솔리디티 프로그래밍

완벽가이드

Solidity 기초 문법부터 디앱 개발과 배포까지

<조대환>

11/26 화 ~ 12/11수 (1회독 100% 완료)

㈜주 가 드

김재원

# **목차**

[**1. 블록체인, 스마트 컨트랙트와 솔리디티 3**](#_Toc184718272)

[**2. 리믹스로 솔리디티 시작하기 6**](#_Toc184718273)

[**3. 가시성 지정자와 함수 11**](#_Toc184718274)

[**4. 조건문과 반복문 21**](#_Toc184718275)

[**5. 매핑, 배열, 구조체 22**](#_Toc184718276)

[**6. 객체 지향 프로그래밍 24**](#_Toc184718277)

[**7. 오류 및 예외 처리 27**](#_Toc184718278)

[**8. 모디파이어/ 이넘/ 임포트/ 라이브러리 30**](#_Toc184718279)

[**9. 이더 송수신 32**](#_Toc184718280)

[**10. 메타마스크를 활용한 배포 33**](#_Toc184718281)

**+부록. Dgit로 깃허브에 스마트 컨트랙트 파일 저장**

**진법 변환**

**전역 변수**

**함수 스타일**

**주소의 내장 변수 및 함수**

## 1. 블록체인, 스마트 컨트랙트와 솔리디티

* 1. 블록체인이란

분산형 네트워크 기반의 데이터 저장 기술

네트워크에 참여한 모든 노드는 동일한 DB가진다.

* + 1. 블록 체인 구조

블록에는 정보를 저장한다. Ex)암호화폐 거래 내역

* + 1. 블록 체인 특징

불변성, 탈중앙화, 합의 알고리즘의 3가지 특징이 있다.

* 1. 이더리움과 스마트 컨트랙트
     1. 이더리움

창시자 비탈릭 부테린이 이더리움 프로토콜과 스마트 컨트랙트 개념을 제안해 블록체인을 확장시켰다.

* + 1. 스마트 컨트랙트

이더리움 핵심 기능이며, 스마트 컨트랙트 전을 블록체인 1.0, 후를 블록체인 2.0이라 한다.

블록체인 1.0은 주로 암호화폐의 거래 내역을 저장한다.

블록체인 2.0은 암호화폐의 거래 내역 저장과 자동화 계약 시스템(신뢰)를 강조한다.

<스마트 컨트랙트 특징>

중개인의 필요성을 해소

코드가 계약을 이행하므로 중개인에게 수수료 지불할 필요가 없다.

<스마트 컨트랙트 작동 방식>

동전 교환기와 닮은 점이 많다.

1. 프로그래밍 언어로 올바르게 구현돼야 함.

2. 둘 이상의 계약 당사자 필요

3. 계약을 증명하는 대상(즉 노드) 필요

<스마트 컨트랙트 장단점>

장점- 보안성, 신뢰성, 비용 절감

단점-최신 기술이라 법적, 제도적 장치가 완벽하지 않다.

* + 1. 이더와 가스

가스

블록체인에 정보를 저장하기 위해서는 가스 비용이 항상 발생한다.

이더

이더리움 네트워크의 암호화폐 단위

* + 1. 이더리움 네트워크

이더리움은 메인넷과 테스트넷을 지원한다.

메인넷의 이더 – 금전적 가치 존재

테스트넷 – 실험 목적의 스마트 컨트랙트 배포

하드포크와 소프트포크

포크 : 블록체인 네트워크의 버전 업데이트

하드포크 : 블록체인 네트워크의 근본적인 틀을 변경

소프트포크 : 기존 네트워크를 기반으로 부분적으로 이뤄지는 업데이트

* + 1. NFT

블록체인에 저장된 대체 불가능 토큰이며 스마트 컨트랙트로 작성돼 있다.

주로 사진, 비디오, 오디오, 문서 및 디지털 파일을 나타내는 데 활용한다.

* + 1. 솔리디티

이더리움에서 제공하는 스마트 컨트랙트 개발 프로그래밍 언어.

C++와 자바스크립트, 파이썬 영향을 받았다.

