자 : &&, ||

bool public k = true && true; // true

bool public m = true && false; // false

bool public n = false && false; //false

bool public l = true || true; // true

bool public o = true || false; // true

bool public p = false || false; // false

// 비교연산자

bool public q = 3>5; //false

bool public r = 3<=4; // true

bool public s = 3==3; // true

bool public t = 3!=3; // false

uint public constant z=3; // 상수

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec2 {

/\*

가시성 지정자

가시성 지정자는 변수, 상수, 함수 적용 가능.

public : 외부/내부 접근가능 ( 변수 적용시 getter 함수 생성)

external : 외부 접근만 가능 ( 변수 적용 불가, this 키워드 사용시 내부 접근 가능)

private : 내부 접근만 가능

internal : 내부 접근만 가능하나, 상속 받은 스마트 컨트랙트는 외부 접근가능

함수 만들수 있는 경우의 수

function 함수명() public {

//로직

}

1. 매개변수와 반환값이 없음

function fun1() public {

a = 5;

}

2. 매개변수는 있지만 반환값이 없음

function fun2(uint num) public{

a = num;

}

3. 매개변수는 없지만 반환값이 있음

function fun3() public returns(uint){

a = 3;

return a;

}

4. 매개변수와 반환값이 있음

function fun4(uint num) public returns(uint){

a = a + num;

return a;

}

\*/

uint public a = 1;

function fun1() public {

a = 5;

}

function fun2(uint \_num) external {

a = \_num;

}

function fun2\_1() external {

this.fun2(5);

}

function fun3() private returns(uint) {

a = 3;

return a;

}

function fun3\_1() public returns(uint) {

return fun3();

}

function fun4(uint \_num) internal returns(uint){

a = a + \_num;

return a;

}

function fun4\_1() public returns(uint) {

return fun4(55);

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec4 {

/\*

event

- 다른 프로그래밍 언어의 print 함수처럼 특정한 값을 출력

- 블록체인의 특정 블록에 값을 저장.

- 함수 내부에서만 사용 가능, emit 키워드 사용.

- event 이벤트명(자료형 변수명);

event info(string name, uint256 money);

\*/

event info(string name, uint256 money);

function fun(string memory name, uint256 money) public {

emit info(name,money);

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec5 {

/\*

매핑

- 키와 값으로 구성

- length(길이) 없음

- mapping(키 자료형 => 값 자료형) 가시성 지정자 변수명;

mapping(uint => uint) public a;

\*/

// mapping(address => uint) public a;

// function addMapping(address \_key, uint \_value) public {

// a[\_key] = \_value;

// }

// function getMapping(address \_key) public view returns(uint) {

// return a[\_key];

// }

// function deleteMapping(address \_key) public {

// delete(a[\_key]);

// //a[\_key]=0;

// }

// function ChangeMapping(address \_key, uint \_value) public {

// a[\_key] = \_value;

// }

/\*

배열

- length(길이) 있음

- 자료형[] 가시성 지정자 변수명;

uint256[] public b;

uint256[10] public ageFixedSizeArray; 10개 고정정

string[] public nameArray = ["Kal","Jhon","Kerri"]; 초기값 설정

\*/

uint256[] public b;

function addArray(uint \_value) public {

b.push(\_value); // 0 : 5, 1 : 10

}

function getArray(uint \_index) public view returns(uint) {

return b[\_index];

}

function ChangeArray(uint \_index, uint \_value)public{

b[\_index] = \_value;

}

function deleteArray() public {

b.pop();

}

function DeleteArray2(uint \_index)public{

delete b[\_index];

}

function getLength() public view returns(uint) {

return b.length;

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec6 {

/\*

구조체(struct)

자신만의 자료형

struct 구조체명 {

자료형 변수명,

자료형 변수명,

자료형 변수명

}

struct Human {

string name;

uint age;

}

\*/

struct Human{

string name;

uint age;

}

mapping(uint256=>Human) public humanMapping;

Human[] public humanArray;

function createHuman(string memory \_name,uint \_age) public pure {

Human(\_name,\_age);

}

function createHumanInMapping(uint \_key,string memory \_name,uint \_age) public {

humanMapping[\_key] = Human(\_name,\_age);

}

function createHumanInArray(string memory \_name,uint \_age) public{

humanArray.push(Human(\_name,\_age));

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec7 {

/\*

조건문 if, else if, else

- 어떠한 조건을 주어, 조건이 성사되면 조건문 안에 정의된 특정 코드 실행.

