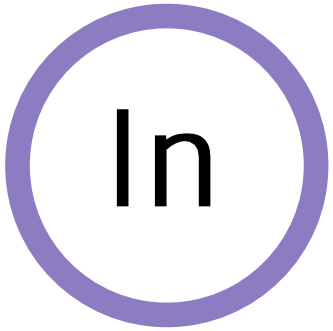
이더리움 DApp

Solidity 언어 실습



Solidity

04 Crypto Zombie(2)

Crypto Zombie Lesson2

Lesson1에선..

ZombieFactory 생성
- 모든 좀비를 기록하도록 state설정
- 좀비 생성 함수
- 각 좀비는 DNA를 통해 랜덤한 외모 설정

Lesson2에선..

- **게임기능**
  - 새로운 좀비 생성 기능
  - 좀비가 다른 좀비를 공격하고 먹을 수 있도록

자세한 사항은 링크 참조
https://cryptozombies.io/ko/lesson/2/chapter/1

04 Crypto Zombie(2)

Chapter2

목표: 특별한 자료형(mapping과 address 이해하기)

```
mapping (address => uint) public accountBalance;
// 금융 앱용으로, 유저의 계좌 잔액을 보유하는 uint를 저장

mapping (uint => string) userIdToName;
// 혹은 userID로 유저 이름을 저장/검색하는 데 매핑을 쓸 수도 있음
```

- **Mapping은 기본적으로 Key-Value 저장소**로 데이터를 저장하거나 검색하는데 이용됨.
- 첫 번째 예시에서 키는 address 이고 값은 uint
- 두 번째 예시에서 키는 uint 이고 값은 string

실습

좀비 소유권 저장하기.
- N번 좀비의 소유권은 address에 있다.
- address가 소유한 좀비는 N개이다.

1. zombieToOwner라는 public mapping생성
   (key: uint, value: address) // *uint가 좀비의 id가 될 예정*
2. ownerZombieCount라는 mapping생성
   (key: address, value: uint) // *address가 소유한 좀비의 개수는 uint이다.*

Chapter3

목표: msg.sender(특별한 전역 변수)

```
mapping (address => uint) favoriteNumber;

function setMyNumber(uint _myNumber) public {
// `msg.sender`에 대해 `_myNumber`가 저장되도록  `favoriteNumber` 매핑을 업데이트

 favoriteNumber[msg.sender] = _myNumber;
// ^ 데이터를 저장하는 구문은 배열로 데이터를 저장할 때와 동일
 }
```

- msg.sender는 해당 Tx를 호출한 계좌(tx 메시지의 from)가 저장
- 위 예시는 mapping과 msg.sender를 함께 사용한 예시임.
- 위 예시에서 만약 다른 사람의 계정의 데이터를 수정하려면,,
  msg.sender에 접근을 위하여 해당 account의 private key를 탈취해야만 함 (보안성 높음)

실습

_createZombie 수정하기

1. 새로운 좀비의 id(배열의 idx)를 얻은 후 zombieToOwner 매핑을 업데이트 하여 msg.sender가 저장되도록 작성
2. msg.sener의 ownerZombieCount라는 증가. (++ 연산자)

04 Crypto Zombie(2)

Chapter4

목표: require 이해하기 – 함수 호출 조건 걸기

```
function sayHiToVitalik(string _name) public returns (string) {
// _name이 "Vitalik"인지 비교한다. 참이 아닐 경우 에러 메시지를 발생하고 함수를 벗어난다
// (참고: 솔리디티는 고유의 스트링 비교 기능을 가지고 있지 않기 때문에
// 스트링의 keccak256 해시값을 비교하여 스트링 값이 같은지 판단한다)

    require(keccak256(_name) == keccak256("Vitalik"));
    // 참이면 함수 실행을 진행
    return "Hi!";
}
```

- require(<조건>) ➔ 해당 조건이 참이면 함수 진행, 해당 조건이 거짓이면 에러발생
  → 잔여가스 반환
  ➔ 함수 호출의 조건을 정할 수 있음.

실습

[createRandomZombie 함수 수정]

- 해당 함수를 어떤 조건 없이 호출할 수 있게 하면, 소유주 하나가 무한개의 좀비를 생성할 수 있음 → 게임이 재미가 없음.
- require함수를 사용해서 오직 유저들이 첫 좀비를 만들때만 호출 가능하게 작성

1. 유저들이 첫 좀비를 만들때만 require를 통과하도록 작성하여라
   ** 힌트: 유저가 좀비가 없으면 ownerZombieCount[msg.sender]의 값은 0이다.