## 2. 리믹스로 솔리디티 시작하기

2.1. 리믹스 둘러보기

* 1. 리믹스로 스마트 컨트랙트 작성, 컴파일, 배포
     1. 워크 스페이스 만들기
     2. 불필요한 폴더와 파일 삭제
     3. 실습 폴더와 파일 만들기
     4. 코드 작성

2.2.5. 컴파일과 배포

2.3. Remixd로 스마트 컨트랙트 저장

2.3.1 Node.js 설치

2.3.2 Remixd 설치

2.3.3 연동할 폴더를 로컬 서버에 호스팅

2.3.4 스마트 컨트랙트를 파일로 저장

2.4. 스마트 컨트랙트의 기본 구조, 자료형, 연산

2.4.1 스마트 컨트랙트의 기본 구조

4가지를 눈여겨 봐야 한다.

1. SPDX-License-Identifier: GPL-3.0

솔리디티 컴파일러는 라이선스를 명시하라고 권장한다.

2. pragma solidity 0.8.7;

스마트 컨트랙트의 솔리디티 컴파일러 버전을 나타낸다.

3. contract

스마트 컨트랙트 작성 시작을 알리는 키워드다.

4. 주석

코드의 부연 설명을 나타낸다.

2.4.2. 변수와 자료형

솔리디티 자료형은 값 타입과 참조 타입으로 나눌 수 있다.

값 타입은 값이 할당되거나 함수 인자로 활용되면 해당 값 자체가 복사된다.

참조 타입은 현재 해당하는 값의 주소만 복사한다.

정수 타입 : int와 uint

**기존 프로그래밍 언어와 다르게 솔리디티는 소수점이 있는 숫자를 표현하는 float와 double을 지원하지 않는다.**

**정수형 타입 int는 음수를 포함한 256가지 수(-128~127)로 구성되어 있는 반면**

**Uint는 0~255의 양수로 구성되어 있다.**

2.4.3 연산자

예제 2.2대입 연산자)

연산자 =

예제 2.3 자료형에 따른 각 변수의 기본값

예제 2.4 산술 연산자

연산자 +, -, \*, /, %, \*\*

소수점을 지원하지 않는다는 사실 확인

예제 2.5) 복합 할당 연산자(확인 후 스킵)

+=, -=, \*=, /=, %=

예제 2.6) 전위 증가와 후위 증가(아는 내용이므로 스킵)

++, --

예제 2.7) 전위 감소와 후위 감소(아는 내용이므로 스킵)

예제 2.8) 비교 연산자 (아는 내용이므로 스킵)

>, <, >=, <=, ==, !=

예제 2.9) 비교 연산자 (아는 내용이므로 스킵)

&&, ||, !

예제 2.10) 시프트 연산자 (아는 내용이므로 스킵)

<<, >>

예제 2.11) 비트 연산자 (아는 내용이므로 스킵)

&, |, ^, ~

예제 2.12) 우선 순위 (아는 내용이므로 스킵)

예제 2.13) 상수 예제 (아는 내용이므로 스킵)

2.5 개념 체크

1. 변수는 선언한 후 값을 변경할 수 있다.

2. !height라는 변수명은 사용할 수 없다.

3. 솔리디티는 소수점을 지원하지 않는다.

4. true와 false를 갖는 자료형은 bool이다.

5. -1을 포함하는 자료형은 int다.

6. 자료형 uint는 uint256을 나타낸다.

7. 자료형 address는 bytes20이다.

8. 자료형 bytes1의 값은 16진수 형태로 입력해야 한다.

9. 자료형 byte는 bytes1을 나타낸다.

10. false || true는 true를 반환한다.

11. true && false는 false를 반환한다.

12. !true == true는 false를 반환한다.

13. 증감 연산자는 피연산자의 값을 1만큼 증가시키거나 감소시킨다.

14. 괄호 연산자의 우선순위는 1이다.

15. 상수는 선언 후 값을 변경할 수 없다.

16. 상수의 키워드는 constant이다.

17.age\_30은 상수명으로 사용할 수 있다.

## 3장 가시성 지정자와 함수

3.1. 가시성 지정자

가시성 지정자는 자바와 C언어 등에서 쓰는 접근 제한자와 같은 개념이다.

가시성 지정자에는 다음 4가지가 있다.

Public, external, private, internal

Public문은 누구에게나 쉽게 열리고 닫힌다.

External문은 외부에서만 문을 열 수 있다.

private문은 오직 집주인만 문을 열고 닫을 수 있다.

internal문은 집 내부에서 친족만 문을 열고 접근 가능하다.

