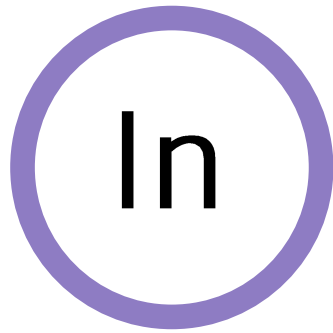


이더리움 DApp

Solidity 언어 실습

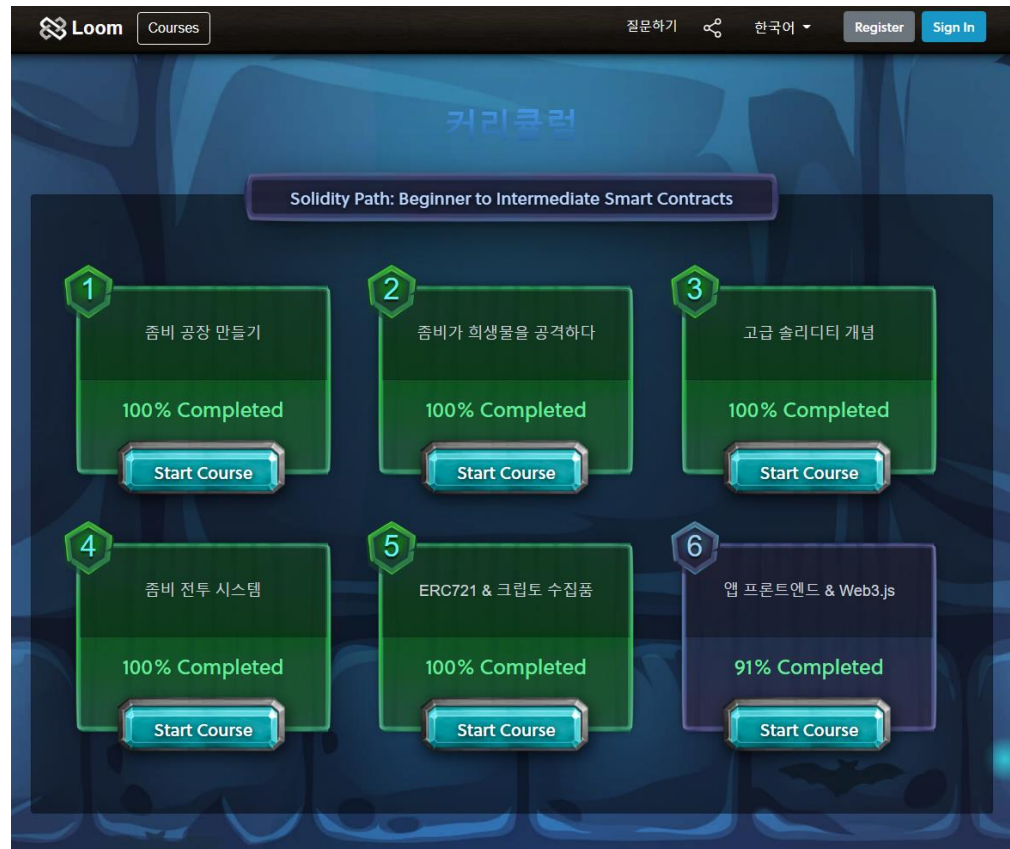




Solidity

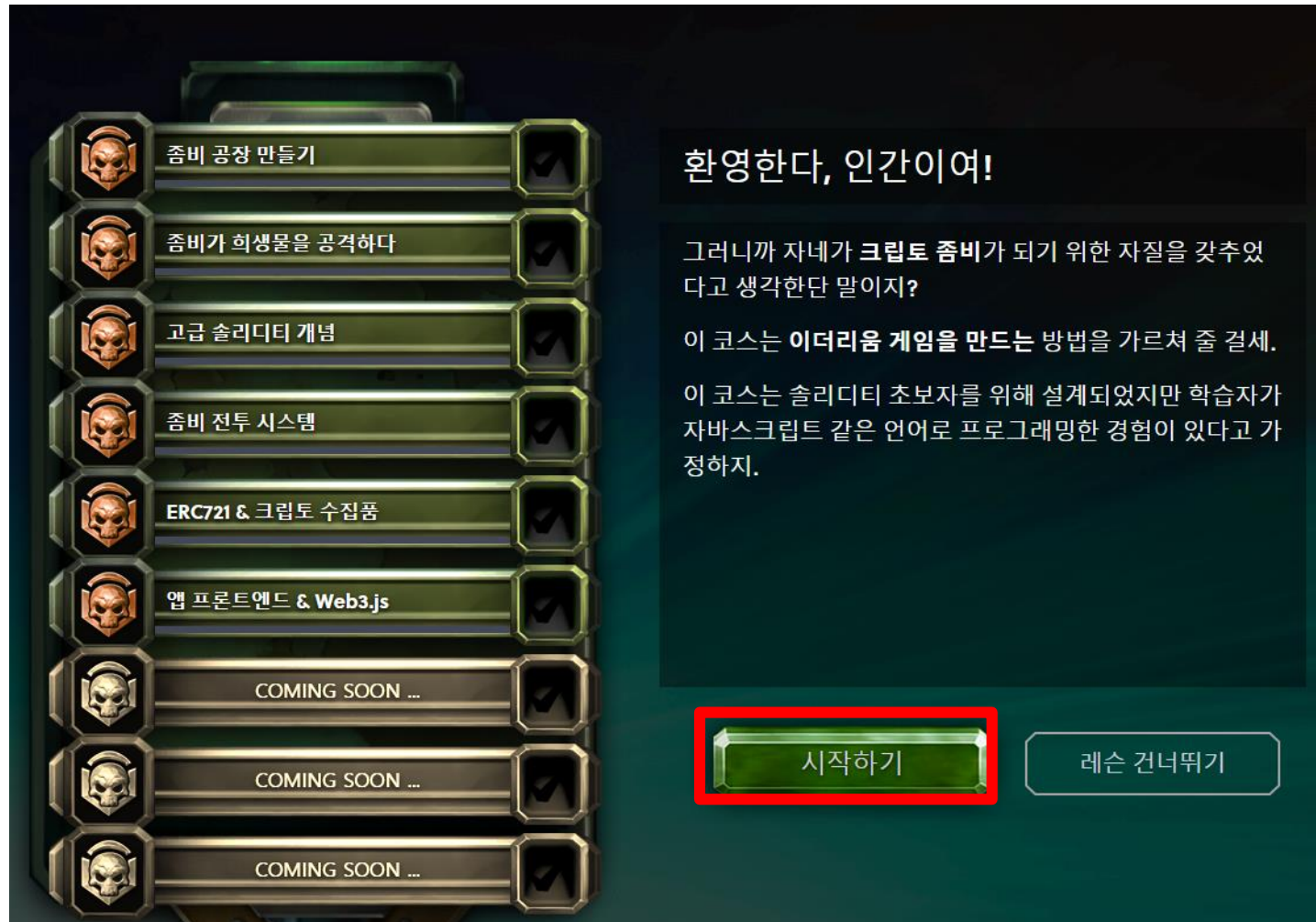
01 크립토 좀비

<https://cryptozombies.io/ko/course>



Loom Network 에서 제작한 Solidity 학습 튜토리얼

01 크립토 좀비



튜토리얼을 따라하면서 필요한 개념을 숙지

01 크립토 좀비

Loom Courses

챕터 1: 레슨 개요

레슨 1에서 자네는 좀비 군대를 건설하기 위한 "좀비 공장"을 만들 겠세.

- 우리 공장은 군대 내 모든 좀비의 데이터베이스를 유지할 겠세
- 우리 공장은 새로운 좀비를 생성하는 함수를 가질 겠세
- 각 좀비는 랜덤하고 독특한 외모를 가질 겠세

이후 레슨에서 인간이나 다른 좀비를 공격하는 능력 등 더 많은 기능을 추가할 것이네. 그전에 새로운 좀비를 생성하는 기본 기능을 추가해야 하네.