04 Solidity Tutorial

상속 (is)

- 상속받으면 부모 Contract의 모든 기능(상태(속성) + 함수) 가져옴.
- 재사용성!
- 코드를 로직별로 잘 구분해서 정리하자!

```
contract Doge {
  function catchphrase( ) public returns (string) {
    return "So Wow CryptoDoge";
  }
}
contract BabyDoge is Doge {
  function anotherCatchphrase( ) public returns (string) {
    return "Such Moon BabyDoge";
    // Doge의 catchphrase 함수 사용 가능.
  }
}
```

04 Solidity Tutorial

import (다른 모듈을 이용하기)

```
1   pragma solidity ^0.4.19;
2
3   // 여기에 import 구문을 넣기
4   import "./zombiefactory";
5
6   contract ZombieFeeding is ZombieFactory {
7
8   }
9
```

- 현재경로: `.` (온점)
- 상위경로: `..` (온점 2개)

➔ import 후 비로소 ZombieFactory 를 이용할 수 있음.

```
contract ZombieFactory {

    event NewZombie(uint zombieId, string name, uint dna);

    uint dnaDigits = 16;
    uint dnaModulus = 10 ** dnaDigits;

    struct Zombie {
        string name;
        uint dna;
    }

    Zombie[] public zombies;

    mapping (uint => address) public zombieToOwner;
    mapping (address => uint) ownerZombieCount;

    function _createZombie(string _name, uint _dna) private {
        uint id = zombies.push(Zombie(_name, _dna)) - 1;
        zombieToOwner[id] = msg.sender;
        ownerZombieCount[msg.sender]++;
        NewZombie(id, _name, _dna);
    }

    function _generateRandomDna(string _str) private view returns (uint) {
        uint rand = uint(keccak256(_str));
        return rand % dnaModulus;
    }

    function createRandomZombie(string _name) public {
        require(ownerZombieCount[msg.sender] == 0);
        uint randDna = _generateRandomDna(_name);
        _createZombie(_name, randDna);
    }

}
```

04 Crypto Zombie(2)

Chapter5 & Chapter6

목표: 상속 이해하기 (상속과 is 키워드) + import (모듈 불러오기)
추가내용: 다음 슬라이드 참조

실습

ZombieFactory를 상속받는 ZombieFeeding 만들기

1. ZombieFeeding.sol이라는 파일 생성
2. ZombieFactory import
3. ZombieFactory를 상속받는 ZombieFeeding contract를 ZombieFeeding.sol파일에 작성

*** ZombieFactory는 Zombie 생성만 담당하는 Contract*
*** ZombieFeeding은 Zombie 번식만 담당하는 Contract*

04 Crypto Zombie(2)

Chapter7 & Chapter8 & Chapter9

목표: storage와 memory
추가내용: 다음슬라이드

- 기본적으로 다음과 같음
  - 함수 외부 변수 = State = Storage
  - 함수 내부 변수 = 임시변수(지역변수) = memory

- 구조체나 배열같은 경우는 다름
  - 컴퓨터 처리 방식때문에 컴퓨터가 헷갈려함 (memory를 써야? Storage를 써야?)
  → 정해줘야 함. <memory: 값만 단순 복사, Storage 포인터 복사>

실습

[먹이 먹고 번식하는 능력 부여!] – ZombieFeeding.sol
- 좀비가 다른 생명체를 먹을 때, DNA를 조합해서 새로운 좀비를 만들도록 할 예정
- 먹이는 오직 주인만이 줄 수 있음.

1. feedAndMultiply라는 public 함수를 생성한다. 인자는 (uint _zombieId, uint _targetDna)
2. 주인만이 먹이를 줄 수 있도록 구성(require문 활용)
   ** hint: msg.sender와 비교할 값은?
3. 먹이를 먹는 대상 좀비 가져오기
   (_zombieId로 Zombie 조회!)
   값을 변경할 예정이므로 storage변수
4. 새로운 좀비의 DNA 구현 (기존 좀비와 targetDna를 평균)
5. 새로운 좀비의 DNA를 얻은 후 _createZombie 호출
   (zombie의 name은 “NoName”으로 한다.
6. (문제점 찾기)➔ 해결하기
   // private은 해당 contract에서만 사용 가능 (private vs internal vs external vs public)

```
contract SandwichFactory {
    struct Sandwich {
        string name;
        string status;
    }
    Sandwich[] sandwiches;

    function eatSandwich(uint256 _index) public {
        Sandwich mySandwich = sandwiches[_index]; // 에러 발생 (구조체 혹은 배열의 경우)

        // 그러므로 `storage` 키워드를 활용 경우
        Sandwich storage mySandwich = sandwiches[_index];
        // `mySandwich`는 저장된 `sandwiches[_index]`를 가리키는 포인터(reference)이다.

        mySandwich.status = "Eaten!";
        // 실제 값 변경 --> `sandwiches[_index]`을 영구적으로 변경한다.

        // 단순 복사를 위하면 `memory`를 이용
        Sandwich memory anotherSandwich = sandwiches[_index + 1];
        // `anotherSandwich`는 단순히 메모리에 데이터를 복사(임시변수?) --> 값을 변경하더라도 영구변경X

        anotherSandwich.status = "Eaten!"; //`sandwiches[_index + 1]`에는 아무런 영향을 끼치지 않음.
        sandwiches[_index + 1] = anotherSandwich; // sandwiches[_index + 1]값을 직접 변경해야 함.
    }
}
```