- 조건의 결과값은 불리언 형태로 조건에 해당이 되는지(참 인지) 안 되는지(거짓) 판단이 되어야한다.

- 함수 내부에서만 작동

if(조건){

//특정 코드

}else if(조건){

}else{

}

\*/

function fun1(uint \_a) public pure returns(uint) {

if(\_a == 3){

return 555;

}

return 111;

}

function fun2(uint \_a) public pure returns(uint) {

if(\_a == 3){

return 555;

}else{

return 111;

}

}

function fun3(uint \_a) public pure returns(uint) {

if(\_a == 3){

return 555;

}else if(\_a>=5 && \_a<11){

return 999;

}else{

return 0;

}

}

function fun4(uint \_a) public pure returns(uint) {

if(\_a == 3){

return 555;

}else if(\_a>=5 && \_a<11){

return 999;

}else if(\_a == 7){

return 7;

}else{

return 0;

}

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec8 {

/\*

반복문 for, while, do-while

- 특정한 작업을 반복적으로 수행함.

- 함수 내부에서만 작동

for(초기식; 조건식; 증감식;){

특정 작업

}

초기식;

while(조건식){

특정 작업

증감식;

}

초기식;

do{

특정 작업

증감식;

}while(조건식);

\*/

/\*

for(초기식; 조건식; 증감식;){

특정 작업

}

\*/

function forFun() public pure returns(uint) {

uint total = 0;

for(uint a = 0; a<2; ++a ){

total = total + a;

}

return total;

}

/\*

초기식;

while(조건식){

특정 작업

증감식;

}

\*/

function whileFun() public pure returns(uint) {

uint total = 0;

uint a = 0;

while(a<2){

total = total + a;

a++;

}

return total;

}

/\*

초기식;

do{

특정 작업

증감식;

}while(조건식);

\*/

function do\_whilFun() public pure returns(uint) {

uint total = 0;

uint a = 8;

do{

total = total + a;

a++;

}while(a>10);

return total;

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec10 {

/\*

- 0.8~

에러핸들러

- assert의 가스량이 줄음

- 솔리디티 다큐멘테이션에 의하면 assert는 오직 내부적 에러 테스트 용도, 불변성 체크 용도로 사용

- assert가 에러를 발생시키면 Panic(uint256) 이라는 에러타입의 에러를 발생 (try/catch에서 사용)

\*/

function assertNow(uint \_a) public pure {

assert(\_a>=5);

}

function revertNow(uint \_a) public pure { //특정한 조건에 부합하면 에러 발생시키고

if(\_a>=5){

revert("Must be more than 5");

}

}

function requireNow(uint \_a) public pure { //특정한 조건에 부합하지 않으면 에러 발생

require(\_a>=5,"Must be more than 5");

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract lec11 {

/\*

msg.sender

- msg.sender 트랜잭션을 주는 존재

balance

- 현재 이더의 잔액

- 주소.balance

\*/

function getMsgSender() public view returns(address) {

return msg.sender;

}

function balance1(address \_a) public view returns(uint) {

return \_a.balance;

}

function balance2() public view returns(uint) {

return (msg.sender).balance;

}

}

/\*

이더 단위

wei, gwei, ether

1 ether = 10^18 wei => 99 \* 10^18

1 gwei = 10^9 wei

\*/

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.6.0 <0.9.0;