좀비 DNA가 활용되는 방법

좀비의 외모는 "좀비 DNA"에 따라 달라질 것이네. 좀비 DNA는 다음과 같이 간단하게 16자리 정수이네:

8356281049284737

실제 DNA처럼 이 숫자의 각 부분은 좀비가 가진 개별 특성과 매핑이 될 것이네. 처음 2자리 숫자는 좀비의 머리 타입과 연결이 되고, 그 다음 2자리 숫자는 좀비의 눈 모양과 연결이 되지.

참고: 본 튜토리얼에서는 단순화해서 좀비의 머리 타입이 7개만 있을 거네 (2 자리 숫자로 100 가지 경우의 수를 구할 수 있지만). 나중에 머리 타입을 다양하게 하고 싶다면 더 많은 머리 타입을 추가할 수도 있을 거네.

이렇게면, 위의 예시 DNA에서 첫 2자리 숫자 83이네. 이를 좀비 머리 타입으로 매핑하기 위해 우리는 $83 \% 7 + 1 = 7$ 연산을 하지. 그래서 이 DNA를 가진 좀비는 7번째 좀비 머리 타입을 갖게 되지.

우측 패널에서 머리 유전자 슬라이더를 7번째 머리 타입 (산타 모자)로 이동시킨 다음, 83에 어떤 특성이 대응되는지 살펴보게.

직접 해보기

1. 페이지 우측의 슬라이드를 가지고 놀아 보게. 다양한 숫자 값에 따라 좀비의 외모가 어떻게 달라지는지 실험해 보게.

자, 충분히 가지고 놀았겠지. 계속 진행할 준비가 되었다면, 아래 "다음 챕터"를 클릭하게. 솔리디티 학습을 본격적으로 시작해 보세!

ZOMBIE
CRYPTO-COLLECTABLES

머리 유전자: 1

눈 유전자: 1

셔츠 유전자: 1

피부색 유전자: 0

눈 색깔 유전자: 0

옷 색깔 유전자: 0

튜토리얼을 따라하면서 필요한 개념을 숙지

01 크립토 좀비

version0이 꽤 업데이트 되었으므로
truffle version에 재구성 할 예정

02 Solidity 공식문서

(version 0.5.16)

<https://docs.soliditylang.org/en/latest/>

version이 꽤 업데이트 되었으므로
truffle version에 재구성 할 예정

(version 0.5.16)

03 Crypto Zombie

다음은 Loom Network에서 만든 CryptoZombie를 0.5.16버전에 맞게 재구성하였습니다.

Zombie Factory 만들기

- 모든 좀비를 기록
- 좀비 생성 함수
- 각 좀비는 DNA를 통해 랜덤하고 독특한 외모 유지

자세한 사항은 링크 참조

truffle을 이용해 프로젝트를 진행할 예정

- mkdir cryptozombie_0.5.16
- cd cryptozombie_0.5.16
- truffle init
- code .


<https://cryptozombies.io/ko/lesson/1/chapter/1>

03 Crypto Zombie(1)

Chapter2

목표: ZombieFactory라는 contract를 만들 예정
version: truffle의 solc.js 버전과 맞춰라

> truffle create contract ZombieFactory

```
contracts >  ZombieFactory.sol
1  pragma solidity ^0.5.16;
2
3  contract ZombieFactory {
4
5
6  }
```

- pragma solidity는 버전을 명시해줘야함.
- 코드의 마지막은 세미콜론(;)
- contract <Contract 이름> {}
// 코드블록은 중괄호{}로 감싸줌.
- // 주석은 // 와 /* */ 를 사용함.

03 Crypto Zombie(1)

Chapter3

목표: 상태 변수와 DataType(자료형) 이해하기

추가 내용: 다음 슬라이드 참조

실습

우리의 좀비 DNA는 16자리 숫자로 결정됨!
dnaDigits라는 uint를 선언하고 16이라는 값을 배정해 보자.

예시:

```
contract Example {  
    // 이 변수는 블록체인에 영구적으로 저장된다  
    uint myUnsignedInteger = 100;  
}
```

이 예시 컨트랙트에서는 myUnsignedInteger라는 uint를 생성하여 100이라는 값을 배정했네.

