

1. 개발 도구 리믹스

확장자가 .sol인 파일이 솔리디티 언어로 만들어진 스마트컨트랙트이다.
솔리디티언어는 자바스크립트와 비슷하다.

먼저 개발도구 리믹스에 대해서 보자.

<https://remix.ethereum.org/>

브라우저의 특정 영역에 저장되므로 브라우저 캐시를 지우지 않는 이상 남아있다.
주기적으로 변경을 하는 경우에는 다운로드하여 보관하는 것이 안전하다.

첫 예제

MyFirstContract.sol

```
pragma solidity >=0.4.0 <0.6.0 ;
```

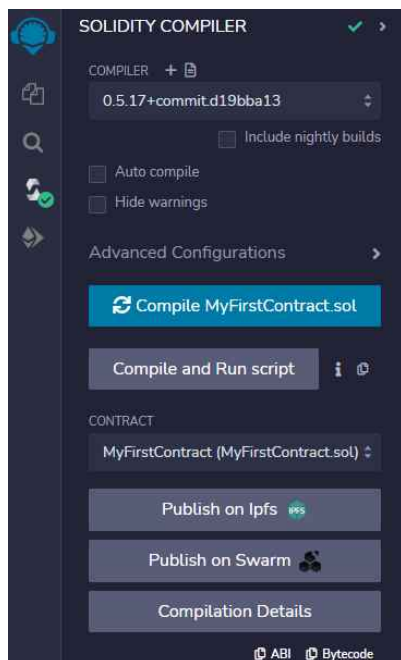
```
contract MyFirstContract{  
    function sayHello() public pure returns(string memory){  
        return 'Hello World';  
    }  
}
```

pragma는 솔리디티 버전을 의미한다.

>=0.4.0 <0.6.0은 0.4이상 0.6미만의 버전을 의미한다.

pure는 가스를 소모하지 않는다.

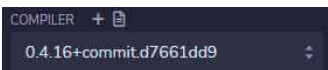
전역변수가 없고 전역변수에 접근해서 업데이트를 하지 않기 때문에 pure라고 적는다.



컴파일러



배포



솔리디티 버전
0.5.17로 변경



MyFirstContract.sol
컴파일

 Compile MyFirstContract.sol

을 클릭하여 컴파일을 한다.

Compilation Details

ABI Bytecode

왼쪽 메뉴의 하단에 있는 Bytecode를 복사해서 출력해보면 Bytecode로 컴파일된 것을 확인할 수 있다. EVM에서 솔리디티가 실행할 수 있는 형태로 변환된다. 자바에서 컴파일된 class파일과 같다.

Compilation Details

를 클릭하면 더 자세한 내용을 알 수 있다.

MyFirstContract

NAME

METADATA

▼ BYTECODE

```
{
  "linkReferences": {},
  "object": "608060405234801561001057600080fd5b5061011d80610020600039f
  "opcodes": "PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH
  "sourceMap": "36:115:0:-;8:9:-1;5:2;;30:1;27;20:12;5:2;36:115:0
```

ABI

STORAGELAYOUT

WEB3DEPLOY

METADATAHASH

FUNCTIONHASHES

GASESTIMATES

Close

ABI를 확인 해보자.


```
[
  {
    "constant": true,
    "inputs": [],
    "name": "sayHello",
    "outputs": [
      {
```

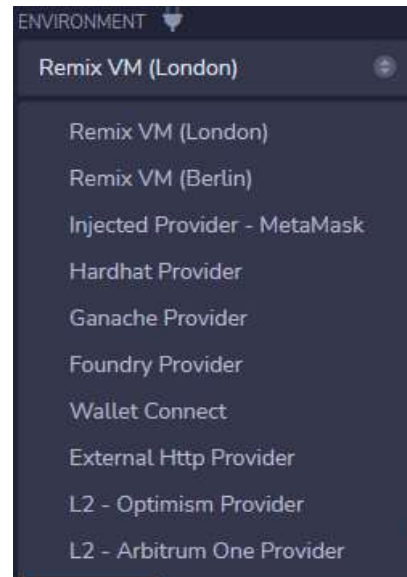
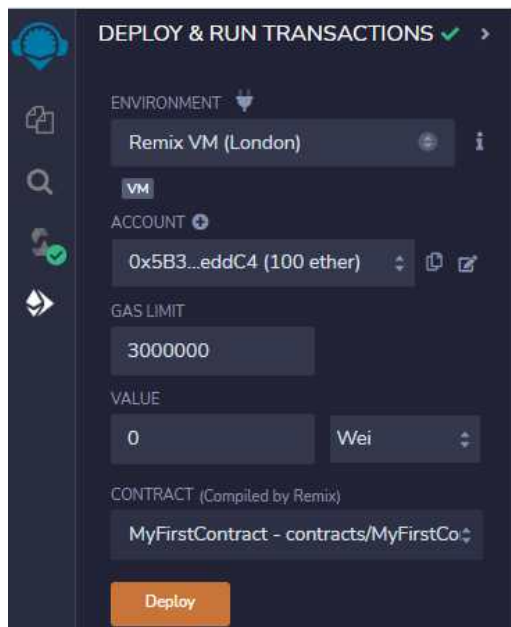
```

        "internalType": "string",
        "name": "",
        "type": "string"
    }
},
"payable": false,
"stateMutability": "view",
"type": "function"
}
]

```

MyFirstContract.sol의 인터페이스 명세가 나온다. 자바스크립트에서 스마트 컨트랙트에 접근하기 위한 인터페이스명세로 쓰이게 된다.

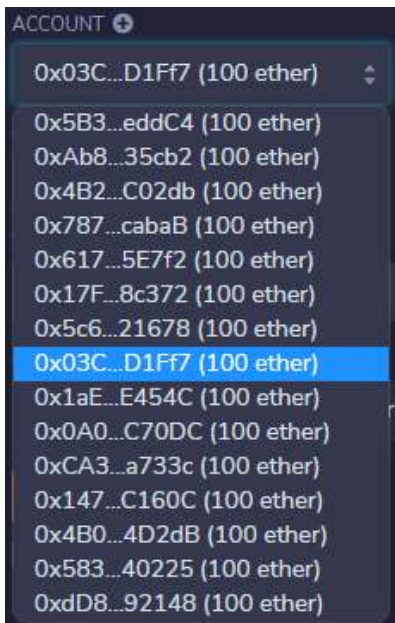
 배포를 해보자.



어디로 배포할 것인지 정해야 하는데 기본값 사용



무엇을 배포하는지 정의되어 있다.



