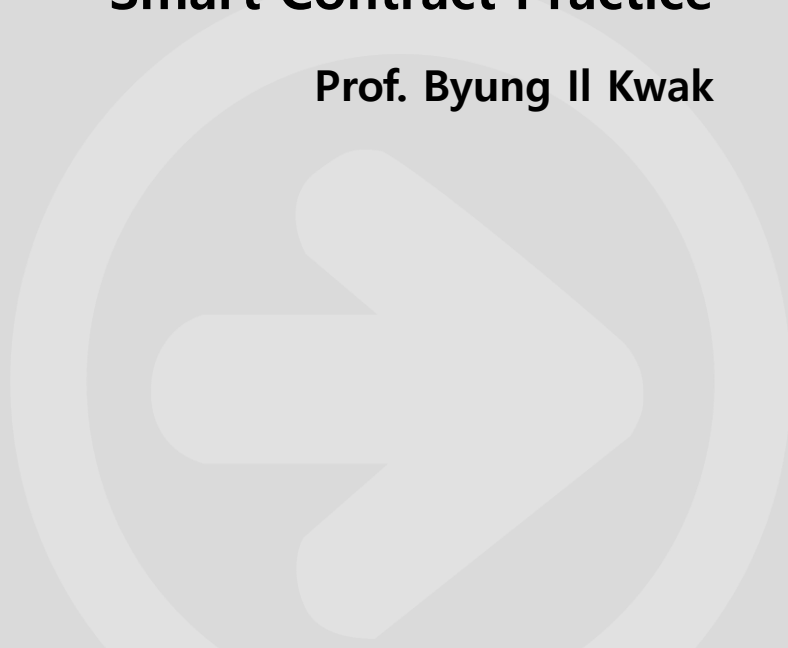




Blockchain #12

Smart Contract Practice

Prof. Byung Il Kwak



- ❑ Security of Blockchain
- ❑ Smart Contract Security

- Understanding of Smart Contract
- Smart Contract Practice

CONTENTS

- ❑ Understanding of Smart Contract

Understanding of Smart Contract

□ 스마트 컨트랙트 (Smart contract)

- 객체지향 프로그램에서 클래스와 유사한 독립적인 코드
- 또한, 데이터와 함수를 가진 배포 가능한 코드 모듈
 - 함수는 확인, 검증, 그리고 전송된 메시지를 저장하는 등의 특정한 목적을 수행함
 - 현실 세계에서 컨트랙트는 규칙, 조건, 법, 강제해야 하는 규정, 기준, 상황, 날짜 및 서명 등의 증명을 위한 항목을 다룸
- 블록체인 문맥에서 스마트 컨트랙트는 탈중앙화 문제들을 풀기 위한 컨트랙트 규칙들을 구현함
 - 게이트키퍼 역할 뿐만 아니라 규칙 엔진으로써 작동하므로, 스마트 컨트랙트를 설계할 때 신중하게 고려해야함
 - 스마트 컨트랙트를 이용한 취약점 빈번하게 발생함

Understanding of Smart Contract

□ 스마트 컨트랙트 (Smart contract)

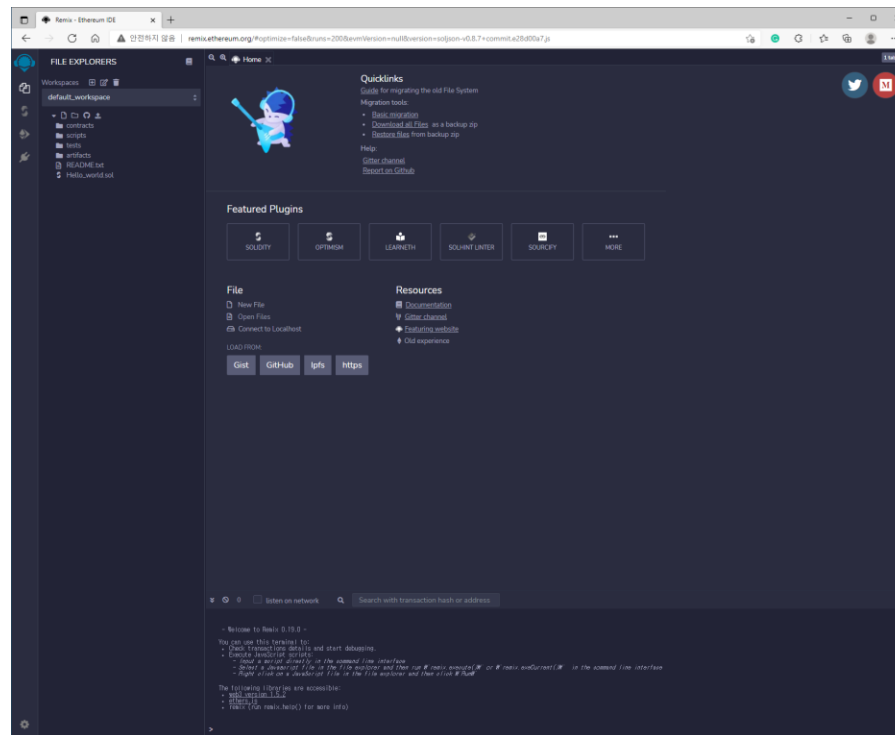
- 이더리움 플랫폼 사용을 위해서는 스마트 컨트랙트 코딩은 솔리디티를 이용해야 하며,
- 리믹스(Remix)라는 통합 개발 환경(IDE)에서 코드를 배포하고 작동 테스트를 최종적으로 수행함
- 스마트 컨트랙트는 비트코인 블록체인 프로토콜이 제공했던 기본적인 신뢰를 확장시키는 코드
 - 스마트 컨트랙트는 암호 화폐 이외의 디지털 자산을 위한 트랜잭션을 지원할 수 있는 **프로그래밍을 가능**하게 해줌
 - 스마트 컨트랙트는 각 블록체인 애플리케이션이 필요로 하는 특정한 확인과 검증을 가능하게함
 - 즉, 범용적인 애플리케이션을 위한 블록체인의 신뢰 레이어를 열어주는것