- 상태변수는 contract 바로 안에 존재하는 변수 (함수 내부 X)
- 상태변수는 블록체인상에 영구히 기록됨

03 Crypto Zombie(1)

Chapter3

목표: 상태 변수와 DataType(자료형) 이해하기
추가 내용: 다음 슬라이드 참조(DataType과 연산)

예시:

```
contract Example {  
    // 이 변수는 블록체인에 영구적으로 저장된다  
    uint myUnsignedInteger = 100;  
}
```

이 예시 컨트랙트에서는 myUnsignedInteger라는 uint를 생성하여 100이라는 값을 배정했네.

- 상태변수는 contract 바로 안에 존재하는 변수 (함수 내부 X)
- 상태변수는 블록체인상에 영구히 기록됨

실습

우리의 좀비 DNA는 16자리 숫자로 결정됨!
dnaDigits라는 uint를 선언하고 16이라는 값을 배정해 보자.

Chapter4

목표: 연산자 이해하기
추가 내용: 다음 슬라이드 참조(DataType과 연산)

- 덧셈: $x + y$
- 뺄셈: $x - y$,
- 곱셈: $x * y$
- 나눗셈: x / y
- 모듈로 / 나머지: $x \% y$ (이클테면, $13 \% 5$ 는 3이다. 왜냐면 13을 5로 나누면 나머지가 3이기 때문이다)

솔리디티는 지수 연산도 지원하지 (즉, "x의 y승", x^y 이지):

```
uint x = 5 ** 2; // 즉,  $5^2 = 25$ 
```

실습

1. dnaModulus라는 uint형 변수를 생성하고 10의 dnaDigits승을 배정한다.

- 우리의 좀비 DNA가 16자리 숫자가 되도록 하기 위해 unit형 변수를 생성하고 10^{16} 값을 배정!
- 이 값을 이후 모듈로 연산자 %와 함께 이용하여 16자리보다 큰 수를 16자리 숫자로 줄일 수 있음.

04 Solidity Tutorial

Primitive Data Types

- Booleans (Bool 형)
- Integers (정수)
- Fixed Point Numbers(실수)
- Address
- String
- bytes(바이트: 문자열, 정수 등 다양한 데이터 저장 가능. bytes1, bytes2, ... bytes32)

04 Solidity Tutorial

Boolean (bool)

- true vs false

- **!** not
- **&&** and 연산
- **||** or 연산
- **==** 같은지 비교
- **!=** 다른지 비교

예)

!true

true && false

true || false

1 == 3

04 Solidity Tutorial

Integers

int / uint:

- int: 부호 있는 정수
- uint: 부호 없는 정수

[연산자]

- 산술연산 (+, -, *, /, %(모듈러 연산), *(거듭제곱))
- 비교연산: (<=, <, ==, !=, >=, >) // 결과가 bool
- Bit 연산: (&, |, ^(xor), ~(not)) //
- Shift 연산: (<<, >>)

integer에 저장할 bit수를 함께 저장할 수 있음

- uint(uint256)
- uint128
- uint64
- uint32
- uint16
- uint8

Fixed Point Numbers

fixed vs ufixed

- 정의 시에는 **fixedMxN** 또는 **ufixedMxN**
- 예) ufixed128x18

M: 정수부 (8의 배수 8~256)
N: 소수부 자리수 (0~80)

비교연산, 산술연산 가능

04 Solidity Tutorial

Address

- address (주소)
- 이더리움 계좌 주소를 의미함.
- 각 계정은 은행 계좌번호와 같은 주소(Address를 가지고 있음)

0x0cE446255506E92DF41614C46F1d6df9Cc
969183

//특정 계정을 가리키는 고유 식별자, 이 주소를
통해 거래 가능

String

- 문자열.
1. bytes32(byte[]): 바이트 단위로 저장 (아스키코드)
 2. string: utf-8 encoding 데이터 저장.