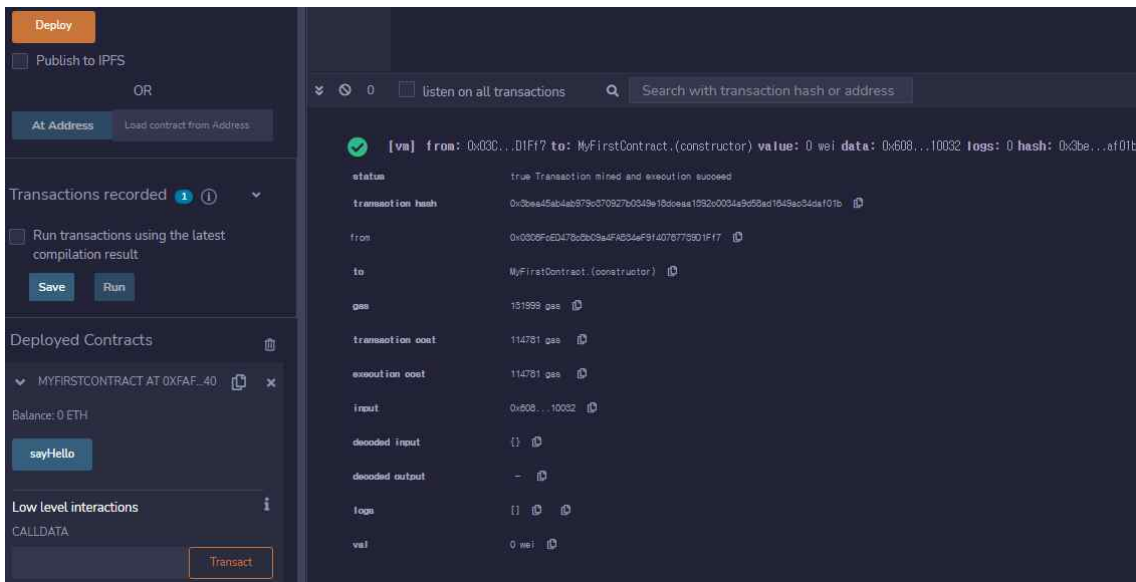
지갑 주소를 기본적으로 여러개 생성해 준다.

지갑 주소는 두가지 종류가 있는데 하나는 결제용 또 하나는 스마트컨트랙트 배포용이다.

스마트먼트랙트배포용으로 선택한다.

Deploy

를 실행시키면 EVM에 배포가 되고 오른쪽하단에 배포된 로그가 출력이 된다.



transaction hash : 트랜잭션마다 해쉬값이 있고 배포하게 되면 어느 블록체인에든 포함되게 된다. (0x3bea45ab4ab979c370927b0349e18dceaa1392c0034a9d58ad1649ac34daf01b)

from : 어디로부터 배포가 되었는지 알 수 있다.

0x03C6FcED478cBbC9a4FAB34eF9f40767739D1Ff7

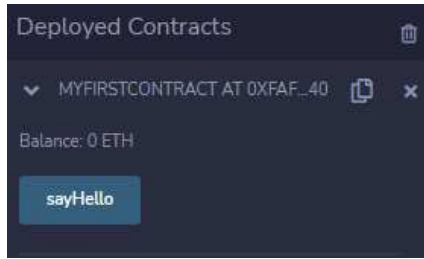
gas : 트랜잭션 발생할 때마다 소모되는 가스 (131999 gas)

Deployed Contracts 밑에서 어디로 배포했는지 알 수 있다,

MYFIRSTCONTRACT AT 0xfAF646893C6D3Ef849FadD67FC1Ca3e347f409B7

즉 0xfAF646893C6D3Ef849FadD67FC1Ca3e347f409B7주소로 배포했다.

배포한 내용



sayHello함수가 배포되었음을 알 수 있다.

sayHello버튼을 클릭하면 오른쪽에 로그가 생기고 버튼 밑에 Hello World가 리턴 된 것을 알 수 있다.



2. 가나슈 및 솔리디티 기본문법

가나슈는 테스트 블록체인이고 로컬환경에서 사용할 수 있다.

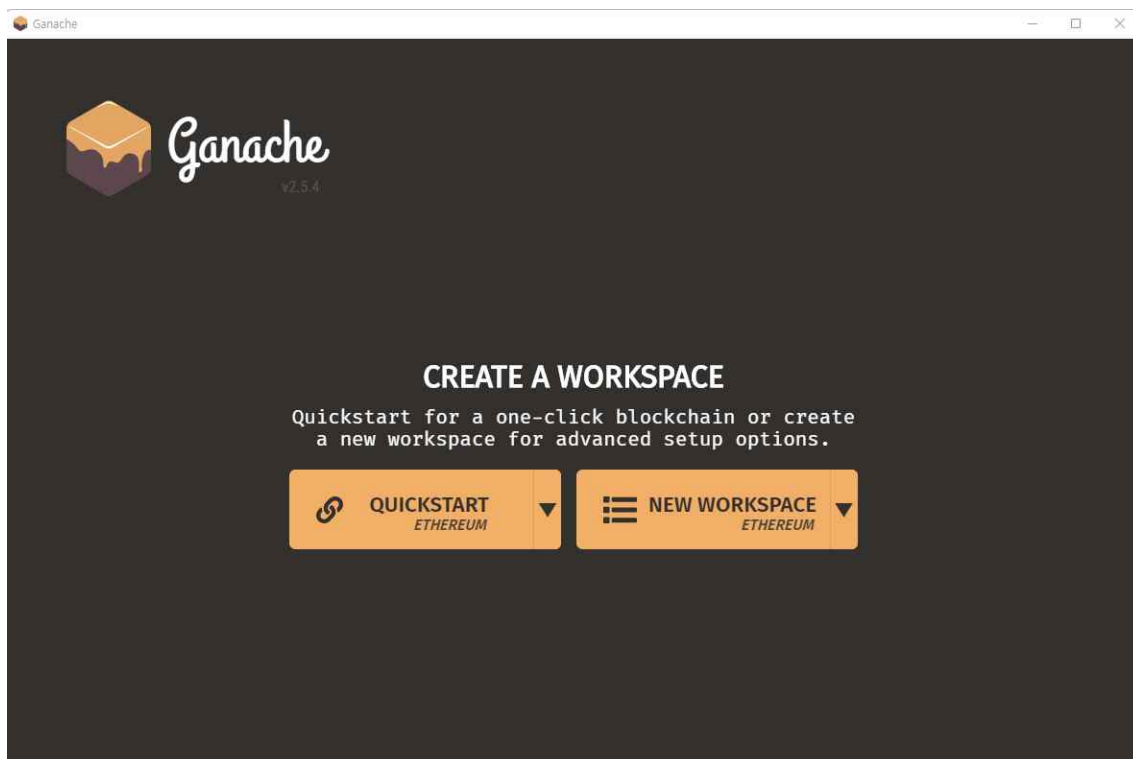
1) 가나슈(Ganache)테스트 블록체인 설치

이더리움 테스트 블록체인을 로컬 컴퓨터에서 간편하게 구동 시켜주는 어플리케이션

<https://trufflesuite.com/ganache/>



가나슈를 실행하면 다음과 같은 화면이 뜬다.



QUICKSTART를 클릭한다.

ACCOUNTS

mnemonic: glad inmate ivory enact sudden cat remain version marriage divert useful cream

HD PATH: m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0xECC31f72Bbdcf7c7ACf6236b1369009b3E9e6323	100.00 ETH	0	0
0xAba488c26201E6f588493b16972656Ea131Ea75	100.00 ETH	0	1
0x145f4295d343A512F250C872Efc32b00021c7F4	100.00 ETH	0	2
0xb2D00976Fa1BCAd81eb2385515Eb9861c15c00B4	100.00 ETH	0	3
0x43CCAb03d3B45C1fC3555cE8Db481e7228959a26	100.00 ETH	0	4
0x6EE44D3355Cc358C0adD684125a3492eFb5b24Ed	100.00 ETH	0	5
0x03Df866d9AED4649D913Cd76634753eC93fafB82	100.00 ETH	0	6

지갑 주소가 10개 정도 있는 것을 확인할 수 있다.
 각 주소마다 가상의 화폐가 100ETH씩 충전되어 있다.