Understanding of Smart Contract

□ 통합 개발 환경

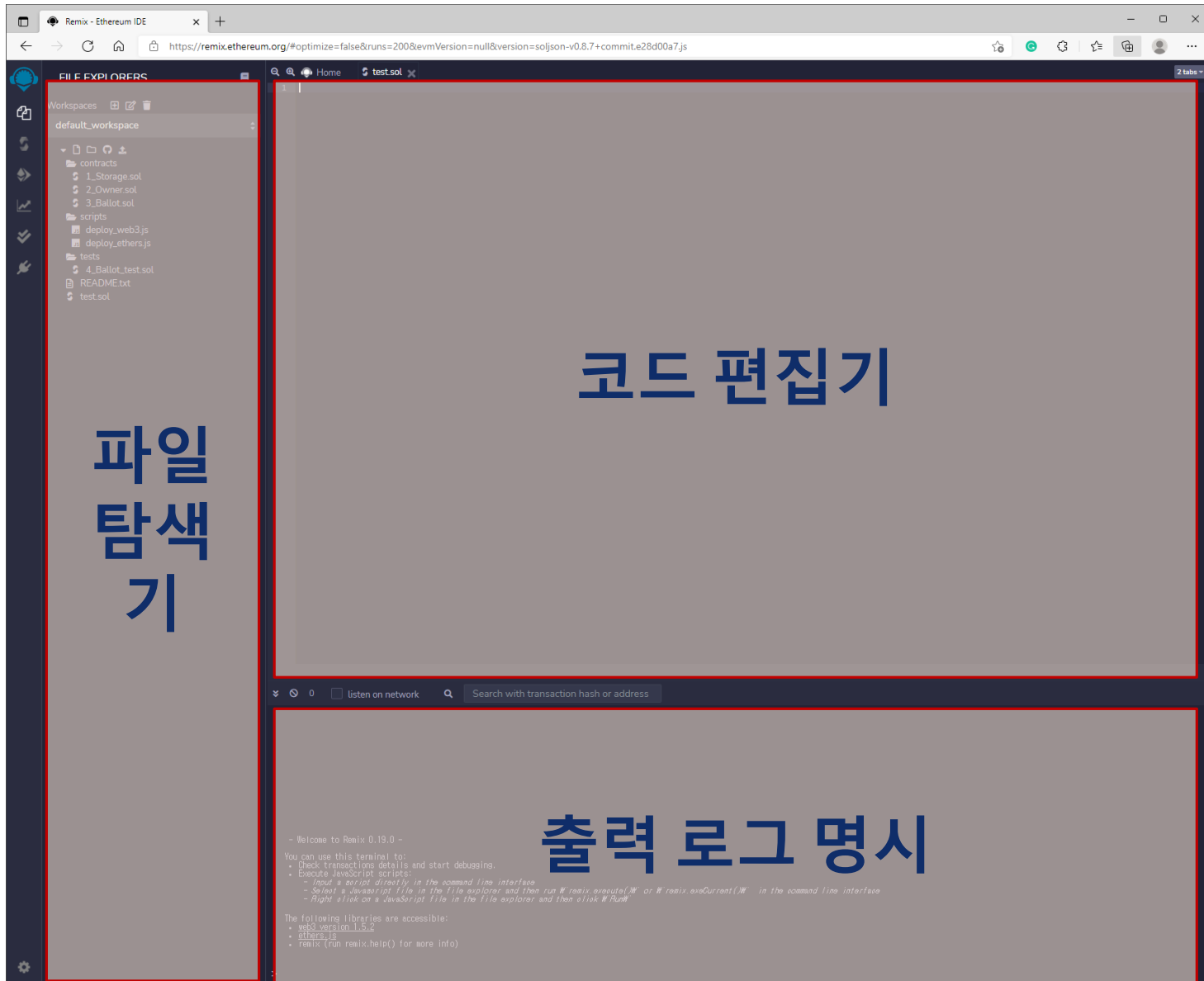
▣ 리믹스(Remix)

- 솔리디티를 사용한 계약 개발 및 테스트를 위한 웹 기반 환경으로, 실제 블록체인에 실행하는 것이 아닌, 계약 배치 테스트, 디버깅 할 수 있는 시뮬레이션 환경