/\*

이더를 보내는 3가지

A 잔액 1 ether

send

A -----------5 ether---------> B : false 리턴

transfer

A -----------5 ether---------> B

call

A -----------5 ether---------> B

1.send : 2300 gas, 성공여부를 true 또는 false로 리턴한다, 주소 타입의 내장 함수

2.transfer : 2300 gas, 실패시 에러를 발생, 주소 타입의 내장 함수

3.call : 가변적인 gas, (gas값 지정 가능), 성공여부를 true 또는 false로 리턴

재진입(reentrancy) 공격 위험성 있음, 2019년 12월 이후 call 사용을 추천

외부스마트 컨트랙트 함수 호출 가능

(준비물 : 호출하려는 스마트 컨트랙트 주소 + 호출하려는 함수명)

payable

Payable은 이더 송수신 할때 필요한 키워드

즉, send, trnafer 이더를 보낼때 주소는 payable이라는 키워드가 필요

이 Payable은 주로 함수,주소,생성자에 붙여서 사용

call은 payable 필요 없음

msg.value

msg.value는 송금보낸 이더의 값

\*/

contract lec13\_1{

function addNumber(uint \_num1, uint \_num2) public pure returns(uint){

return \_num1 + \_num2;

}

function whoIsMsgSender() public view returns(address){

return msg.sender;

}

}

contract lec13 {

event howMuch(uint256 \_value);

//1.송금 기능

// ~ 0.7

//(bool sent, ) = \_to.call.gas(1000).value(msg.value)("");

//require(sent,"Failed");

//0.7 ~

function callNow (address payable \_to) public payable{

(bool sent, ) = \_to.call{value: msg.value}("");

require(sent, "failed");

emit howMuch(msg.value);

}

//2. 외부 스마트 컨트랙 함수 부르기

function callAddNumber(address \_lec13\_1Address, uint \_num1, uint \_num2) public returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_lec13\_1Address.call(

abi.encodeWithSignature("addNumber(uint256,uint256)",\_num1,\_num2)

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

function callWhoIsMsgSender (address \_lec13\_1Address) public returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_lec13\_1Address.call(

abi.encodeWithSignature("whoIsMsgSender()")

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

}

/\*

0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ->callWhoIsMsgSender (lec13) ->

whoIsMsgSender (lec13-1) -? 반환 msg.sender

callWhoIsMsgSender (lec13) 함수의 msg.sender = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

whoIsMsgSender (lec13-1) 함수의 msg.sender = lec13의 주소

\*/

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.6.0;

/\*

fallback

이름 그대로 대비책 함수

왜 쓰는가 ?

- 스마트 컨트랙이 이더를 받을 수 있게 한다.

- 이더 받고 난 후 어떠한 행동을 취하게 할 수 있다.

- call함수로 없는 함수가 불려질때, 어떠한 행동을 취하게 할 수 있다.

0.6 이전

function() external payable {

}

0.6 이후

fallback -> fallback과 recieve

\*/

contract safe{

event received(address \_from,uint \_amount);

function() external payable{

//2300 제한된 가스 -> 로직을 실행.

// lec14 \_lec14 = new lec14();

emit received(msg.sender, msg.value);

}

function checkMybalance() public view returns(uint){

return address(this).balance; //safe 컨트랙트의 잔액

}

}

contract lec14 {

function sendNow(address payable \_to) public payable{

bool sent = \_to.send(msg.value); // return true or false

require(sent,"Failed to send either");

}

function transferNow(address payable \_to) public payable{

\_to.transfer(msg.value);

}

function callNow (address \_to) public payable{

// ~ 0.7

(bool sent, ) = \_to.call.value(msg.value)("");

require(sent,"Failed to send either");

//0.7 ~

// (bool sent, ) = \_to.call{value: msg.value}("");

// require(sent, "Failed to send Ether");

}

function callWrong (address \_safeAddress) public returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_safeAddress.call(

abi.encodeWithSignature("wrong()")

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

function callWrong2 (address \_safeAddress) public payable returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_safeAddress.call.value(msg.value)(

abi.encodeWithSignature("wrong()")

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.6.0 <0.9.0;

/\*

0.6 이후 fallback

fallback은 recevie 와 fallback 으로 두가지 형태로 나뉘게 되었습니다.

receive: 순수하게 이더만 받을때 작동 합니다.

fallback: 없는 함수를 호출할 때 ,없는 함수를 호출하면서 이더를 보낼 때 작동합니다.