위 메뉴에서 BLOCKS를 선택해보자

BLOCKS

← BACK

BLOCK 0

GAS USED	GAS LIMIT	MINED ON	BLOCK HASH
0	6721975	2022-12-13 13:26:47	0x57662eac032aaeebc545dcbc02c976c3d4fe6986b73e67a12f527882dcfb8ac

0번 블록이 생성되어 있는 것을 알 수 있다.

TRANSACTIONS를 열어보면 아무것도 없다.

TRANSACTIONS

The 'Ganache' application shows the following accounts:

ADDRESS	BALANCE
0xECC31f72Bbdcf7c7ACf6236b1369009b3E9e6323	100.00 ETH
0xEAba488c26201E6f588493b16972656Ea131Ea75	100.00 ETH
0x145f4295d343A512F250C872Efc32b00021c7F4	100.00 ETH
0xb2D00976Fa1BCAd81eb2385515Eb9861c15c00B4	100.00 ETH
0x43CCAb03d3B45C1fC3555cE8Db481e7228959a26	100.00 ETH
0x6EE44D3355Cc358C0adD684125a3492eFb5b24Ed	100.00 ETH
0x03Df866d9AED4649D913Cd76634753eC93fafB82	100.00 ETH

리믹스와 가나슈가 서로 연동되지 않아서 리믹스에 있는 주소와 가나슈에 있는 주소가 서로 다르다.

이제 리믹스가 가나슈가 가지고 있는 블록체인으로 연동되게 해보자.
리믹스의 배포에서 가나슈 프로바이더를 선택한다.

The 'Ganache Provider' dialog box contains the following information:

Note: To run Ganache on your system, run:

```
yarn global add ganache
ganache
```

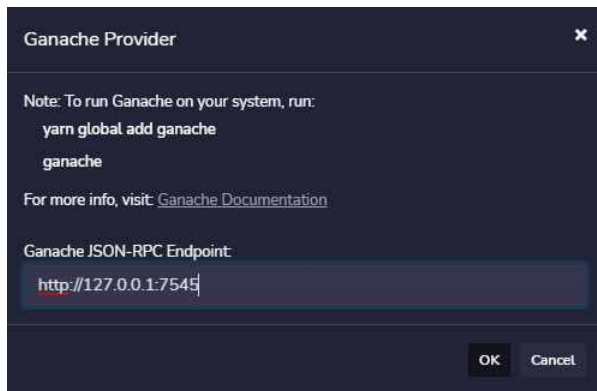
For more info, visit: [Ganache Documentation](#)

Ganache JSON-RPC Endpoint:

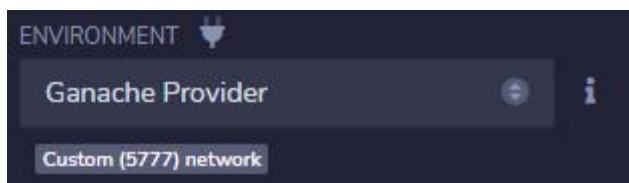
```
http://127.0.0.1:8545
```

Buttons: OK, Cancel

기본 포트가 8545인데 가나슈 포트인 7545로 변경하자.



OK눌러서 연결한다,



EVM이 Custom으로 변경되었다.

ACCOUNT

- 0xECC...e6323 (100 ether)
- 0xECC...e6323 (100 ether)**
- 0xEAb...1Ea75 (100 ether)
- 0x145...1c7F4 (100 ether)
- 0xb2D...c00B4 (100 ether)
- 0x43C...59a26 (100 ether)
- 0x6EE...b24Ed (100 ether)
- 0x03D...afB82 (100 ether)
- 0xcd6...4133E (100 ether)
- 0x679...2f057 (100 ether)
- 0x172...37bF1 (100 ether)

Deploy

☐ Publish to IPFS

OR

At Address

Transactions recorded 0

☐ Run transactions using the latest compilation result

Save Run

MNEMONIC

glad inmate ivory enact sudden cat remain version marriage divert useful cream

ADDRESS	BALANCE
0xECC31f72Bbdcf7c7ACf6236b1369009b3E9e6323	100.00 ETH
0xEAb488c26201E6f588493b16972656Ea131Ea75	100.00 ETH
0x145f4295d343A512F250C872Efc32b00021c7F4	100.00 ETH
0xb2D00976Fa1BCAd81eb2385515Eb9861c15c00B4	100.00 ETH
0x43CCAb03d3B45C1fC3555cE8Db481e7228959a26	100.00 ETH
0x6EE44D3355Cc358C0adD684125a3492eFb5b24Ed	100.00 ETH

가나슈 계정과 연동된 것을 알 수 있다.

리믹스에서 가나슈로 배포하고 가나슈 TRANSACTIONS을 열어보자.

TRANSACTIONS

TX HASH: 0x14c19a319629b9570fdb079ae5ca2f9dbf2d7c2b74baab92e784880d65b07e30

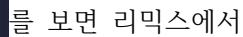
FROM ADDRESS: 0xECC31f72Bbdcf7c7ACf6236b1369009b3E9e6323

CREATED CONTRACT ADDRESS: 0x0BA9c4c692bE7fd37118f9CCAb2Df471c2a2b337

GAS USED: 114781

VALUE: 0

배포했더니 가나슈에 트랜잭션이 들어왔다.

[illegible]

위 가냐슈에서도 배포된 곳이 같다는 것을 알수 있다.

ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK

GAS PRICE

GAS LIMIT

HARDFORK

NETWORK ID

RPC SERVER

MINING STATUS

WORKSPACE

QUICKSTART

SAVE

SWITCH

1

20000000000

6721975

MUIRGLACIER

5777

HTTP://127.0.0.1:7545

AUTOMINING

← BACK

BLOCK 1

GAS USED

GAS LIMIT

MINED ON

BLOCK HASH

114781

6721975

2022-12-13 13:50:45

0x1285ab5934789ddf36e73945bd9fc6291e696c2ee4cce93aa5308f5a07b3c71b

TX HASH

0x14c19a319629b9570fdb079ae5ca2f9dbf2d7c2b74baab92e78488065b07e30

CONTRACT CREATION

FROM ADDRESS

CREATED CONTRACT ADDRESS

GAS USED

VALUE

0xECC31f72Bdcf7c7ACf6236b1369089b3E9e6323

0x08A9c4c692be7fd37118f9CCAb2Df471c2a2b337

114781

0

TX DATA가 트랜잭션 해시이고 BLOCK HASH에 포함되게 된다.