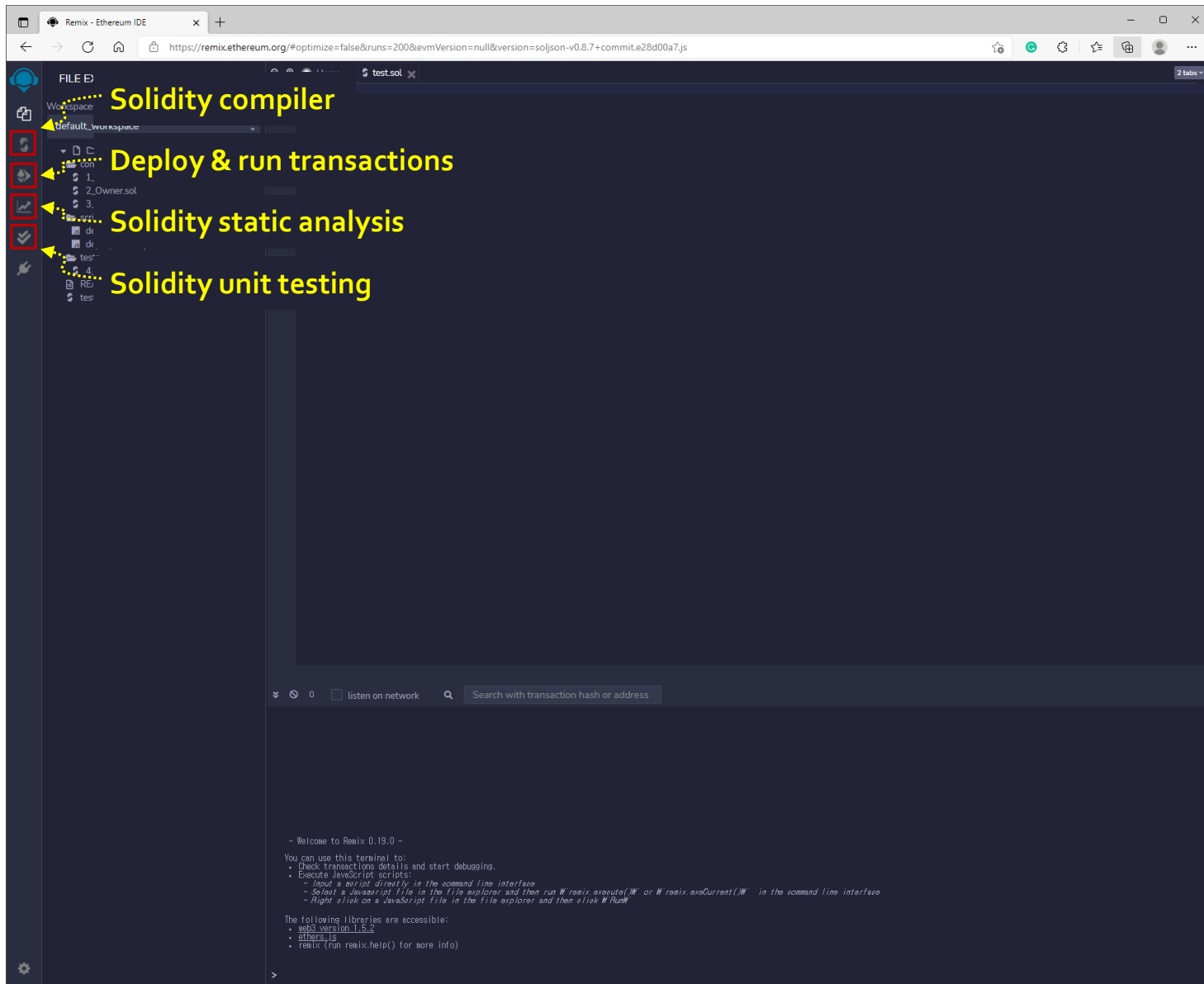


<https://remix.ethereum.org>

Understanding of Smart Contract



Understanding of Smart Contract



Understanding of Smart Contract

□ 계약 개발과 배치

- 계약을 개발하고 배치하기 위해서는 크게 '작성', '테스트', '검증', '배치' 4개 단계로 구성

- '배치' 단계 뒤에 사용자 인터페이스를 만들어 웹 서버를 통해 최종 사용자에게 제공하는 추가 단계가 있을 수 있음
- 계약에 사람의 입력이나 모니터링이 필요하지 않은 경우, 웹 인터페이스가 필요하지 않지만, 대개는 계약과 상호작용할 웹 인터페이스를 만들어야함

■ 작성

- 솔리디티로 계약 소스 코드를 작성함.
- 아무 텍스트 편집기나 써서 작성하면 됨
 - 리눅스 Vim, 윈도우 메모장 등을 통해 솔리디티 소스 코드의 구문과 형식에 맞춰 작성

Understanding of Smart Contract

□ 계약 개발과 배치

▣ 테스트

- 자동화된 방법으로 수행하는 것이 일반적
- 리믹스에서 함수를 수동으로 실행하고 결과를 검사해 수동으로 함수 테스트를 수행할 수 있음
- '계약'을 시뮬레이션 환경 (ex. 이더리움JS의 테스트 RPC, 가나슈) 또는 사설망에서 검증하고 잘 작동해 테스트를 통과하면, 공용 테스트넷, 그리고 최종적으로 실제 운영중인 블록체인에 배치할 수 있음