3.2. 함수의 개념과 정의 방법

3.2.1 함수의 개념

함수를 사용함으로써 유지보수, 재활용, 가독성에서 이점을 얻을 수 있다.

3.2.2 함수 정의

예제 3.2) 기본 함수 정의

function이라는 키워드를 통해 선언을 시작한다.

예제 3.3) 매개변수를 가진 함수

매개변수 10을 넣고 같은 값이 대입되는 지 확인

예제 3.4) 두 개 이상의 매개변수를 가진 함수

A에 d가 입력되는지 확인

예제3.5) 반환값을 가진 함수

초기값 3을 가지고 있던 a가 함수에 의해서 100이 반환되는 지 확인

예제 3.6) 두 개 이상 반환값을 가진 함수

100과 0이 반환되는지 확인

예제 3.7) 함수 반환값에 이름 명시

예제 3.8) 함수의 반환값을 변수에 대입

예제 3.9) 매개변수와 반환값이 있는 함수

3.3. 기본 모디파이어

모디파이어는 특정한 기능을 미리 정의해 여러 함수에 적용할 수 있다.

솔리디티에서 제공하는 모디파이어에는 pure, view, payable이 있다.

3.3.1. pure와 view의 비교

Pure

Pure를 함수에 적용하면 함수 밖에 선언된 변수를 pure가 적용된 함수 내부로 갖고 올 수 없다.

View

View 모디파이어는 pure보다 조금 더 느슨하다.

3.3.2. 함수에 모디파이어를 적용하는 방법

Function myFun() public pure{//함수 로직}

예제 3.10) pure가 적용된 함수

예제 3.11) pure가 잘못 적용된 함수

예제 3.12) view가 적용된 함수

3.4. 함수와 참조 타입

참조 타입은 배열, 매핑, 구조체로 구성되어 있다.

솔리디티는 스토리지, 메모리, 콜데이터, 스택이라는 4개의 영역이 있다.

스토리지

영속적인 읽기 쓰기가 가능한 저장공간이다. 키와 값 쌍으로 이루어진 매핑 구조이다.

메모리

휘발적으로 읽고 쓰기가 가능한 저장공간이다.

콜데이터

메모리와 비슷하게 휘발성이 있지만, 데이터 저장이 불가능하다.

스택

가상 머신에서 휘발성을 가진 데이터를 유지 관리하는 공간이다.

예제 3.15) 함수 참조 타입 사용 시 매개변수, 반환값, 변수 메모리 저장

예제 3.16) external 함수 참조 타입 사용 시 매개변수 calldata 지정

예제 3.17) 상태 변수와 지역 변수

3.6. 함수와 가시성 지정자

4개의 가시성 지정자 비교

예제 3.18) 4개의 가시성 지정자

3.7. 개념 체크

1. 가시성 지정자는 총 4개로 구성되어 있다.

2. view는 가시성 지정자에 포함되지 않는다.

3. external은 내부 접근이 불가능하다.

4. internal은 외부 접근이 불가능하다.

5. private은 내부 접근이 가능하다.

6. public은 외부와 내부 접근이 가능하다.

7. external은 변수에 적용이 불가능하다.

8. 함수를 선언하는 키워드는 function이다.

9. 함수의 모디파이어는 pure, view, payable이다.

10. pure가 적용된 함수는 함수 외부인 스토리지 공간에 저장된 변수에 접근할 수 없다.

11. view가 적용된 함수는 함수 외부인 스토리지 공간에 저장된 변수에 접근할 수 있다.

12. 솔리디티는 4개의 저장공간을 갖고있다.

13. 스토리지 공간에 저장된 변수는 단기적이 아니다.

14. 메모리 공간에 저장된 변수는 단기적이다.

15. 함수의 매개변수는 메모리에 저장된다.

16. 지역 변수는 그 지역 변수가 정의된 함수 밖에서 사용이 불가능하다.

17. 참조 타입 변수를 함수의 매개변수에 사용 시 memory 키워드가 필요하다.

18. 참조 타입 변수를 external 함수의 매개변수에 사용 시 calldata 키워드가 필요하다.

19. private가 명시된 변수는 public으로 명시된 함수를 통해 외부로 출력할 수 있다.