나)스마트 컨트랙트 기본 문법

프래그마(pragma)

- 프래그마는 솔리디티 파일 첫 줄에 작성한다. 컴파일러의 버전을 지정하는 명령이다.
- 솔리디티는 지금도 계속 업데이트 중이며 지속적으로 발전하고 있다. 버전 업데이트가 되면서 API가 생성, 변경, 삭제되는 경우가 종종 발생하므로 버전정보 표기는 호환성 측면 중요한 역할을 한다.
- 구문은 아래와 같이 작성한다. pragma와 solidity 예약어는 소문자로 작성하며, 버전정보는 major, minor, patch숫자로 작성하고 마지막에는 ;으로 종료한다.
- 구문:
pragma solidity major.minor.patch
- pragma solidity ^0.4.19;
- pragma solidity >=0.4.0 <0.6.0

표현	설명
^0.4.19	>=0.4.19 <0.5.0의미
<0.4.19	0.4.19미만과 호환
<=0.4.19	0.4.19이하와 호환
>0.4.19	0.4.19초과와 호환
>=0.4.19	0.4.19이상과 호환
=0.4.19	0.4.19만 호환
0.4.19	=0.4.19와 같은 뜻
0.4.19 0.4.20	0.4.19 또는 0.4.20
>=0.4.19 <0.5.0	0.4.19보다 크거나 같고 0.5.0미만

상태변수(State Variables)

- 상태변수는 컨트랙트 내부, 함수 밖에서 선언된 변수를 말한다.
- 상태변수는 이더리움 영구 저장소에 저장된다.(블록체인에 저장되는 변수, 가스가 소모된다. 즉 ETH가 필요하다. 변수값을 변경할 때도 ETH가 필요 저장할 때도 ETH가 필요)
- 솔리디티 프로그램을 이해하기 위해서는 상태변수 개념과 특성을 파악하는 것이 중요하다. 자바에서 멤버변수와 같은 개념이다. 변수는 값을 가지고 있다. 변수는 값이 저장된 저장소에 위치에 주어진 이름이다. 변수를 사용하기 위해서는 반드시 선언하여야 한다. 상태변수는 실행 중에 값이 변경 될 수 있다.
- 데이터타입 [internal | private | public] 변수명;
- string public stringVar = "문자열"
- byte32 public byteVar = "바이트 문자열"
- bool private boolVar = true;
- uint number = 124;

가시성	설명
internal	변수가 선언된 컨트랙트에서만 사용 가능. 상속된 컨트랙트에서 사용 가능
private	내부 컨트랙트에서만 호출이 가능 internal과 비슷하지만 상속된 컨트랙트에서는 사용할 수 없음
public	내 외부 모든 컨트랙트에서 사용가능 set, get함수가 자동으로 생성

데이터타입

value Type :

address : 주소표현 : 20byte : 지갑주소나 contract주소를 저장,

bool :

int/uint

byte

enum

Reference Type : array,

mapping(자바의 맵하고 같다. Map<String,int>) : mapping(키 => 값)

string

struct

bool

● 예제

pragma solidity >=0.4.9;

```
contract BoolExam{
```

```
    function getBoolean() public pure returns(bool){
```

```
        bool boolVar = false;
```

```
        return boolVar;
```

```
    }
```

```
}
```

bool타입의 변수를 리턴하는 간단한 프로그램이다. 컨트랙트의 기본 구조는 이전에 살펴본 것과 같이 pragma를 선언하는 것으로 시작한다.

(1라인) pragma정보를 선언한다.

(3라인) BooleanExam컨트랙트를 선언한다.

(4라인) getBoolean 함수를 선언한다.

- pure:메모리 저장소만 사용, 상태변수 값을 변경하지 않음. 수수료 없음

- public : 컨트랙트 내부 어디서든 사용가능

- returns : 함수의 결과 변환값이 bool형식으로 1개 있음

(5라인) bool 타입의 지역 변수 리턴

```
function getBooleanFnc() public pure returns(bool _result){
```

```
    bool boolVar = false;
```

```
    _result = boolVar;
```

```
}
```

```
function getMultiReturnsBool() public pure returns(bool, bool, bool, bool){
```

```
    bool care1 = true || false;
```

```
    bool care2 = false || true;
```

```
    bool care3 = true || true;
```

```
    bool care4 = false || false;
```

```

        return (care1,care2,care3,care4);
    }
    returns(bool, bool, bool, bool)문에 4개를 리턴하게 되어 있으면 리턴 값도 4개이어야 한다.
    return (care1,care2,care3,care4);

```

정수

```

function getResult() public pure returns (int _result1, int _result2){
    int a = 3;
    int b = 2;

    _result1 = 3/2*10;
    _result2 = a/b*10;
    // return (_result1, _result2);
}

```

Solidity의 데이터 저장 공간

- * Storage

대부분의 변수, 함수들이 저장되는 영역, Persistent, 가스비가 비싸다.

- * Memory

함수 범위내에서만 유효(영속적이지 않음) 가스비가 싸다.

- * Calldata

special data location that contains function arguments

- * Stack

1024MB 제한

예제) <https://solidity-by-example.org/data-locations/>

```

uint[] public arr;
mapping(uint => address) map;
struct MyStruct {
    uint foo;
}
mapping(uint => MyStruct) myStructs;

```

Storage에 저장됨, 즉 블록체인에 저장

// You can return memory variables

```

function g(uint[] memory _arr) public returns (uint[] memory) {
    // do something with memory array
}

```

```
}
```

Memory에 저장됨

```
function h(uint[] calldata _arr) external {  
    // do something with calldata array  
}
```

calldata에 저장됨

배열

1) 구문

데이터 타입 [size][memory| storage][public|private] 이름;

2)예제

```
uint[10] fixedArray;          .// 10개의 공간이 있는 정적배열  
uint[] dynamicArray;         // 크기가 정해지지 않은 동적배열  
delete fixedArray;           // 배열의 초기화  
fixedArray.length;           // 배열의 길이  
fixedArray.push(1);           // 배열 추가 ,종적 배열에 요소를 추가할 때 사용  
fixedArray[3];               // 배열 속성값 조회  
fixedArray[3] = 3            // 배열 속성값 추가  
uint[] memory myArray = new uint[](5) // 배열 선언
```

<https://solidity-by-example.org/array/> 참조

```
uint [] number;  
uint [] year = [uint(1992), 1994, 1997];  
uint [] age = new uint[](5);  
  
function pushItem(uint _item) public returns(uint){  
    uint idx = year.push(_item);  
    return idx;  
}  
function getItem(uint _idx) public view returns(uint){  
    return year[_idx];  
}  
function testArray(uint _len) public view returns(uint[] memory){  
    uint[] memory array = new uint[](_len);  
    for(uint i = 0; i < _len ; i++){  
        array[i] = i;  
    }  
    return array;  
}
```

열거형

구문 : enum 열거형명 {변수명1, 변수명2, ..., 변수명n}

예제 :

```
pragma solidity >=0.4.0 <0.6.0 ;
```

```
contract EnumExam{
    enum status{start, stop, running, pending}
    status myStatus = status.stop;

    function getStatus() public view returns(status){
        status s = myStatus;
        s = status.start;
        return s;
    }
}
```

참고 사이트

<https://github.com/zacscoding/blockchain/tree/master/ethereum/smartcontract>

Gas

- 이더리움 플랫폼에서 Transaction을 실행하기 위한 네트워크 수수료의 단위
- 코드에 따라 다르게 측정(복잡도가 낮을수록 수수료가 낮다)
- EVM네트워크의 낭비를 최소화하기 위한 목적
- 채굴자들에게 보상
- 예> 트랜잭션 발생 : 21000gas

수수료

- Gas * Gas Price
- 비유> 휘발유 30Liters * 1,790원/Liter = 53,700원
- 1트랜잭션에 21000gas가 필요하고, gas price시세를 확인해 보니