Understanding of Smart Contract

□ Solidity Language

- ▣ 솔리디티는 이더리움에서 계약을 프로그래밍하는 데 사용할 수 있는 도메인 특화 언어
 - 구문은 C에 가까움
- ▣ 솔리디티는 정적 타입의 언어로써, 변수타입은 컴파일 시에 검사되고, 각 변수들은 상태 변수 또는 로컬 변수로 컴파일 시에 타입이 정해져야함
 - 컴파일 시에 모든 검증과 검사를 완료해 데이터 타입의 해석 같은 특정 유형의 버그를 실행 전에 개발 단계에서 일찍 잡아낼 수 있는 장점을 가짐
 - 다만, 특정 언어를 기반으로 스마트 컨트랙트가 구성되다 보니, 취약점이 지속적으로 발견되고 있음

Understanding of Smart Contract

□ Solidity Language

▣ 타입

– Value type

- Boolean: true or false
- Integer
 - int (int8, int16, ...) : 부호 있는 정수
 - uint (uint8, uint16, ...) : 부호 없는 정수
- Address
 - balance, send, call function

Understanding of Smart Contract

□ Solidity Language

▣ Address Value type

– balance

- 잔고를 웨이(wei)단위로 리턴

– send

- 주소(이더리움의 160bit 길이 주소)에 이더를 전송할 때 사용하며, 트랜잭션 결과에 따라 true, false를 리턴

```
address to = 0x6414cc08d148dce9ebf5a2d0b7c220ed2d3203da;  
address from = this;  
if (to.balance < 10 && from.balance > 50) to.send(20);
```

– call 함수

- call, callcode, delegatecall 같은 호출 함수를 이용해 ABI가 없는 함수와 상호작용할 수 있음.
- 이 함수들은 타입 안전성이나 계약의 보안에 영향을 미쳐 안전하지 않을 수 있음. (주의해서 사용해야함!!)

Understanding of Smart Contract

□ Solidity Language

▣ Array value

- 정적 및 동적 크기 바이트 배열이 있음
- 정적 바이트 배열: bytes1 ~ bytes32

```
bytes32[10] bankAccounts;
```

- 동적 바이트 배열: bytes와 string이 있음
 - bytes 키워드는 raw byte data에 사용
 - string 키워드는 UTF-8로 인코딩된 문자열에 사용

```
bytes32[] trades;
```

- 동적 할당 배열에서는 길이를 리턴하는 length 멤버가 있음

```
trades.length;
```

Understanding of Smart Contract

□ Solidity Language

▣ Literal

- 리터럴은 고정된 데이터를 나타내는데 사용

- Integer literal

- 정수 리터럴은 0부터 9까지 범위의 십진수로 된 시퀀스

```
uint8 = 2;
```

- String literal

- 문자열 리터럴은 큰 따옴표나 작은 따옴표와 함께 쓰인 문자 집합

```
'packt' "packt"
```

- Hex literal

- 헥스 리터럴은 키워드 hex를 접두어로 사용하며, 큰 따옴표나 작은 따옴표 안에 지정됨

```
(hex 'AABBCC');
```


Understanding of Smart Contract

□ Solidity Language

▣ enum (열거 타입)

- 열거 타입은 솔리디티에서 사용자 정의 타입을 만드는 방법
- 열거 타입은 모든 정수 타입과 서로 명시적인 타입 변환이 가능함

```
enum Order {Filled, Placed, Expired};  
Order private ord;  
ord = Order.Filled;
```

Understanding of Smart Contract

□ Solidity Language

▣ 전역변수

- 전역 네임스페이스에서 항상 사용할 수 있는 변수
- 블록과 트랜잭션에 관한 정보를 제공함
- 또한, 암호화 함수 및 주소 관련 변수들도 존재

`keccak256(...)` returns (*bytes32*)

keccak256 함수는 함수 인자 값의 케착 256비트 해시 계산

`ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s)` returns (address)

ecrecover 함수는 타원 곡선 서명에서 공개 키의 연관 주소를 리턴

`block.number`

현재 블록 번호를 리턴

Understanding of Smart Contract

□ Solidity Language

▣ Struct (구조체)

- 구조체는 각기 다른 여러 데이터 타입을 논리적인 그룹으로 묶을 때 사용함

```
pragma solidity ^0.4.0;
contract TestStruct {
    struct Trade
    {
        uint traded;
        uint quantity;
        uint price;
        string trader;
    }

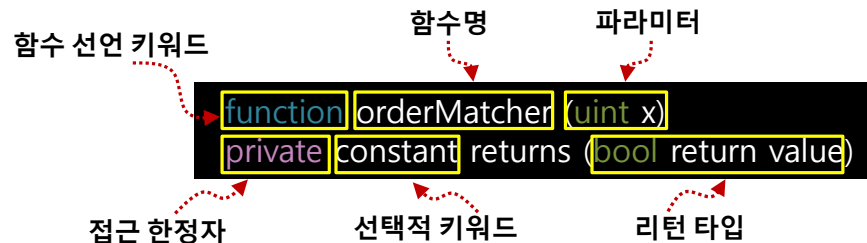
    // 다음과 같이 구조체를 초기화해 사용할 수 있음
    Trade tStruct = Trade({traded:123, quantity:1, price:1, trader:"anyone"});
}
```