03 Crypto Zombie(1)

Chapter5

목표: 구조체 이해하기

예시:

```
contract Example {  
    // 이 변수는 블록체인에 영구적으로 저장된다  
    uint myUnsignedInteger = 100;  
}
```

이 예시 컨트랙트에서는 `myUnsignedInteger`라는 `uint`를 생성하여 100이라는 값을 지정했네.

- 구조체를 사용하면 압축해서 저장 가능 (메모리 구조상)
- 새로운 데이터 타입을 만든다고 생각해도 좋음.

실습

1. Zombie라는 struct를 생성한다.
2. 우리의 Zombie 구조체는 name (string형)과 dna (uint형)이라는 2가지 특성을 가진다.

Chapter6

목표: 배열 이해하기

추가 내용: 다음 슬라이드 참조(동적배열&정적배열, 접근제어자)

```
// 2개의 원소를 담을 수 있는 고정 길이의 배열:  
uint[2] fixedArray;  
// 또다른 고정 배열으로 5개의 스트링을 담을 수 있다:  
string[5] stringArray;  
// 동적 배열은 고정된 크기가 없으며 계속 크기가 커질 수 있다:  
uint[] dynamicArray;
```

구조체도 배열로 가능

```
Person[] people; // 이는 동적 배열로, 원소를 계속 추가할 수 있다.
```

Public 선언 가능 – 다른 Contract에서 읽을 수 있음.

```
Person[] public people;
```

실습

1. Zombie 구조체의 public 배열을 생성하고 이름을 zombies로 한다.

04 Solidity Tutorial

배열

정적 배열(Fixed Size Array)

- 고정 길이의 배열

예)

```
uint[2] fixedArray;  
string[10] stringFixedArray;  
Person[10] fixedPeople;
```

동적 배열 (Dynamic Size Array)

- 고정된 크기 X
- 계속 크기가 커질 수 있는 배열

예)

```
Uint[] dynamicArray;  
Person[] people;
```

04 Solidity Tutorial

배열에 추가하기 – push(동적배열)

```
uint[ ] numbers;  
numbers.push(5);  
numbers.push(10);  
numbers.push(15);  
// numbers 배열은 [5, 10, 15]과 같음
```

push함수는 호출 후에 배열의 새로운 길이를 반환!

- 위코드에서 numbers.push(15)코드의 반환값은 3

```
struct Person {  
    uint age;  
    string name;  
}
```

```
Person[ ] public people;  
// Person 구조체를 만들고 , Person 의 객체를 갖는 people 동적 배열 생성
```

```
Person satoshi = Person(172, "Satoshi"); // 새로운 Person 객체 생성  
people.push(satoshi); // 생성한 객체를 people 배열에 추가
```

04 Solidity Tutorial

public vs private ?

- **public**: 누구나 Contract의 함수를 호출하고 코드를 실행 가능
 - **private**: Contract 내에서만 해당 변수나 함수에 접근 가능
- Solidity의 default는 public(입력하지 않으면 public)
- 보안을 생각하면 **Private** 함수가 필요.
- **private** 함수는 관례적으로 변수 앞에 **_** 를 붙인다.

```
uint[ ] numbers;  
  
function _addToArray(uint _number) private {  
    numbers.push(_number);  
}
```

Contract 안의 다른 함수들만이 **_addToArray** 함수를 호출가능하고, 이 함수를 호출해야 **numbers** 배열에 객체를 추가할 수 있음

04 Solidity Tutorial

접근제어자 Internal / external

- **Internal:** 함수가 정의된 컨트랙트를 상속하는 컨트랙트에서도 접근 가능
- **external:** 함수가 컨트랙트 바깥에서만 호출될 수 있고 컨트랙트 내의 다른 함수에 의해 호출될 수 없음.

```
contract Sandwich {
    uint private sandwichesEaten = 0;

    function eat() internal {
        sandwichesEaten++;
    }
}

contract BLT is Sandwich {
    uint private baconSandwichesEaten = 0;

    function eatWithBacon() external returns (string) {
        baconSandwichesEaten++;
        // eat 함수가 internal로 선언되었기 때문에 여기서 호출이 가능
        eat();
    }
}
```

03 Crypto Zombie(1)

Chapter7 & Chapter8

목표: 함수 선언하기

```
function eatHamburgers(string _name, uint _amount) {  
  
}
```

```
eatHamburgers("vitalik", 100);
```

- 함수의 인자는 _를 앞에 두는 것이 관례(state와 구분을 위함)
- 새로운 데이터 타입을 만든다고 생각해도 좋음.