Recommended priority fee in Gwei

<https://ethgasstation.info/>

2 FAST < 2m \$0.07 / Transfer Gas Price (legacy): 53	2 STANDARD < 5m \$0.07 / Transfer Gas Price (legacy): 14.3	2 SAFE LOW < 30m \$0.07 / Transfer Gas Price (legacy): 14.3
--	--	---

- Standard에 14.3GWei
- 따라서, 수수료 = 21000 * 14.3GWei
$$= 21000 * 14.3 * 10^{-9} \text{Ether} = 21000 * 30 * 10^{-9} * 1624\$ = 1.02312\$$$

2022.08.23.에 1ETH = 1624\$

수수료 예시

<https://etherscan.io/tx/0xd116b817a904798779d5e12898ea38ac7f1f6ff0002b0d1718f363b1c0aaaeb2>

거래되는 모든 트랜잭션을 살펴 볼 수 있는 사이트

② From:	0x9962c6fc02957d5da6ea27f4cf966d52f5b7436d
② To:	Contract 0x7f268357a8c2552623316e2562d90e642bb538e5 (OpenSea: Wyvern Exchange v2)
② Value:	0 Ether (\$0.00)
② Transaction Fee:	0.003159811653341322 Ether (\$4.19)
② Gas Price:	0.000000038535228339 Ether (38.535228339 Gwei)
② Ether Price:	\$3,146.36 / ETH
② Gas Limit & Usage by Txn:	81,998 81,998 (100%)
② Gas Fees:	Base: 37.035228339 Gwei Max: 40.125611855 Gwei Max Priority: 1.5 Gwei
② Burnt & Txn Savings Fees:	Burnt: 0.003036814653341322 Ether (\$4.02) Txn Savings: 0.000130408267544968 Ether (\$0.17)

From : 보내는 지갑의 주소

To : 받는 지갑의 주소이지만 contract 주소가 적혀 있음.

Value : 송금 금액 , 0 Ether는 송금은 하지 않고 트랜잭션만 발생

Transaction Fee : 수수료

Gas Limit & Usage by Txn : 가스사용량 81,998

$81998 * 38.535\text{Gwei} = 4.19\$$ 트랜잭션 수수료

Gas Price = Base: 37.035228339 Gwei + Max Priority: 1.5 Gwei

Base: 블록을 생성하는데 사용되는 가스

Max Priority: 채굴자가 가져가는 가스, 송신자가 정할 수 있는 양

Max: 트랜잭션 수수료의 최대값을 지정한 것, 송신자가 정할 수 있는 양

Txn Savings: 0.000130408267544968 Ether (\$0.17) 환불되는 금액

Web3.js를 활용한 차량 웹페이지 개발

- web3.js(Ethereum Compatible JavaScript API)는 네트워크상에 있는 EVM과 통신하기 위한 라이브러리이다.

HTTP 및 IPC를 통한 이더리움의 JSON-RPC클라이언트 API의 완전한 자바스크립트 구현체이다.

web3.js는 ABI를 이용해서 EVM과 통신한다.

ABI는 모든 함수명, 함수의 입출력 명세, 인자의 모든 정보를 JSON형태로 제공한다.

컨트랙트 내의 함수를 호출하거나 컨트랙트로부터 데이터를 조회할 수 있다.

Solidity, Remix, Ganache, Web3.js를 이용한다.

리믹스에서 MyAuction.sol파일을 만든다.

```
pragma solidity ^0.4.24;
```

```
contract Auction { // 하나의 경매(차량) 상품을 여러 입찰자가 입찰을 한다.
```

```
    address internal auction_owner;    // 옥션의 오너, 상품을 제공한 사람
```

```
    uint256 public auction_start;      // 경매 시작 시간
```

```
    uint256 public auction_end;        // 경매 종료 시간
```

```
    uint256 public highestBid;         // 입찰금액
```

```
    address public highestBidder;      // 입찰자들
```

```
    enum auction_state { // 경매 상태
```

```
        CANCELLED, STARTED
```

```
        // 취소        // 시작
```

```
    }
```

```
    struct car { // 구조체
```

```
        string Brand; // 차 종
```

```
        string Rnumber; // 차 고유 번호
```

```
    }
```

```
    car public Mycar; // 구조체인 Mycar 선언
```

```
    address[] bidders; // 지갑주소 또는 컨트랙트 주소를 저장하는 배열, 입찰자들을 담음
```

```
    mapping(address => uint) public bids; // 주소를 키로 입찰금액이 값인 bids
```

```
    auction_state public STATE; //auction_state 구조체 변수 STATE
```

```
    modifier an_ongoing_auction(){ // 현재 진행 중인지 확인 하기 위한 modifier
```

```
        require(now <= auction_end); // 종료 날짜보다 현재날짜가 적으면 진행중인 옥션
```

```
        _;
```

```
    }
```

```
    // modifier는 어떤 함수를 수행할 때 사전 조건을 검사하는 용도로 사용
```

```

// require는 필수를 의미
modifier only_owner(){ // 현재 접속한 사용자가 옥션은 만든 사용자인지 확인
    require(msg.sender == auction_owner);
    _;
}

// 자바의 추상메소드와 같다.
//Ether를 받는 함수인 경우 payable을 사용해야 한다.
function bid() public payable returns (bool) {}
function withdraw() public returns (bool) {}
function cancel_auction() external returns (bool) {}

// 스마트컨트랙트에서 반응이 있을 때 반응하는 함수
// 입찰이 있을 때마다 실행, indexed를 주므로 여러 입찰자에서 해당 입찰자를 확인할 수 있다.
event BidEvent(address indexed highestBidder, uint256 highestBid);

event WithdrawalEvent(address withdrawer, uint256 amount);
event CanceledEvent(string message, uint256 time);
}

contract MyAuction is Auction{ // Auction을 상속 받은 MyAuction 컨스트랙트
    // 생성자          경매유지시간      옥션소유자          차종          차량번호
    constructor(uint _biddingTime, address _owner,string _brand,string _Rnumber)
public {
    auction_owner = _owner;
    auction_start=now; // 현재 시간이 경매 시작 시간
    auction_end = auction_start + _biddingTime * 1  hours; // 시간을 곱한다.
    STATE=auction_state.STARTED; // 경매상태
    Mycar.Brand=_brand;
    Mycar.Rnumber=_Rnumber;
}

// 입찰자가 입찰할 때 실행되는 함수, 입찰할 때 Ether를 사용해야 하므로 payable이 필요
// modifier로 지정된 an_ongoing_auction함수가 실행이 되어 유효성 검사를 한다.
// 현재 진행 중이 경매인지 검사
// an_ongoing_auction함수 실행이 true이면 계속 실행, false이면 종료됨
function bid() public payable an_ongoing_auction returns (bool){
    require(bids[msg.sender]+msg.value> highestBid,"You can't bid, Make a
higher Bid"); // bids[msg.sender]는 현재 보낸 사람의 값