0.6 이후

기본형 : 불려진 함수가 특정 스마트 컨트랙이 없을때 fallback 함수가 발동 합니다.

fallback() external {

}

receive() external payable{

}

payable 적용시 : 이더를 받고 나서도 fallaback 함수가 발동합니다.

fallback() external payable {

}

\*/

contract safe{

event received(address \_from,uint \_amount);

event justFallback(string \_str);

fallback() external payable {

emit justFallback("No function");

}

receive() external payable{

// lec15 \_lec15 = new lec15();

emit received(msg.sender, msg.value);

}

function checkMybalance() public view returns(uint){

return address(this).balance;

}

}

contract lec15 {

function sendNow(address payable \_to) public payable{

bool sent = \_to.send(msg.value); // return true or false

require(sent,"Failed to send either");

}

function transferNow(address payable \_to) public payable{

\_to.transfer(msg.value);

}

function callNow (address payable \_to) public payable{

// ~ 0.7

// (bool sent, ) = \_to.call.value(msg.value)("");

// require(sent,"Failed to send either");

// 0.7 ~

(bool sent, ) = \_to.call{value: msg.value}("");

require(sent, "Failed to send Ether");

}

function callWrong (address \_safeAddress) public returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_safeAddress.call(

abi.encodeWithSignature("wrong()")

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

function callWrong2 (address \_safeAddress) public payable returns(bool,bytes memory) {

(bool sent,bytes memory outputFromCalledFunction) = \_safeAddress.call{value :msg.value}(

abi.encodeWithSignature("wrong()")

);

require(sent,"failed");

return(sent,outputFromCalledFunction);

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/\*

생성자

- 스마트 컨트랙트가 배포될 때 제일 먼저 작동하는 함수

- 스마트 컨트랙트를 배포할 때 마다 특정한 값을 세팅해줄

contract lec16 {

constructor(){

}

}

상속

- 특정 스마트 컨트랙트의 모든 기능을 상속 받음

- 상속을 주는쪽 부모 스마트 컨트랙트

- 상속을 받는쪽 자식 스마트 컨트랙트

contract lec16 is 스마트컨트랙트 명 {

}

오버라이딩

- 상속 받은 함수를 변경

- virtual : 부모 스마트 컨트랙트에 있는 오버라이딩할 함수에 지정

- override : 자식 스마트 컨트랙트에 있는 오버라이딩할 함수에 지정

인스턴스화

- 특정 스마트 컨트랙트를 인스턴스화하여 여러개 사용가능

\*/

contract fathersWallet {

uint public money;

constructor (uint \_moeny){

money = \_moeny;

}

function addMoney(uint \_moeny) public{

money = money + \_moeny;

}

function changeMoney(uint \_moeny) public virtual {

money = \_moeny;

}

}

contract son3 {

fathersWallet wallet1 = new fathersWallet(1000);

fathersWallet wallet2 = new fathersWallet(10000);

fathersWallet wallet3 = new fathersWallet(100000);

function addWalletAll(uint \_money1, uint \_money2, uint \_money3) public{

wallet1.addMoney(\_money1);

wallet2.addMoney(\_money2);

wallet3.addMoney(\_money3);

}

function CheckWalletAll() view public returns(uint,uint,uint){

uint money1 = wallet1.money();

uint money2 = wallet2.money();

uint money3 = wallet3.money();

return(money1,money2,money3);

}

}

contract son is fathersWallet{

constructor (uint \_moeny) fathersWallet(\_moeny){

}

}

contract son2 is fathersWallet(91000){

function changeMoney(uint \_moeny) public override{

money = \_moeny \* 2;