Chapter9

목표: private&public실습

```
uint[] numbers;  
  
function _addToArray(uint _number) private {  
    numbers.push(_number);  
}
```

- private은 함수 이름 앞에 _를 붙이는 것이 관례(public과 비교 위함)
- solidity에서 기본적으로 함수는 public
- 모든 함수가 public이면 누구든 민감한 함수를 호출할 수 있다
→ 보안에 취약할 수 있다.

실습

1. _createZombie 라는 private 함수 생성 인자는 2개 (string _name, uint _dna)
2. 해당 함수는 전달받은 인자들로 새로운 Zombie를 생성하고 zombies배열에 추가하는 함수

03 Crypto Zombie(1)

Chapter10

목표: 함수 제어자와 ReturnValue(반환값)

추가 내용: 다음 슬라이드 참조(반환값, 함수 제어자)

```
string greeting = "What's up dog";

function sayHello() public returns (string) {
    return greeting;
}
```

- 반드시 return value가 있을시엔 함수 선언부에 returnType 입력 필수!

실습

1. _generateRandomDna라는 private 함수를 만들고 인자는 (string _str) 반환타입은 (uint)
2. 이 함수는 컨트랙트 변수를 보지만 변경하지는 않을 것이므로 view로 선언.
3. 함수내용은 _str을 keccak256 해시값(fake random)을 구해서 uint256 rand에 저장
→ 해당 rand값을 16자리로 만들어서 반환 (hint: rand % ?)

Chapter11

목표: keccak256해시함수와 형변환(type casting)

```
//6e91ec6b618bb462a4a6ee5aa2cb0e9cf30f7a052bb467b0ba58b8748c00d2e5
keccak256("aaaab");
//b1f078126895a1424524de5321b339ab00408010b7cf0e6ed451514981e58aa9
keccak256("aaaac");
```

- keccak256 해시함수 내장되어 있음

```
uint8 a = 5;
uint b = 6;
// a * b가 uint8이 아닌 uint를 반환하기 때문에 에러 메시지가 난다:
uint8 c = a * b;
// b를 uint8으로 형 변환해서 코드가 제대로 작동하도록 해야 한다:
uint8 c = a * uint8(b);
```

- 대부분의 프로그래밍에서 datatype의 이해와 type casting은 중요함.

04 Solidity Tutorial

return

- 함수에서 어떤 값을 return 받으려는지 반환하는 값의 type을 명시

```
string greeting = "What's up dog";  
  
function sayHello( ) public returns (string) {  
    return greeting;  
}
```

04 Solidity Tutorial

다수의 반환값 설정 가능 (Multiple Returns)

- 다수의 반환값 처리 가능
- **returns** 에 반환값을 적고 마찬가지로 함수 호출 시 다수 값을 할당

```
function multipleReturns() internal returns(uint a, uint b, uint c) {  
    return (1, 2, 3);  
}  
  
function processMultipleReturns() external {  
    uint a;  
    uint b;  
    uint c;  
    // 다음과 같이 다수 값을 할당  
    (a, b, c) = multipleReturns();  
}  
  
// 혹은 단 하나의 값에만 관심이 있을 경우:  
function getLastReturnValue() external {  
    uint c;  
    // 다른 필드는 빈칸으로 작성  
    ( , ,c) = multipleReturns();  
}
```


04 Solidity Tutorial

view

- 함수가 데이터를 보기만 하고 변경하지 않을 때는 **view** 함수로 선언

→ gas 소모 X

```
function sayHello() public view returns (string)
```

pure

- 어떤 데이터에도 접근하지 않음

→ gas 소모 X

```
function _multiply(uint a, uint b) private pure returns (uint) {  
    return a * b;  
} // 이 함수는 App에서 읽는 것도 하지 않고, 반환값이 전달된 인자값에 따라서 달라짐
```