```

```

        // 기존의 경매한 금액보다 큰지 확인
        // false가 되면 종료되고 남은 가스는 반환된다.
        highestBidder = msg.sender; // 현재 보낸사람을 최고 경매자로 저장
        highestBid = msg.value; // 현재 금액을 최고 경매 금액으로 저장
        bidders.push(msg.sender); // 누가 입찰했지는 배열에 저장
        bids[msg.sender]= bids[msg.sender]+msg.value;//현재 입찰한 사람의 값을 저장
        ///      기존 입찰한 금액 + 현재입찰한 금액
        emit BidEvent(highestBidder, highestBid);
        return true;
    }

    // 소유주만 취소 가능할 수 있게 only_owner함수 실행
    // 현재 진행중인 경매에 대해서만 취소 할 수 있게 an_ongoing_auction실행
    function cancel_auction() external only_owner an_ongoing_auction returns (bool){

        STATE=auction_state.CANCELLED;
        emit CanceledEvent("Auction Cancelled", now);//emit로 경매가 취소를 알림
        return true;
    }

    function destruct_auction() external only_owner returns (bool){

        require(now > auction_end,"You can't destruct the contract,The auction is still open");
        for(uint i=0;i<bidders.length;i++){
            {
                assert(bids[bidders[i]]==0);
            }

            selfdestruct(auction_owner);
            return true;

        }

        // 경매 입찰을 철회
        function withdraw() public returns (bool){
            require(now > auction_end ,"You can't withdraw, the auction is still open");
            // 경매 끝 시간 이후에 들어옴.
            uint amount; // 현재 들어온 입찰자의 입찰 금액을 저장하기 위한 변수

```

```

        amount=bids[msg.sender];
        bids[msg.sender]=0; // 현재 접속한 입찰자의 입찰 금액을 0으로 저장
        msg.sender.transfer(amount); // 입찰금액을 전송
        emit WithdrawalEvent(msg.sender, amount);
        return true;
    }
    // view는 보여주기만 하는 것이라 카스 소모를 하지 않는다.
    function get_owner() public view returns(address){
        return auction_owner; // 누가 오너인지 확인
    }
}

auction.js
// var web3 = new Web3();
var web3 = new Web3('ws://localhost:7545'); // ws은 웹소켓, 웹소켓으로 가나슈에 접속

// web3.setProvider(new web3.providers.HttpProvider("http://localhost:7545"));

var bidder;
//= web3.eth.getAccounts(); // web3.eth.accounts[0];

// getAccounts를 이용하여 가나슈에 있는 계정을 모두 가져온다.
web3.eth.getAccounts().then(function(acc){
    // bidder=acc;
    console.log(acc)
    web3.eth.defaultAccount = acc[0]
    bidder = acc[0]

    // contract에 있는 auction_end의 값을 methods를 이용하여 result에 저장
    auctionContract.methods.auction_end().call().then( (result)=>{
        document.getElementById("auction_end").innerHTML=result;
    });// result 값을 html태그에 출력

    auctionContract.methods.highestBidder().call().then( (result)=>{
        document.getElementById("HighestBidder").innerHTML=result;
    });

    auctionContract.methods.highestBid().call().then( (result)=>{
        console.log("highest bid info: ", result)
    })

```

```

        var bidEther = web3.utils.fromWei(web3.utils.toBN(result), 'ether');
                                // wei를 ether로 변환
        document.getElementById("HighestBid").innerHTML=bidEther;

    });

    auctionContract.methods.STATE().call().then( (result)=>{
document.getElementById("STATE").innerHTML=result;

    });

                                // Mycar값 받아오기
    auctionContract.methods.Mycar().call().then( (result)=>{

        // 구조체 안에 있는 변수는 배열로 받아 온다.
        document.getElementById("car_brand").innerHTML=result[0];
        document.getElementById("registration_number").innerHTML=result[1];

    });

//          address[] bidders: 입찰자의 지갑주소
//          mapping(address => uint) public bids   입찰자를 키로한 입찰금액
    auctionContract.methods.bids(bidder).call().then( (result) => {

        var bidEther = web3.utils.fromWei(web3.utils.toBN(result), 'ether');
        document.getElementById("MyBid").innerHTML=bidEther;

        console.log(bidder);

    });

});

// web3.eth.defaultAccount = bidder;
var auctionContract = new web3.eth.Contract(
    [
    {
        "constant": false,
        "inputs": [],
        "name": "bid",
        "outputs": [

```

```

    {
      "name": "",
      "type": "bool"
    }
  ],
  "payable": true,
  "stateMutability": "payable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [],
  "name": "cancel_auction",
  "outputs": [
    {
      "name": "",
      "type": "bool"
    }
  ],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [],
  "name": "destruct_auction",
  "outputs": [
    {
      "name": "",
      "type": "bool"
    }
  ],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": false,
  "inputs": [],
  "name": "withdraw",

```

```

"outputs": [
  {
    "name": "",
    "type": "bool"
  }
],
"payable": false,
"stateMutability": "nonpayable",
"type": "function"
},
{
  "inputs": [
    {
      "name": "_biddingTime",
      "type": "uint256"
    },
    {
      "name": "_owner",
      "type": "address"
    },
    {
      "name": "_brand",
      "type": "string"
    },
    {
      "name": "_Rnumber",
      "type": "string"
    }
  ],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "constructor"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "name": "highestBidder",
      "type": "address"
    },
  ],

```



```
{
  "indexed": false,
  "name": "highestBid",
  "type": "uint256"
},
{
  "name": "BidEvent",
  "type": "event"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": false,
      "name": "withdrawer",
      "type": "address"
    },
    {
      "indexed": false,
      "name": "amount",
      "type": "uint256"
    }
  ],
  "name": "WithdrawalEvent",
  "type": "event"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": false,
      "name": "message",
      "type": "string"
    },
    {
      "indexed": false,
      "name": "time",
      "type": "uint256"
    }
  ],
  "name": "CanceledEvent",
```

```

    "type": "event"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "auction_end",
    "outputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [],
    "name": "auction_start",
    "outputs": [
      {
        "name": "",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": true,
    "inputs": [
      {
        "name": "",
        "type": "address"
      }
    ],
    "name": "bids",
    "outputs": [
      {

```

```

        "name": "",
        "type": "uint256"
    }
],
"payable": false,
"stateMutability": "view",
"type": "function"
},
{
    "constant": true,
    "inputs": [],
    "name": "get_owner",
    "outputs": [
        {
            "name": "",
            "type": "address"
        }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
},
{
    "constant": true,
    "inputs": [],
    "name": "highestBid",
    "outputs": [
        {
            "name": "",
            "type": "uint256"
        }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
},
{
    "constant": true,
    "inputs": [],
    "name": "highestBidder",
    "outputs": [

```

```

        {
            "name": "",
            "type": "address"
        }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
},
{
    "constant": true,
    "inputs": [],
    "name": "Mycar",
    "outputs": [
        {
            "name": "Brand",
            "type": "string"
        },
        {
            "name": "Rnumber",
            "type": "string"
        }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
},
{
    "constant": true,
    "inputs": [],
    "name": "STATE",
    "outputs": [
        {
            "name": "",
            "type": "uint8"
        }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}