04 Crypto Zombie(2)

변수의 저장 위치: Storage / Memory

- **Solidity는 기본적으로 storage. (가스를 소모)**
- **단순 복사를 위해서는 memory 사용(가스를 소모x)**

EVM에는 데이터를 저장할 수있는 3 개의 위치 존재. 스택, 메모리, 스토리지.

- **스택** : EVM의 모든 계산은 스택에서 수행됩니다. 명령어의 피연산자는 스택에서 가져 오며 중간 작업의 결과도 스택에 저장
- **메모리** : 함수 인수, 지역 변수 및 반환 값과 같은 임시 데이터를 저장하는 데 사용됩니다. x86-64 시스템의 RAM (Random Access Memory)과 마찬가지로 메모리는 휘발성입니다. 전원을 끄면 데이터가 손실
- **스토리지** : EVM의 스토리지는 키를 값에 매핑하는 영구 키-값 저장소이며 키는 256 비트 워드입니다

```
Struct Sandwich{
  string status;
  uint number;
}

Sandwich[] sandwiches;
```

** 함수의 인자에서 storage vs memory 입력 강제(solidity >=0.5)
** (함수에서는 **string 혹은 구조체, 배열 등을 인자로서 전달 받을 때**, storage or memory 입력 강제함)
이는 memory에 할당하여 휘발성이 있는지, 아니면 실제 state변수를 전달받아 referenc로 받을지(state값 변경 가능) 선택함

04 Crypto Zombie(2)

Chapter10

목표: Contract에서 다른 Contract 호출하기

```
contract LuckyNumber {
    mapping(address => uint256) numbers;

    function setNum(uint256 _num) public {
        numbers[msg.sender] = _num;
    }

    function getNum(address _myAddress) public view returns (uint256) {
        return numbers[_myAddress];
    }
}
```

- 우리가 소유하지 않은 Contract를 호출 할 때에는 **Interface**가 있어야 함.
- 위와 같은 contract가 있을시에 통신하기 위해선 다음처럼 interface를 정의할 필요가 있음.

```
interface NumberInterface {
    function getNum(address _myAddress) public view returns (uint256);
}
```

- interface는 사용할 함수에 대해 선언만 할 뿐임!
- 함수 본문은 작성 X

실습

[cryptoKitties 인터페이스 작성] – 우측 코드는 크립토키티 컨트랙트의 함수!

1. KittyInterface 작성 + interface 내에서 getKitty 함수 선언

```
function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
) {
    Kitty storage kit = kitties[_id];

    // if this variable is 0 then it's not gestating
    isGestating = (kit.siringWithId != 0);
    isReady = (kit.cooldownEndBlock <= block.number);
    cooldownIndex = uint256(kit.cooldownIndex);
    nextActionAt = uint256(kit.cooldownEndBlock);
    siringWithId = uint256(kit.siringWithId);
    birthTime = uint256(kit.birthTime);
    matronId = uint256(kit.matronId);
    sireId = uint256(kit.sireId);
    generation = uint256(kit.generation);
    genes = kit.genes;
}
```

04 Crypto Zombie(2)

Chapter11

목표: 인터페이스 활용하기

```
interface NumberInterface {
    function getNum(address _myAddress) public view returns (uint256);
}
```