04 Solidity Tutorial

view vs pure

- view 사용은 해당 함수를 실행해도 어떤 데이터도 저장/변경 X
- pure 사용은 해당 함수가 어떤 데이터도 블록체인에 저장하지 않을 뿐만 아니라 블록체인으로 부터 어떤 데이터도 읽지 않음
- view, pure 가 함수에 modifier 로 붙으면 가스를 소모하지 않음

04 Solidity Tutorial

external view

```
function changename(address _owner) external view returns (uint[]) {  
    // 필요한 함수 내용들  
}
```

- view 함수는 사용자에게 의해 외부에서 호출 되었을 때, 가스를 소모하지 않음.
- view 함수는 실제로 어떤 것도 수정하지 않고 데이터를 읽기만 함
- ➔ 가스 사용을 줄이기 위해서는 external view로 설정하는 것이 gas 소모를 줄일 수 있음.

03 Crypto Zombie(1)

Chapter12 & 13

목표: 종합실습(1) & Event 이해하기

```
contract Test {
    event Deposit(address indexed _from, bytes32 indexed _id, uint _value);
    function deposit(bytes32 _id) public payable {
        emit Deposit(msg.sender, _id, msg.value);
    }
}
```

- event는 대문자로 시작이 관례
- event를 emit(발생 or 방출)할 때는 키워드 emit

```
var abi = /* abi as generated using compiler */;
var ClientReceipt = web3.eth.contract(abi);
var clientReceiptContract = ClientReceipt.at("0x1234...ab67" /* address */);

var event = clientReceiptContract.Deposit(function(error, result) {
    if (!error) console.log(result);
});
```

- event가 EVM에서 발생하면 dApp Frontend에서 이벤트를 받을 수 있고, 정해놓은 이벤트 핸들러가 실행됨.
- emit Event는 FrontEnd(Client)에 해당 contract가 실행되었음을 알림
- Event(이하 이벤트)란 트랜잭션 내에서 호출될 수 있는 일종의 리턴값이 없는 함수
- 이벤트를 호출하면 그 호출한 기록이 **Transaction Receipt**라 불리는 트랜잭션 결과에 저장됩니다. 일종의 로그(log)

실습

1. createRandomZombie라는 인자가(string _name)인 public함수를 생성한다. (public이라고 명시 해주길 바람)
2. _name을 가지고 Dna를 만들고 (_generateRandomDna 호출) uint256 randData에 저장한 후 _createZombie함수 호출: (_name과 _dna 전달)
3. NewZombie라는 event를 선언한다. zombieId (uint형), name (string형), dna (uint형)을 인자로 전달받아야 한다.
4. _createZombie 함수를 변경하여 새로운 좀비가 zombies 배열에 추가된 후에 NewZombie 이벤트를 실행하도록 구성. 이벤트를 위해 좀비의 id가 필요할 것이다.
5. 좀비의 배열에서 인덱스를 zombie의 id로 할 예정

** hint (array.push의 반환값은 array의 길이이다. 그럼 마지막에 추가한 요소의 idx: array의 길이 -1)

03 Crypto Zombie(1)

변수의 저장 위치: Storage / Memory

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

EVM에는 데이터를 저장할 수 있는 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

03 Crypto Zombie(1)

lesson1 마무리 실습

** ZombieFactory의 모든 함수의 string형 인자를 memory로 받도록 명시하자!

```
function _createZombie(string memory _name, uint256 _dna) private {  
    uint256 id = zombies.push(Zombie(_name, _dna)) - 1;  
    NewZombie(id, _name, _dna);  
}
```

solidity 0.5 부터 keccak256에 strin을 바로 전달 X

➔ keccak256(abi.encodePacked(string))으로 사용 (abi 방식으로 인코딩→ bytes로 변환) 또는 bytes(string)호출

```
function _createZombie(string memory _name, uint256 _dna) private {  
    uint256 id = zombies.push(Zombie(_name, _dna)) - 1;  
    NewZombie(id, _name, _dna);  
}
```