```

```

]
);
var contractAddress = '0x2783eAA4BB06C2C15288b33AcF38226252a88E12';
// var auction = auctionContract.at(contractAddress);
auctionContract.options.address = '0x2783eAA4BB06C2C15288b33AcF38226252a88E12';

function bid() {
var mybid = document.getElementById('value').value;

// 스마트 컨트랙트에 있는 bid()함수 호출
auctionContract.methods.bid().send(
    { from: '0xa6623cc5264B05757852AD6aB663dB90E171080e',
      value: web3.utils.toWei(mybid, "ether"), gas: 200000}).then((result)=>{
        console.log(result)
    })

//

    document.getElementById("biding_status").innerHTML="Successfull bid, transaction
ID : "+ result.transactionHash;

});

// Automatically determines the use of call or sendTransaction based on the
method type
// auctionContract.methods.bid().call({value: web3.utils.toWei(mybid, "ether"), gas:
200000}, function(error, result){
//   if(error)
//     {console.log("error is "+ error);
//     document.getElementById("biding_status").innerHTML="Think to bidding higher";
//   }
//   if (!error)
//     document.getElementById("biding_status").innerHTML="Successfull bid,
transaction ID"+ result;
// });

}

function init(){
    // setTimeout(() => alert("아무런 일도 일어나지 않습니다."), 3000);

```

```
}
```

```
var auction_owner=null;
```

```
auctionContract.methods.get_owner().call().then((result)=>{
```

```
    auction_owner=result;
```

```
    if(bidder!=auction_owner)
```

```
        $("#auction_owner_operations").hide();
```

```
})
```

```
// auctionContract.methods.get_owner(function(error, result){
```

```
    // if (!error){
```

```
        // auction_owner=result;
```

```
        // if(bidder!=auction_owner)
```

```
            // $("#auction_owner_operations").hide();
```

```
        // }
```

```
// }
```

```
// );
```

```
function cancel_auction(){
```

```
    // .auction_end().call().
```

```
auctionContract.methods.cancel_auction().call().then( (result)=>{
```

```
    console.log(result)
```

```
});
```

```
}
```

```
function Destruct_auction(){
```

```
auctionContract.methods.destruct_auction().call().then( (result)=>{
```

```
    console.log(result) //The auction is still open when now() time < auction_end  
time
```

```
});
```

```
// auctionContract.methods.destruct_auction().call({from:  
'0xAf91a82C2D968c6Ed3eeFe5dD90D860d011B7B0A'},function(error, result){
```

```
// console.log(result);
```

```
// });
```

```
}
```

```

auctionContract.events.BidEvent(/*{highestBidder:"A",highestBid:"888"},*/function(error
, event){
    console.log(event);
})
.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.on('data', function(event){
    console.log(event); // same results as the optional callback above
    $("#eventslog").html(event.returnValues.highestBidder + ' has bidden(' +
event.returnValues.highestBid + ' wei)');

})
.on('changed', function(event){
    // remove event from local database
    console.log(event);
})
// var BidE
/*filter.get(callback): Returns all of the log entries that fit the filter.
filter.watch(callback): Watches for state changes that fit the filter and calls the
callback. See this note for details.*/
// var BidEvent = auctionContract.events.BidEvent(); // var BidEvent =
auction.BidEvent({}, {fromBlock: 0, toBlock: 'latest'});

// BidEvent.watch(function(error, result){
//     if (!error)
//     {
//         $("#eventslog").html(result.args.highestBidder + ' has bidden('
+ result.args.highestBid + ' wei)');
//     } else {

//         console.log(error);
//     }
// });
auctionContract.events.CanceledEvent( function(error, event){
console.log(event);
})

```

```

.on("connected", function(subscriptionId){
    console.log(subscriptionId);
})
.on('data', function(event){
    console.log(event); // same results as the optional callback above
    $("#eventslog").html(event.args.message+' at '+event.args.time);
})
.on('changed', function(event){
    // remove event from local database
})
.on('error', function(error, receipt){ // If the transaction was rejected by the
network with a receipt, the second parameter will be the receipt.

});

// var CanceledEvent = auctionContract.events.CanceledEvent();

// CanceledEvent.watch(function(error, result){
//     if (!error)
//     {
//         console.log(result);

//         $("#eventslog").html(result.args.message+' at
'+result.args.time);
//     } else {

//         console.log(error);
//     }
// });

// const filter = web3.eth.subscribe({
//     fromBlock: 0,
//     toBlock: 'latest',
//     address: contractAddress,
//     topics: [web3.utils.sha3('BidEvent(address,uint256)')]
// }, function(error, result){
//     if (!error)
//     console.log(result);
// })

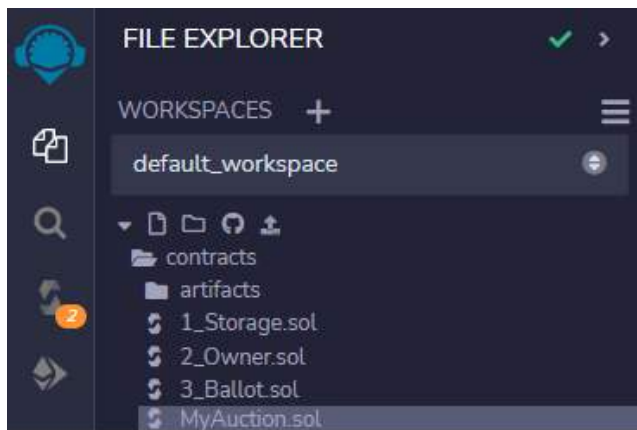
```



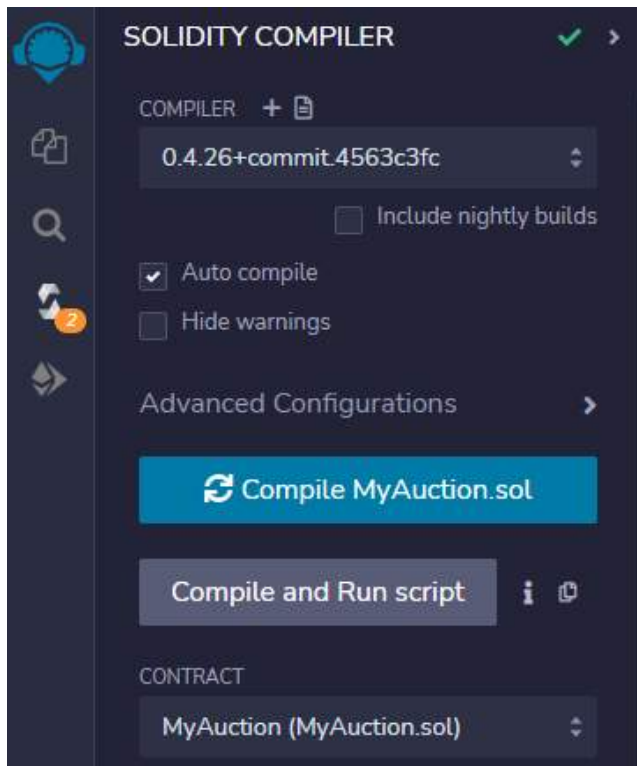
```
// filter.get((error, result) => {
//     if (!error)
//     console.log(result);
//     //console.log(result[0].data);

// })
```

리믹스에서 컨트랙트를 생성하고 컴파일



솔리디티 파일을 생성하고 위 내용을 작성한다.



CONTRACT를 MyAuction.sol변경한 후 컴파일한다.


EVM을 가나슈로 변경한 후 배포.(가나슈 포트인 7545로 변경)
생성자에서 받는 4개의 PARAMETER값을 입력한다.