- 위와 같이 interface가 정의되면
- 아래와 같이 컨트랙트 내에서 호출할 수 있음.

```
contract MyContract {
  address NumberInterfaceAddress = 0xab38...
  // ^ 이더리움상의 FavoriteNumber 컨트랙트 주소이다
  NumberInterface numberContract = NumberInterface(NumberInterfaceAddress)
  // 이제 `numberContract`는 다른 컨트랙트를 가리키고 있다.

  function someFunction() public {
    // 이제 `numberContract`가 가리키고 있는 컨트랙트에서 `getNum` 함수를 호출할 수 있다:
    uint num = numberContract.getNum(msg.sender);
    // ...그리고 여기서 `num`으로 무언가를 할 수 있다
  }
}
```

- 필요사항: Contract Account Address + 인터페이스

실습

[cryptoKitties getKitty 호출]

1. 크립토키티의 Address: 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d
2. address ckAddress 변수에 할당하고
3. kittyInterface로 정의 및 선언하기(전역으로)

```
function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
) {
    Kitty storage kit = kitties[_id];

    // if this variable is 0 then it's not gestating
    isGestating = (kit.siringWithId != 0);
    isReady = (kit.cooldownEndBlock <= block.number);
    cooldownIndex = uint256(kit.cooldownIndex);
    nextActionAt = uint256(kit.cooldownEndBlock);
    siringWithId = uint256(kit.siringWithId);
    birthTime = uint256(kit.birthTime);
    matronId = uint256(kit.matronId);
    sireId = uint256(kit.sireId);
    generation = uint256(kit.generation);
    genes = kit.genes;
}
```

04 Crypto Zombie(2)

Chapter12

목표: 인터페이스 활용하기2

실습

[feedOnKitty함수 생성]
1. feedOnKitty 함수 생성 (uint _zombieId, uint _kittyId) public
2. 함수내에서 uint kittyDna 변수 선언
3. _kittyID를 전달하여 kittyInterface에 getKitty함수 호출
4. genes를 kittyDna 에 저장
5. feedAndMultiply함수 호출 → _zombieId와 kittyDna 전달

```
function getKitty(uint256 _id) external view returns (
    bool isGestating,
    bool isReady,
    uint256 cooldownIndex,
    uint256 nextActionAt,
    uint256 siringWithId,
    uint256 birthTime,
    uint256 matronId,
    uint256 sireId,
    uint256 generation,
    uint256 genes
) {
    Kitty storage kit = kitties[_id];

    // if this variable is 0 then it's not gestating
    isGestating = (kit.siringWithId != 0);
    isReady = (kit.cooldownEndBlock <= block.number);
    cooldownIndex = uint256(kit.cooldownIndex);
    nextActionAt = uint256(kit.cooldownEndBlock);
    siringWithId = uint256(kit.siringWithId);
    birthTime = uint256(kit.birthTime);
    matronId = uint256(kit.matronId);
    sireId = uint256(kit.sireId);
    generation = uint256(kit.generation);
    genes = kit.genes;
}
```

04 Crypto Zombie(2)

Chapter13

목표: if문 이해하기
추가: 다음 슬라이드 참조

dna중 앞에 12자리만 이용할 예정!
- 마지막 dna2자리로 특별한 특성 만들자
- 고양이는 마지막 dna가 99가 되었으면 한다!

실습

[if문 활용]
1. 먼저, feedAndMultiply 함수 정의를 변경하여 _species라는 string을 세번째 인자 값으로 전달받도록 한다.

2. 그 다음, 새로운 좀비 DNA를 계산한 후에 if 문을 추가하여 _species와 "kitty" 스트링 각각의 keccak256 해시값을 비교하도록 한다.

3. if 문 내에서 DNA 마지막 2자리를 99로 대체하고자 한다.
   ** newDna = newDna - newDna % 100 + 99; 로직을 이용하는 것이다.
   설명: newDna가 334455라고 하면 newDna % 100는 55이고, 따라서 newDna - newDna % 100는 334400이다. 마지막으로 여기에 99를 더하면 334499를 얻게 된다.

4. 마지막으로, feedOnKitty 함수 내에서 이뤄지는 함수 호출을 변경해야 한다. feedAndMultiply가 호출될 때, "kitty"를 마지막 인자값으로 전달한다.

04 Crypto Zombie(2)

조건문 if

```
function eatBLT(string sandwich) public {
    // 스트링 간의 동일 여부를 판단하기 위해 keccak256 해시 함수 + encodedPacked를 이용해야 한다는 것을 기억하자
    if (keccak256(abi.encodePacked(sandwich)) == keccak256(abi.encodePacked("BLT"))) {
        eat();
    }
}
```

```
if(조건) {
  // 조건이 참이면 실행
 }
```

- sandwich로 받은 값과 BLT로 받은 값이 같은지 확인
- solidity 0.4.19에는 문자열 비교가 없다.