20. internal이 명시된 변수는 public으로 명시된 함수를 통해 외부로 출력할 수 있다.

## 4. 조건문과 반복문

4.1. 조건문

4.1.1. 조건문의 구조

4.1.2. if 문의 예

4.1.3. if와 else를 사용하는 조건문의 예

4.1.4. if, else if, else를 사용하는 조건문의 예

4.1.5. 조건문 구조에 따른 차이

4.2. 반복문

4.2.1. 3가지 반복문 구조

4.2.2. for문의 예

4.2.3. while문의 예

4.2.4. do-while 문의 예

4.3. 반복문 응용

4.3.1. 반복문과 조건문의 조합

4.3.2. 중첩 반복문

4.3.3. 반복문의 continue와 break

4.4. 개념 체크

1. 조건문은 if로 시작한다.

2. 조건문의 조건을 정의할 때 조건에 대한 결괏값이 불리언형으로 판단돼야 한다.

3. 조건문 else는 앞의 조건문이 거짓일 때 실행된다.

4. 조건문 else에 따로 조건을 정의할 수 없다.

5. 조건문 if 이후 따로 조건을 정의하려면 else if를 써야 한다.

6. 조건문 if와 else if의 조건이 같다면 if안에 있는 조건문 로직이 실행된다.

7. 조건문 if(true)는 조건이 항상 참이므로 어떠한 값이 와도 조건문을 실행한다.

8. 반복문에는 for, while, do-while이 있다.

9. 반복문은 초기식, 조건식, 증감식에 따라 반복문 횟수가 결정된다.

10. 반복문 for는 조건식에 부합하지 않아도 한 번은 무조건 실행되지 않는다.

11. 반복문 do-while은 조건식에 부합하지 않아도 한 번은 무조건 실행된다.

12. 반복문 내부에 조건문을 정의할 수 있다.

13. 한 개 이상의 반복문을 중첩하여 쓸 수 있다.

14. 반복문의 키워드는 continue와 break이다.

15. 반복문 키워드 continue는 현재 실행하고 있는 반복문을 종료시키지 않는다.

## 5. 매핑, 배열, 구조체

5.1. 매핑

매핑은 자바스크립트의 오브젝트와 같이 카와 값의 형태를 저장한다.

5.1.1. 매핑 구문

매핑을 명시하는 방법은 mapping 키워드로 선언한다.

Mapping(address=>uint)public balances

5.1.2. 매핑에 키와 값을 추가

5.1.3. 매핑의 키와 값을 삭제

5.2. 배열

5.2.1. 배열의 구조

배열에는 자료형이 같은 한 개 이상의 값을 순차적으로 저장한다.

배열에서 자리의 순서를 인덱스라고 한다.

5.2.2. 배열의 인덱스에 대응하는 값을 구하기

5.2.3. 배열에 값을 추가하고 변경

5.2.4. 배열의 값을 삭제

5.2.5. 순차 검색 알고리즘

배열, 반복문, 조건문을 통해 간단한 순차 검색 알고리즘 만들기

솔리디티는 문자열과 문자열을 비교하는 기능이 없으므로 내장 함수 keccak256을 통해 문자열을 해시로 만든다.

5.3. 구조체

사용자 정의 자료형이라 생각할 수 있다. 변수, 매핑, 배열 등과 깉이 자료형을 명시해야 하는 곳에 적용할 수 있다.

5.3.1. 구조체 정의 및 반환의 예

5.3.2. 구조체를 적용한 배열과 매핑의 예

5.4. 참조 타입의 데이터 저장 영역

참조 타입의 큰 특징은 함수 내부에 매개변수와 같은 지역변수를 정의할 때 memory 또는 calldata라는 데이터 저장공간을 필수적으로 명시한다는 것이다.

5.4.1. 값 타입과 참조 타입의 데이터 저장 방법

5.4.2. memory에 저장된 변수를 다른 memory 변수에 대입

5.4.3. storage에 저장된 변수를 local storage 변수에 넣기

5.4.4. storage에 저장된 변수를 memory 변수에 넣기

5.4.5. 그 외의 참조 타입 storage가 포함된 관계

5.5 개념체크

1. 매핑, 불리언, 배열, 구조체는 참조 타입이 아니다.

2. 배열은 키와 값으로 구성되어 있지 않다.