➔ Understanding of Smart Contract

❑ Solidity Language

▣ 함수

- 솔리디티의 함수는 컨트랙트와 연관되어 있는 코드 모듈임
- 함수 선언은 이름, 파라미터들, 접근 한정자(access modifier), constant 키워드, 리턴 타입으로 구성되며, 파라미터와 constant 키워드, 리턴 타입은 없어도 무방함



```
function <함수명> (<파라미터>) <접근 한정자>
returns (<리턴 데이터 타입> <변수명>)
{
    <함수 본문>
}
```

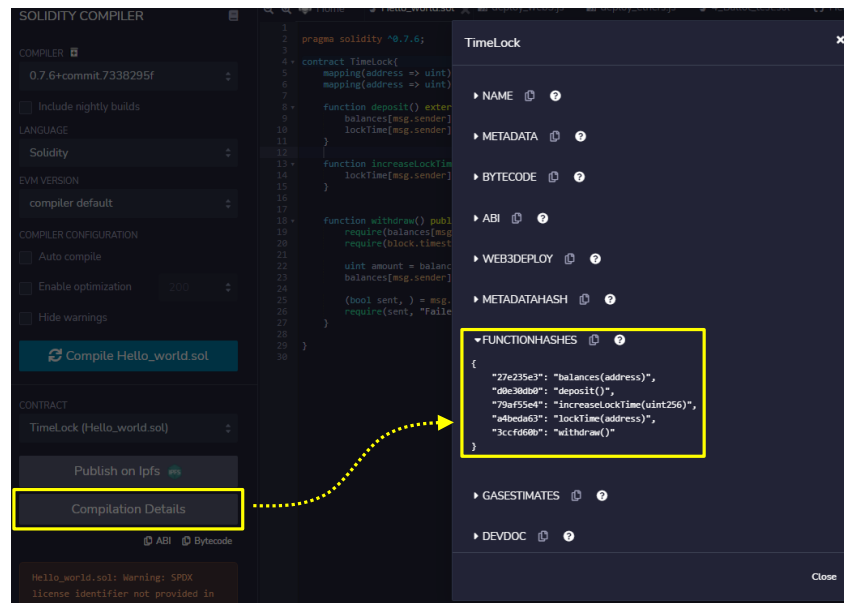
Understanding of Smart Contract

□ Solidity Language

▣ 함수

- 함수 서명

- 솔리디티의 함수는 함수의 서명으로 식별되며, 서명은 전체 시그니처 문자열의 keccak256 비트 해시의 처음 네 바이트를 의미함
- Remix IDE에서 서명을 확인할 수 있음



Understanding of Smart Contract

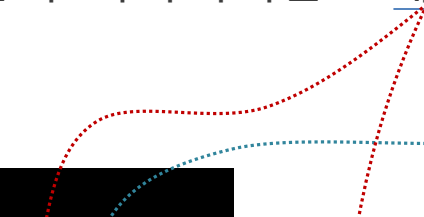
□ Solidity Language

▣ 함수

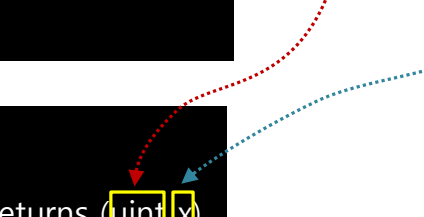
- 함수 입력 및 출력 파라미터

- 함수의 입력 및 출력 파라미터는 <데이터 타입> <파라미터 명> 형식으로 선언

```
contract myContract
{
    function checkValues(uint x, uint y)
    {
    }
}
```



```
contract myContract
{
    function getValue() returns (uint x)
    {
        z = x + y;
    }
}
```



Understanding of Smart Contract

□ Solidity Language

▣ 함수

- 함수에서 여러 개의 값을 리턴할 수 있음
 - 최대 14개까지 리턴할 수 있으며, 데이터 타입이 달라도 됨
- 내부 함수 호출
 - 현재 컨트랙트의 컨텍스트 안에 있는 함수는 내부적으로 직접 호출할 수 있음. 동일한 계약 내에 있는 함수를 호출하게 되며, EVM 바이트코드 수준에서 간단한 JUMP 호출이 발생함
- 외부 함수 호출
 - 외부 함수 호출을 수행하면 계약에서 다른 계약으로 메시지 호출이 이뤄지게 되고, 모든 함수 파라미터가 메모리에 복사됨
 - 내부 함수를 `this` 키워드 사용해 호출하는 것도 외부 호출로 간주
 - `this` 변수는 현재 계약을 참조하는 포인터로 볼 수 있음
 - 명시적으로 주소 변환 가능하며 컨트랙트의 모든 멤버가 주소에서 상속됨

Understanding of Smart Contract

□ Solidity Language

▣ 함수

– 폴백함수(fallback function)

- 컨트랙트 함수이지만 이름이 없고 인자와 리턴 데이터가 없는 함수
- 폴백함수는 이더를 받을 때마다 실행
 - 이더를 수령하는 컨트랙트 경우, 컨트랙트에서 구현 필수
 - 그렇지 않으면, 예외 발생과 수령한 이더가 리턴됨
- 컨트랙트에서 일치하는 함수 서명이 없는 경우에도 실행됨
- 계약에서 이더를 받을 것으로 예상되는 경우 폴백함수를 선언할 때 한정자를 payable로 지정해야 하며, 지정하지 않을 경우 폴백함수에서 이더를 받지 못함

```
function ()  
{  
    throw;  
}
```