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/\* modifier

함수에 적용을 하여, 함수의 행위를 제한

1. 매개 변수가 있는 모디파이어

modifier 모디파이어 이름(자료형 매개변수이름){

//모디파이어 로직

\_;

}

2.매개 변수가 없는 모디 파이어

modifier 모디파이어 이름{

//모디파이어 로직

\_;

}

\*/

contract lec17{

function BuyCigarette1(uint256 \_age) public pure onlyAdults(\_age) returns(string memory){

return "Your payment is succeeded";

}

function BuyCigarette2(uint256 \_age) public pure onlyAdults(\_age) returns(string memory){

return "Your payment is succeeded";

}

function BuyCigarette3(uint256 \_age) public pure onlyAdults(\_age) returns(string memory){

return "Your payment is succeeded";

}

function BuyCigarette4(uint256 \_age) public pure onlyAdults(\_age) returns(string memory){

return "Your payment is succeeded";

}

/\*

1. 매개 변수가 있는 모디파이어

modifier 모디파이어 이름(자료형 매개변수이름){

//모디파이어 로직

\_;

}

\*/

modifier onlyAdults(uint256 \_age){

require(\_age>20,"You are not allowed to pay for the cigarette");

\_; //함수가 언제 시작하는지

}

function BuyCigarette5(uint256 \_age) public pure onlyAdults(\_age) returns(string memory){

return "Your payment is succeeded";

}

/\*

2.매개 변수가 없는 모디 파이어

modifier 모디파이어 이름{

//모디파이어 로직

\_;

}

\*/

uint256 public num = 5;

modifier numChange{

\_;

num = 10;

}

function numChangeFunction() public numChange{

num = 15;

}

}

// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 < 0.9.0;

/\*

Interface

interface : 스마트컨트랙 내에서 정의되어야할 필수 요소를 나타냄 (설명서)

1, 함수는 external로 표시

2, enum, structs 가능

3, 변수, 생성자 불가(constructor X)

\*/

interface ItemInfo {

struct item{

string name;

uint256 price;

}

function addItemInfo(string memory \_name,uint256 \_price) external;

function getItemInfo(uint256 \_index) external view returns(item memory);

}

contract lec20 is ItemInfo{

item[] public itemList; // [itme("name", 100),itme("name2", 10023),itme("name3", 1002) ]

function addItemInfo(string memory \_name,uint256 \_price) public override {

itemList.push(item(\_name,\_price));

}

function getItemInfo(uint256 \_index) public view override returns(item memory) {

return itemList[\_index];

}

}

NFT

소유권 증명

소유권 거래 Royalty

투명성

EIP : Ethereum Improvement Proposal 이더리움 생태계를 발전시키기 위한 제안서

ERC : EIP의 종류 중 하나로, 어플리케이션 레벨의 표준 및 컨벤션 정의

메타데이터 : 그림에 대한 데이터

중앙화된 저장소 : amazon s3, google cloud

탈중앙화된 저장소 : IPFS -> 메타데이터를 중앙화된 저장소가 아닌 IPFS로 저장하면 NFT의 메타데이터가 변경되지 않을 것임을 강제할 수 있다.

터미널에서

Ipfs init

ipfs daemon

<http://localhost:5001/ipfs/bafybeifeqt7mvxaniphyu2i3qhovjaf3sayooxbh5enfdqtiehxjv2ldte/#/>

ipfs.exe 경로에서 이미지 파일 ipfs 올릴 수 있음 -> daemon 실행한채로

그 경로에 이미지 넣고 ipfs add brozonew.png

Ipfs get 키 값 누르면 다운도 가능

ipfs pin add QmQz53rCofUVFEQSd8YeHETy4Ym4To7R6KNBg6pt8zyHWf -> 고정 영구 저장

ipfs pin ls

opensea 메타 데이터 형식

<https://ipfs.io/ipfs/QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/1>

ipfs.io/ipfs/

opensea contract -> 소스코드 다 있음

Hardhat

npx hardhat init

npm i --save @openzeppelin/contracts

ipfs add -r metadata

Pinata

Managed ipfs 서비스