3. 배열은 push라는 내장함수를 통해 값을 추가한다.

4. 배열은 pop이라는 내장함수를 통해 값을 삭제한다.

5. 매핑은 pop이라는 내장함수가 없다.

6. 매핑은 인덱스가 없다.

7. 배열의 값은 키워드 delete를 통해 지울 수 있다.

8. 배열의 길이는 length를 통해서 알 수 있다.

9. 매핑 값은 키워드 delete를 통해 지울 수 있다.

10. 구조체는 자신만의 자료형을 작성하는 것이다.

11. 구조체 내부에 string 변수를 저의할 때 memory를 명시안 한다.

12. 구조체는 점 연산자를 통해 구조체 내부에 정의된 변수에 접근할 수 있다.

13. 참조 타입 storage와 memory의 관계는 값 자체를 복사한다.

14. 매개변수에 storage를 명시하려면 해당 함수의 가시성 지정자가 internal 이어야 한다.

15. 참조 타빙 memory와 memory의 관계는 값을 가리키는 주소를 복사한다.

## 6.객체 지향 프로그래밍

6.1. 객체 지향 기본

이벤트와 생성자를 알아본 다음, 상속, 캡슐화를 알아본다.

6.1.1. 이벤트

솔리디티의 이벤트는 스마트 컨트랙트 또는 유저의 특정한 상태를 출력하고 그 상태를 블록체인에 저장한다.

From, to, amount가 있다.

이벤트의 장점은 두 가지이다.

1. 작은 가스 비용으로 데이터를 저장할 수 있다.

2. 프런트엔드와 쉽게 소통할 수 있다.

6.1.2. 생성자

생성자는 스마트 컨트랙트가 배포될 때 제일 먼저 실행되는 함수이며 모든 스마트 컨트랙트에 생성자를 필수적으로 선언하지 않아도 된다.

생성자와 같이 활용되는 immutable은 상수 키워드인 constant와 비슷한 기능을 한다. 상수와 같이 값이 한번 입력되면 변경이 불가능하다.

Constant 와 immutable의 차이점은 초깃값의 선언 여부다. 상수는 초깃값을 필수적으로 입력해야하지만, immutable은 초깃값을 선언하지 않아도 된다.

배열 같은 참조 타입은 immutable을 적용할 수 없고 오직 uint같은 타입만 적용할 수 있다.

생성자 정의

예제6.2.

Immutable

예제 6.3.

6.1.3. 객체 지향

객체 지향 언어는 여러 객체가 상호작용하는 언어이다.

6.1.4. 상속

상속은 한 스마트 컨트랙트가 자신의 변수나 함수를 다른 스마트 컨트랙트에 주는 것이다.

Private와 internal의 접근 범위

6.1.5. 캡슐화

변수와 변수를 처리하는 함수의 로직을 캡슐로 감싸는 것이다.

6.2. 객체 지향 고급

다형성과 추상화에 대해서 알아본다.

6.2.1. 다형성

같은 프로그래밍 인자가 서로 다른 자료형을 갖는 것이다.

6.2.2. 다중 상속

한 개 이상의 부모 스마트 컨트랙트를 상속할 때 다중 상속이라고 한다.

6.2.3. 추상화

추상화는 만들고자 하는 여러 개의 스마트 컨트랙트의 공통 함수를 묶어 명시하는 행위다. 스마트 컨트랙트는 객체의 설계도다.

추상 스마트 컨트랙트

6.3. 개념체크

1. 이벤트는 스마트 컨트랙트를 배포하면 제일 먼저 실행되는 함수가 아니다.

2. 생성자는 스마트 컨트랙트를 배포하면 제일 먼저 실행되는 함수이다.

3. 이벤트를 정의할 때 string에 memory를 붙이지 않는다.

4. 이벤트는 변수를 저장할 때보다 비교적 적은 가스 비용으로 데이터를 저장할 수 있다.

5. 이벤트를 정의할 때 키워드 event를 사용한다.

6. 이벤트를 출력할 때 키워드 emit을 사용한다.

7. 생성자를 정의할 때 생성자 명을 정의하지 않는다.

8. 생성자의 키워드는 constructor이다.

9. 솔리디티는 객체 지향 언어이다.

10. 객체 지향 언어의 대표적인 특징 중 하나는 상속이다.