* address.call() 메소드를 사용해 호출 할 수 있음

* 폴백함수가 호출되면 throw가 실행되고, 상태는 호출하기 전으로 되돌아감

Understanding of Smart Contract

□ Solidity Language

▣ 함수

- 함수 가시성

- 함수를 정의할 때 접근 지정자를 설정할 수 있음
- external
 - 다른 컨트랙트 및 트랜잭션에서 함수 접근이 가능
- public
 - 기본 값이며, 함수를 공용으로 만들어 내부적으로 호출하거나 메시지를 사용해 호출 할 수 있음
- internal
 - 부모 컨트랙트에서 파생된 컨트랙트에서만 볼 수 있는 내부 함수로 생성
- private
 - 선언한 해당 컨트랙트에서만 접근 가능한 전용 함수 지정

Understanding of Smart Contract

□ Solidity Language

▣ Data location (데이터 위치)

- 데이터 위치는 특정 복합 데이터 타입이 저장될 위치를 지정
 - 기본 값 또는 지정된 annotation에 따라 데이터 위치가 storage 또는 memory로 정해짐
 - 배열과 구조체에 적용할 수 있으며 storage 또는 memory 키워드를 사용해 지정함
- 메모리와 저장소 간의 복사 비용이 매우 클 수 있기 때문에, 위치를 지정하게 되면 가스 소비를 통제하는 데 도움이 됨
- calldata는 함수 인자를 저장하는데 사용 (즉, **외부 함수의 파라미터들은 calldata** 를 사용함)
- 기본적으로 **함수 파라미터는 memory**에 저장되고, 그 외의 모든 **로컬 변수는 storage**를 사용함

Understanding of Smart Contract

□ Solidity Language

▣ mapping (매핑)

- 매핑은 키를 값으로 사상하는 데 사용
- 키와 값을 연관시키는 방법
- 맵 안의 모든 값은 모두 0으로 먼저 초기화 됨

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

offers라는 매핑을 선언

Key type

Value type

```
mapping (string => uint) bids;  
bids["packt"] = 10;
```

문자열 값을 정수 값으로 매핑하는
딕셔너리 / 해시 테이블로, bids라는 매핑에는
packt라는 문자열이 들어있으며 10이라는 값으로 매핑

Understanding of Smart Contract

□ Solidity Language

▣ 제어구조

- if ... elsem, do, while, for, break, continue, return
- C언어나 자바스크립트 같은 언어와 동일하게 작동함

```
if (x == 0)
  y = 0;
else
  z = 1;
```

만약 x가 0과 같으면 y에 0을 넣고, 그렇지 않으면 z에 1을 할당

```
do {
  x++;
} (while z>1);
```

z가 1보다 큰 동안 x를 증가

```
while (x > 0) {
  z++;
}
```

x가 0보다 큰 동안 z를 증가

Understanding of Smart Contract

□ Solidity Language

▣ 제어구조

- if ... elsem, do, while, for, break, continue, return
- C언어나 자바스크립트 같은 언어와 동일하게 작동함

```
for (uint8 x=0; x<=10; x++)  
{  
    // 무언가 작업 수행  
    z++;  
    if(z == 5) break;  
}
```

```
return 0
```

for, break, continue: x가 10보다 작거나
같은 때까지 무언가 작업을 수행하고,
for 루프는 10회 실행되며, z가 5이면
for 루프를 탈출함

실행을 중지하고 0이라는 값을 리턴

Understanding of Smart Contract

□ Solidity Language

▣ 이벤트

- 솔리디티의 이벤트 타입은 EVM 로그의 특정 이벤트를 저장하는데 사용
 - 외부 인터페이스에 계약의 변경이나 이벤트를 통지해야 할 때 유용하며, 이러한 로그는 블록체인상에 트랜잭션 로그로 저장
 - 컨트랙트에서는 로그에 접근하지 못하며, 로그는 컨트랙트의 상태 변경 또는 이벤트 발생(조건 충족)을 알림

```
pragma solidity ^0.4.0;
contract valueChecker
{
    uint8 price=10;
    event valueEvent(bool returnValue);
    function Matcher(uint8 x) public returns (bool) {
        if (x>=price) {
            valueEvent(true);
            return true;
        }
    }
}
```

Matcher 함수에 전달된
파라미터 x가 10보다
크거나 같은 경우
valueEvent 이벤트가
true를 리턴함

Understanding of Smart Contract

□ Solidity Language

▣ 상속

- is 키워드를 사용해 특정 컨트랙트에서 다른 컨트랙트를 파생할 수 있음

```
pragma solidity ^0.4.0;
contract valueChecker
{
    uint8 price=20;
    event valueEvent(bool returnValue);
    function Matcher(uint8 x) public returns (bool) {
        if (x>=price) {
            valueEvent(true);
            return true;
        }
    }
}
contract valueChecker2 is valueChecker
{
    function Matcher2( ) public view returns (uint)
    {
        return price + 10;
    }
}
```

valueChecker2가 valueChecker
계약에서 파생되며, 파생된
컨트랙트는 부모 컨트랙트의 전용
멤버(private member)를 제외한
모든 멤버에게 접근이 가능함
uint8 price = 20을 **uint8 private
price = 20**으로 변경하면,
valueChecker2계약에서 접근하지
못함