11. 인스턴스화할 때 키워드 new가 필요하다.

12. 상속 시 변수가 internal로 명시됐다면 자식 스마트 컨트랙트가 상속받을 수 있다.

13. 추상 스마트 컨트랙트와 인터페이스는 스마트 컨트랙트의 설계도이다.

14. 인터페이스와 ABI는 아무런 정보 손실 없이 변환이 가능하다.

15. 캡슐화와 정보 은닉화를 통해 프로그램의 보안성을 높일 수 있다.

## 7. 오류 및 예외 처리

7.1. assert / revert / require

이 세 키워드의 공통점은 오류를 발생시켜 트랜잭션을 실패로 만든다는 것이다.

NFT는 대체 불가능한 토큰이다.

Assert

Assert는 내부적으로 문제가 있을 때 오류를 발생시키도록 설계됬다.

Revert

Revert는 오류를 발생시키지만 assert와 다르게 오류가 나는 이유를 메시지로 출력할 수 있다.

Require

Require는 쉽게 if의 조건문과 revert가 하나로 합쳐진 것이라고 생각할 수 있다.

7.1.1 컴파일러 버전에 따른 assert 가스비 비교

Assert 에서 9/0이 되면 assert에서 오류를 발생시킨다. 최신 컴파일러 버전에서는 트랜잭션 시 예상 가스 비용을 지불하지만 오류로 인해 소모되지 않은 가스비를 환불받는다.

7.1.2. revert와 require 정의 방법

7.2. try/ catch

7.2.1. try/catch 구문

7.2.2. try/catch 정의 방법

예제 7.3에는 3개의 이벤트가 있다. 오류를 조치할 때 출력된다.

Output5함수

Try/catch에 적용되며 uint 자료형을 가진 매개변수 \_num을 받는다.

Output5WithTryCatch 함수

Try에 적용할 수 있는 경우는 외부 스마트 컨트랙트의 함수를 호출할 때와 외부 스마트 컨트랙트를 인스턴스화 할 때다.

7.2.3. 인스턴스화에 try/catch 적용

7.2.4. 외부 함수에 try/catch 적용

7.2.5. try/catch 블록의 try 블록에서 오류가 발생할 경우

7.3. 개념 체크

1. assert, revert, require를 통해 오류를 발생시킬 수 있다.

2. 내부적인 문제가 발생하면 assert를 통해 오류를 발생시킨다.

3. 0으로 5를 나눌 시 오륲가 발생하는 데 이 오류는 asset를 통해 발생한다.

4. assert는 오류 메시지를 출력할 수 없다.

5. revert는 오류가 나는 조건을 매개변수로 입력받을 수 없다.

6. require는 오류가 나는 조건을 매개변수로 입력받을 수 있다.

7. require는 오류 메시지를 매개변수로 입력받을 수 있다.

8. 솔리디티 컴파일러 버전 0.8 이전 assert는 가스비를 모두 소비한다.

9. require와 revert는 가스비를 환불받아 비교적 낮은 가스를 소모한다.

10. try/catch는 외부에 있는 함수를 호출할 떄와 인스턴스화할 떄만 적용가능하다.

11. try/catch가 적용된 함수에서 오류가 발생하면 트랜잭션 실패가 나지 않는다.

12. catch error는 revert 오류를 처리한다.

13. catch panic은 assert 오류를 처리한다.

14. catch만 선언하면 assert, revert, require 오류 상관없이 모든 오류를 처리한다.

15. try/catch 블록에서 오류가 발생하면 catch는 오류를 처리할 수 없다.

## 8. 모디파이어/ 이넘/ 임포트/ 라이브러리

모디파이어는 다수의 함수에 특정 로직을 부여한다.

8.1. 모디파이어의 개념과 정의 방법

모디파이어는 스프링 프레임워크의 AOP와 유사하다. AOP는 관점 지향 프로그래밍이라하며 공통된 기능을 분리해 다른 로직에 적용한다.

모디 파이어와 require를 같이 활용하는 이유

모디파이어를 활용하면 100개의 함수에 대해서 일일이 변경하지 않아도 된다.

Require를 적용해 모디파이어를 생성

모디파이어를 정의하려면 modifier라는 키워드로 시작해 모디파이어명과 매개변수를 선언한다. 로직에는 require와 \_;가 있다.

\_;는 모디파이어가 적용될 함수의 로직이 시작되는 지점을 나타낸다.

8.1.2. 모디파이어 정의 예시

8.1.3. 모디파이어의 함수 실행 순서

\_;는 함수의 로직이 실행되는 시점을 나타낸다.

8.2. 이넘

이넘은 uint8범위를 가진 상수 집합이다. 0~255 값에 이름을 지정할 수 있다.

이넘은 상수의 집합이므로 한 번 정해진 값은 변하지 않는다.

enum이라는 키워드로 이넘의 이름을 지정한다.

이넘과 require를 활용해 개발자가 의도한 순서대로 통제할 수 있다.

8.2.1. 이넘의 정의 예시

8.3. 임포트

외부 파일을 불러올 수 있다.

8.3.1. 임포트 예제

8.4. 라이브러리

스마트 컨트랙트의 경우 스토리지에 있는 상태 변수를 저장하거나 변경하지만, 라이브러리는 상태 변수를 변경하는 용도가 아닌 특정한 값을 입력받아 정의된 함수 로직에 따라 연산 후 값을 반환한다.

8.4.1. 라이브러리 특징

1. 코드 재사용과 가스 절약에 이점

2. 자료형에 적용 가능

3. 상태 변수와 상속의 정의가 불가능

4. 정의된 함수가 internal과 external일 경우 내부적으로 다르게 동작

5. 이더를 받을 수 있다.

8.4.2. 라이브러리 정의

다음과 같은 방식으로 라이브러리를 정의한다.

Library Math{

Function add(uint a, uint b) internal pure returns(uint){}}

8.4.3. 라이브러리 사용

Using 구문을 사용해 라이브러리를 스마트 컨트랙트에 적용한다.

8.4.4. 실습: 오버플로를 막는 라이브러리

오버플로는 특정 값이 최대 허용 범위를 넘어간 것이고, 언더플로는 특정 값이 최소 허용 범위를 넘어간 것이다.

8.5 개념 체크

1. 외부 파일에 있는 스마트 컨트랙트는 임포트를 통해 불러올 수 있다.

2. 이넘은 상수의 집합이다.

3. 이넘은 uint8의 범위를 갖고 있다.

4. 모디파이어는 다수의 함수에 특정한 로직을 부여할 수 있다.

5. 모디파이어의 \_;는 모디파이어가 적요된 함수의 로직을 나타낸다.

6. 상대 경로의 ../는 상위 폴더 위치를 나타낸다.

7. 라이브러리는 재사용이 가능하다.

8. 라이브러리는 상태 변수를 저장할 수 없다.

9. 라이브러리는 자료형에 적용할 수 있다.

10. 라이브러리를 활용함으로써 가스 소비를 절약할 수 있다.

## 9. 이더 송수신

9.1. 이더 송수신 기본 개녕: address/payable/msg.sender/msg.value

주소형 타입 address와 키워드 payable을 알아보고 전역변수에 대해서 살펴 본다.

9.1.1. address 자료형

블록체인 네트워크에 존재하는 각 주소는 중복되지 않고 익명의 유저를 대변한다.

사람의 주소는 EOA(eternally owned account), 즉 외부 소유 계정이라 한다.

스마트 컨트랙트의 주소는 CA(contract Account)라고 한다.

9.1.2. payable 키워드

call함수를 통해 payable이 적용되지 않는 주소는 이더를 받을 수 없다.

스마트 컨트랙트의 주소로 이더를 받으려면 fallback 또는 receive 함수가 필요하다.

9.1.3. 전역 변수 msg.sender와 msg.value

전역 변수는 블록의 현재 번호, 블록의 가스 제한량과 같은 블록체인의 전반적인 정보를 나타낸다.

전역 변수 자료형 반환값

Msg.sender address 현재 호출한 메시지 발신자

Msg.value uint 메시지와 함께 전송된 wei의 개수

전역 변수 msg.sender의 응용

9.2. 이더를 보내는 3가지 함수: send/call/transfer

이더를 스마트 컨트랙트의 주소인 CA로 보내려면 스마트 컨트랙트에 함수 fallback 또는 receive를 명시해야 한다.