Understanding of Smart Contract

□ Solidity Language

▣ 라이브러리

- 특정 주소에 한번만 배치되며 해당 코드는 EVM의 CALLCODE 또는 DELEGATECALL 연산자 코드를 통해 호출됨
- 라이브러리의 핵심 아이디어는 **코드 재사용**
- 라이브러리는 컨트랙트와 비슷하며 호출 컨트랙트에 대한 기본 컨트랙트 역할을 수행함

```
library Addition
{
    function Add(uint x, uint y) returns (uint z){
        return x + y;
    }
}
```

```
import "Additions.sol"
function Addtwovalues() returns(uint)
{
    return Addition.Add(100, 100);
}
```

* 라이브러리는 상태 변수를 가질 수 없으며, 상속하거나 상속될 수 없음

* 이더를 받을 수 있는
컨트랙트와는 대조적으로, 이더를 받지 못함

Understanding of Smart Contract

□ Solidity Language

▣ 버전 지시자

- 호환 가능한 컴파일러 버전을 지정

```
pragma solidity ^0.5.0;
```

▣ 임포트

- 미리 작성해놓은 '.sol' 파일을 import하여 그 기능을 사용할 수 있음(여러 파일에 나누어 개발할 때 용이)

CONTENTS

- ❑ Smart Contract Practice 1

- ❑ Let's do Lesson 1

- ❑ <https://cryptozombies.io/en/lesson/1/chapter/1>

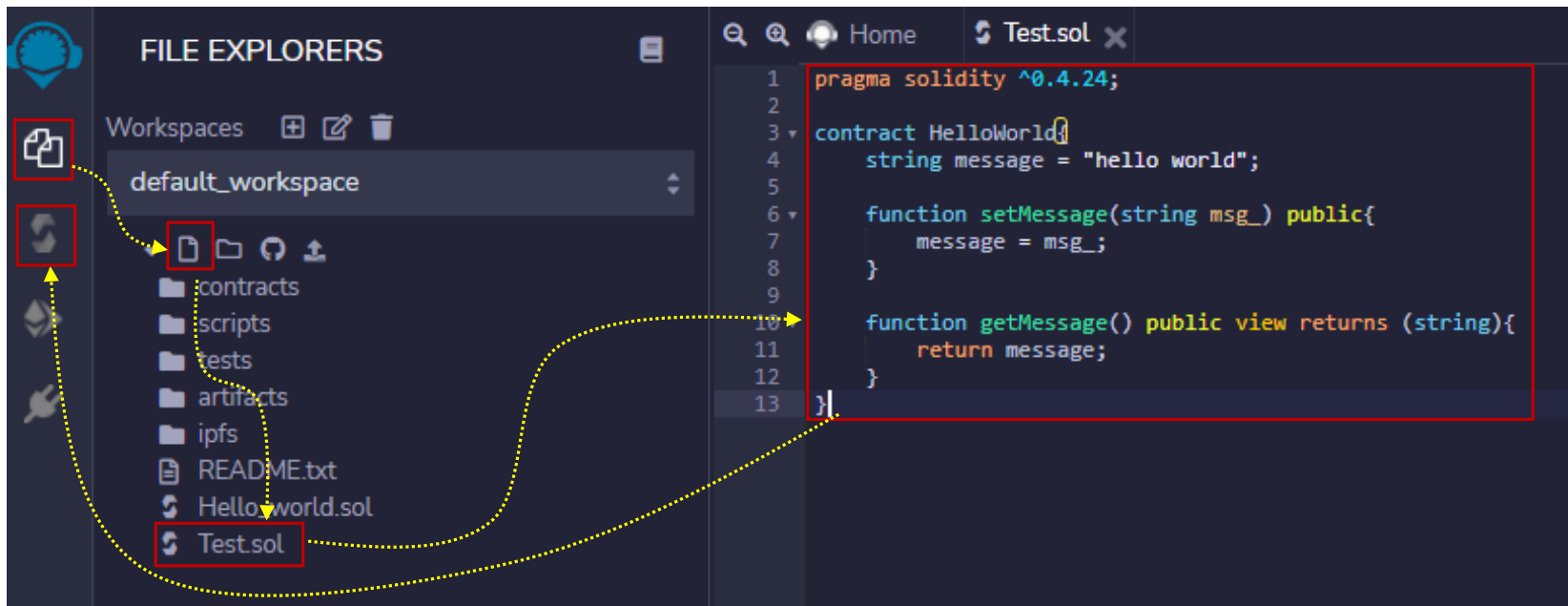
- ❑ Chapters 1-13

- ❑ Let's do Lesson 5

- ❑ <https://cryptozombies.io/ko/lesson/5/chapter/1>
 - ❑ Chapters 1-3,
 - ❑ Chapters 5-8,
 - ❑ Chapters 10