9.3. 함수 fallback과 receive

9.3.1. 함수 fallback의 두 가지 기능

1. CA로 전송된 이더를 받는 기능

2. 스마트 컨트랙트에 존재하지 않는 함수가 불렀을 때 대신 호출받음

9.3.2. 솔리디티 버전 0.6미만 fallback

버전 0.6미만 fallback 특징은 다음과 같다.

1. 이름이 존재하지 않는 무기명 함수다.

2. fallback 함수에 가시성 지정자 external을 필수로 붙여야 한다.

3. fallback 함수는 매개변수가 존재하지 않는다.

4. fallback 함수는 반환값이 존재하지 않는다.

5. fallback 함수에 payable이 적용돼야 스마트 컨트랙트가 이더를 받을 수 있다.

6. 스마트 컨트랙트가 이더를 받으면 payable이 적용된 함수 fallback이 실행됨

7. 명령 코드의 가스 비용이 증가할 수 있어 fallback 함수를 실행하기 충분하지 않을 수 있다.

8. 이러한 문제를 해결하려면 함수 call을 이용해 이더를 보내야 한다.

9. fallback에 payable 적용하지 않을 시 스마트 컨트랙트는 fallback을 통해 이더를 받을 수 있다.

10. 하나의 스마트 컨트랙트에는 하나의 fallback 함수를 수용할 수 있다.

9.3.3. 솔리디티 버전 0.6이상 fallback과 receive

9.4. 함수 call과 delegatecall

주소형 타입의 내장 함수 call과 delegatecall에 대해서 알아본다. 문법은 같으나 작동하는 방식이 다르다.

9.4.1. 함수 call

call함수는 이더를 보내는 기능만 하는 것이 아니라 외부 스마트 컨트랙트의 함수를 호출할 수 있다.

이더를 보내는 call은 다음과 같다.

(0x12..4D3).call{value:1000}(“”)

함수를 호출하는 call은 다음과 같다.

(0x3F..4D3).call(abi.encodeWithSignature(“add(uint256,uint256)”,10,15))

Fallback과 call 함수

9.4.2. 함수 delegatecall

주소형 타입의 내장 함수 delegatecall은 call과 다르게 이더를 보낼 수 없지만 call과 같이 외부 스마트 컨트랙트에 존재하는 함수를 호출할 수 있다. Call과 다른 점은 함수를 호출하는 방식이 다르다는 것이다.

기본적으로 블록체인에 한 번 배포된 스마트 컨트랙트는 변경이 불가능 하므로, 로직을 변경하려면 기존 스마트 컨트랙트를 버려야 한다. 새롭게 배포하게 되면 기존 스마트 컨트랙트에 저장된 정보가 없어진다. 입력할 정보가 많다면 시간이 많이 소요되며 정보가 누락될 가능성이 있다.

Delegatecall로 이러한 불편함을 해소할 수 있다.

9.5 개념 체크

1. 주소형 타입은 20바이트다.

2. 주소형 타입은 EOA와 CA로 나뉜다.

3. EOA는 외부 소유 계정이다.

4. CA는 컨트랙트 주소이다.

5. 현재 전송된 잔액은 msg.value를 통해 알 수 있다.

6. 현재 트랜잭션을 준 주소는 msg.sender를 통해 알 수 있다.

7. 이더를 보내는 함수는 transfer이다.

8. 함수 call을 통해 이더를 보낼 때 보내는 주소는 payable을 적용하지 않는다.

9. 함수 send를 통해 이더를 보낼 때 보내는 주소는 payable을 적용해야 한다.

10. transfer 함수를 통해 이더를 보낼 때 보내는 주소는 payable을 적용해야 한다.

11. 함수 sendsms 2300gwei를 receive함수에 제공한다.

12. 존재하지 않는 함수가 호출될 때 fallback함수가 대신 호출된다.

13. 함수 call은 외부 스마트 컨트랙트에 있는 함수를 호출할 수 있다.

14. 함수 call은 이더를 보내면서 외부 스마트 컨트랙트에 있는 함수를 호출할 수 있다.

15. 함수 delegate는 이더를 보낼 수 없다.

## 10.메타마스크를 활용한 배포

10.1. 메타마스크

10.1.1. 메타마스크 설치

메타마스크는 대표적인 암호화폐 지갑이며 이더리움 네트워크와 상호 작용할 수 있다.

참고 자료

솔리디티 프로그래밍 완벽 가이드 . 조대환