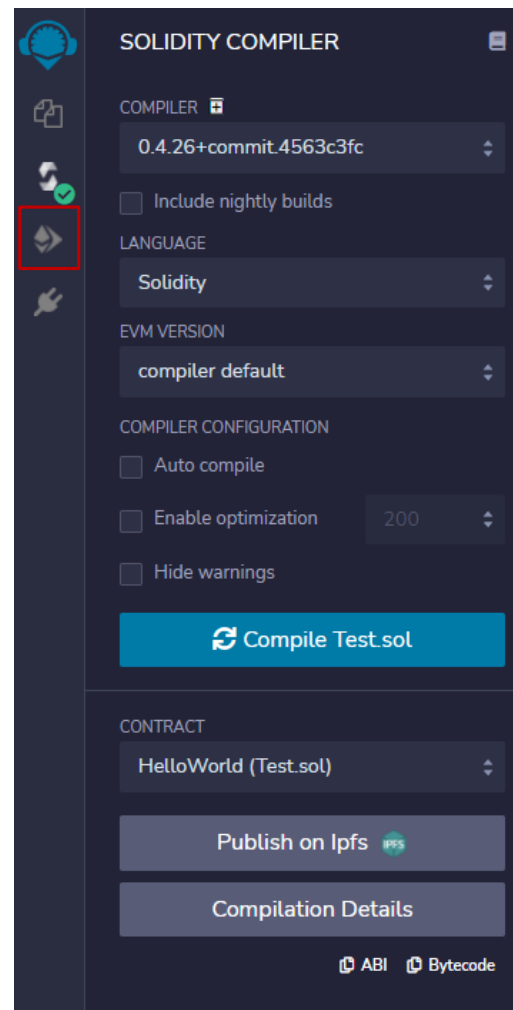
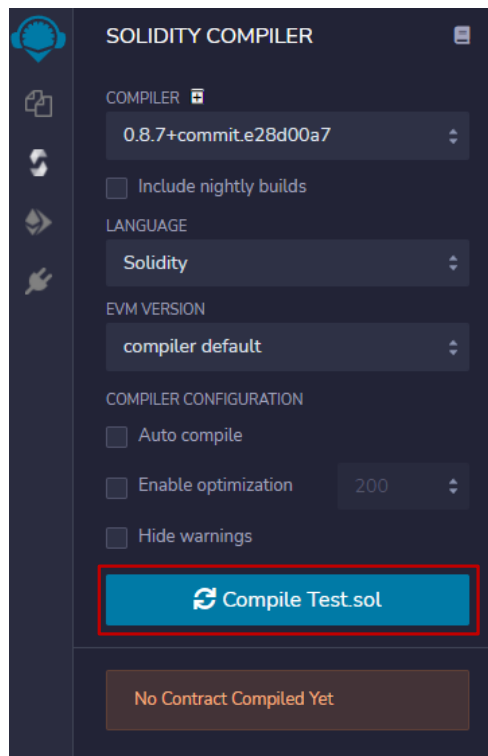
Smart Contract Practice

□ Remix



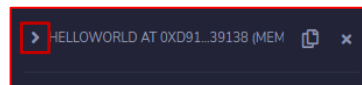
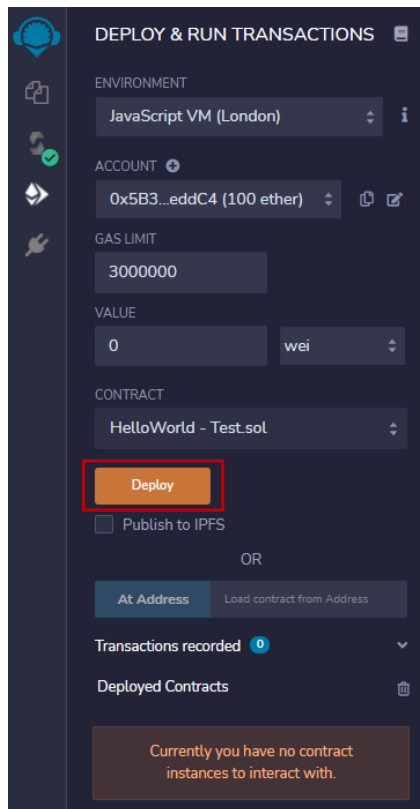
Smart Contract Practice

Remix

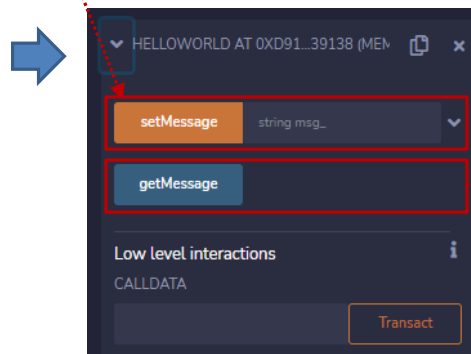


Smart Contract Practice

Remix



```
1 pragma solidity ^0.4.24;
2
3 contract HelloWorld {
4     string message = "hello world";
5
6     function setMessage(string msg_) public {
7         message = msg_;
8     }
9
10    function getMessage() public view returns (string) {
11        return message;
12    }
13 }
```



References

- ❑ Lecture slides from BLOCKCHAIN @ BERKELEY
- ❑ <https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/>
- ❑ "Mastering Ethereum - Building Smart Contracts and Dapps"
- ❑ <https://slides.com/ironpark/parity-smart-contract#/5>

Q & A