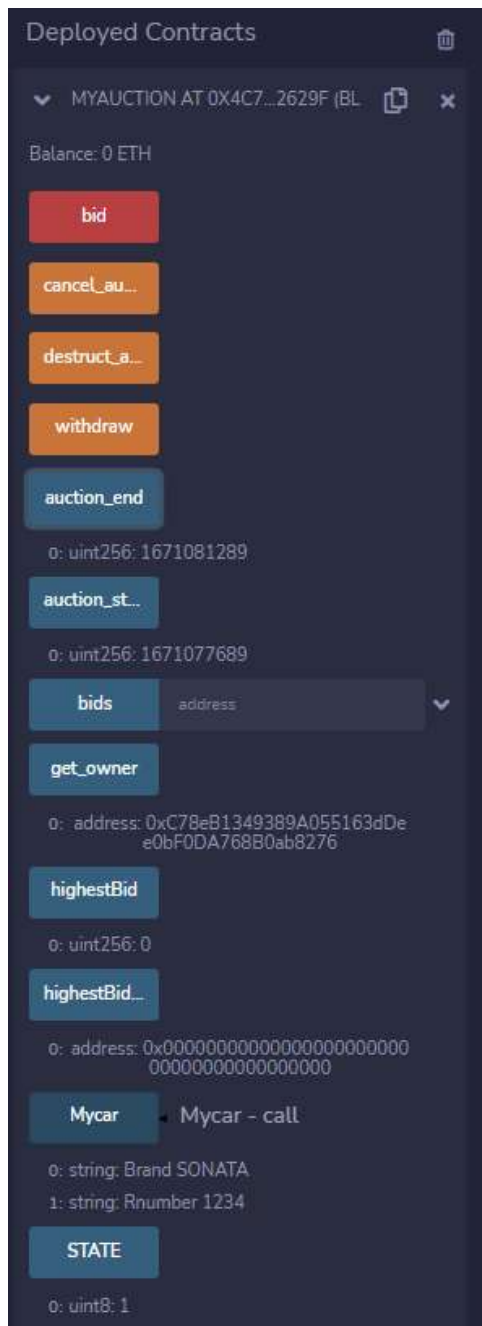
_BIDDINGTIME은 8시간

_OWNER는 가나슈에서 받은 ACCOUNT중 하나를 적어준다.

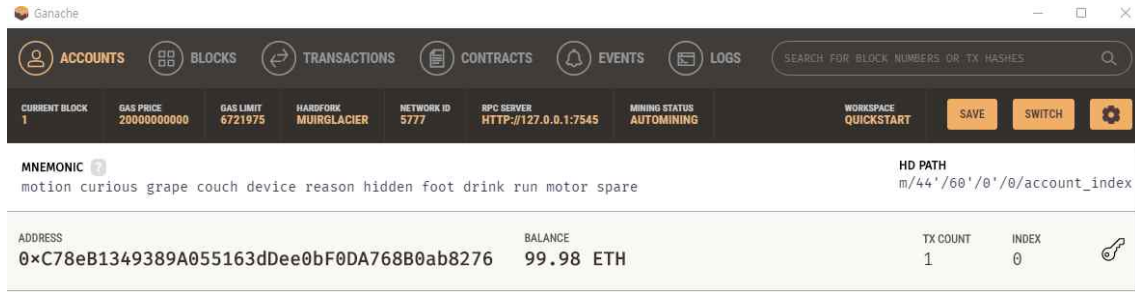
0xC78eB1349389A055163dDee0bF0DA768B0ab8276

_BRAND는 SONATA

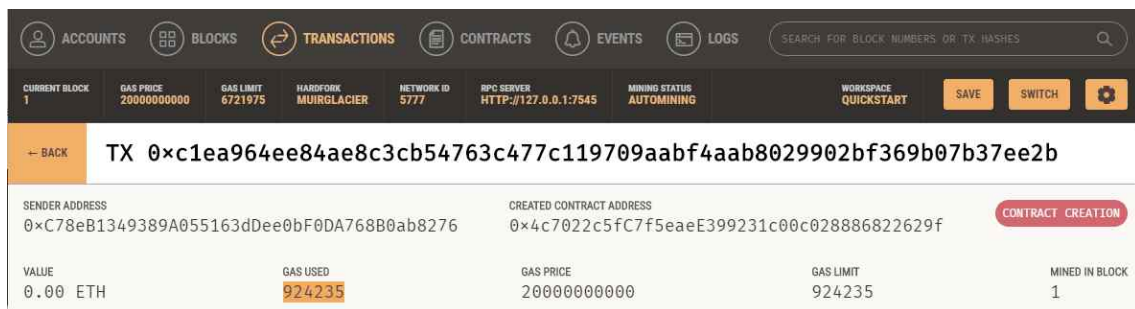
_RNUMBER은 1234로 하고  을 클릭하면 배포가 되어서 아래 그림과 같은 내용이 뜬다.



가나슈에서 확인해 보면 배포하면서 가스가 소비된 것을 알 수 있다.



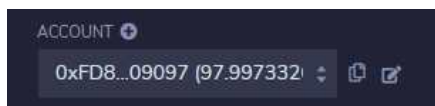
0xC78eB1349389A055163dDee0bF0DA768B0ab8276에서 배포했음을 알 수 있다.



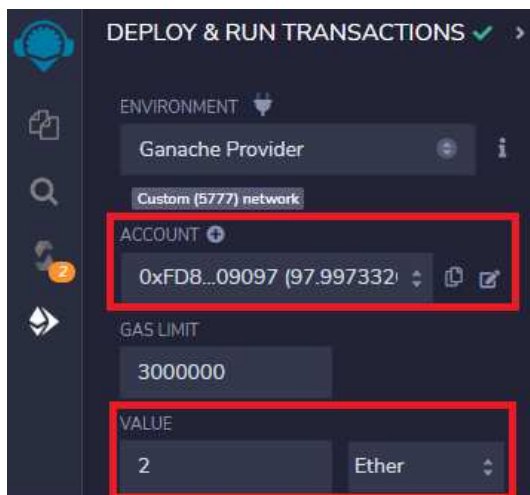
TRANSACTION에서 924235만큼의 가스를 사용한 것을 확인 할 수 있다.

이제 리믹스에서 입찰을 해보자

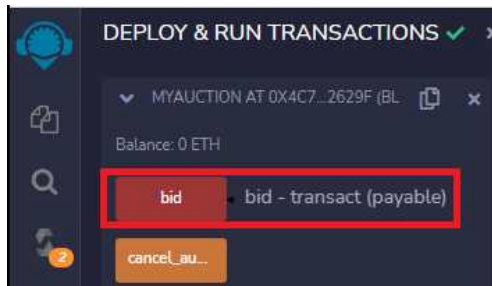
리믹스로부터 받은 지갑주소 중에서 먼저 입찰자를 선택을 해준다.



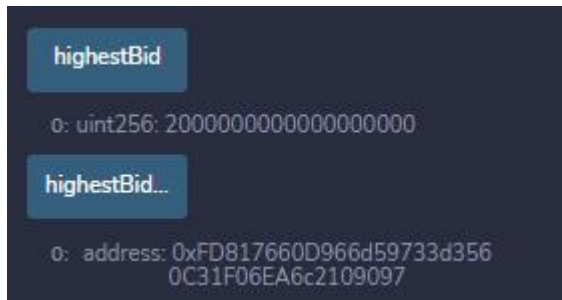
먼저 입찰 금액을 2ETHER 로 적어 준다.



입력 후 입찰을 해보자.

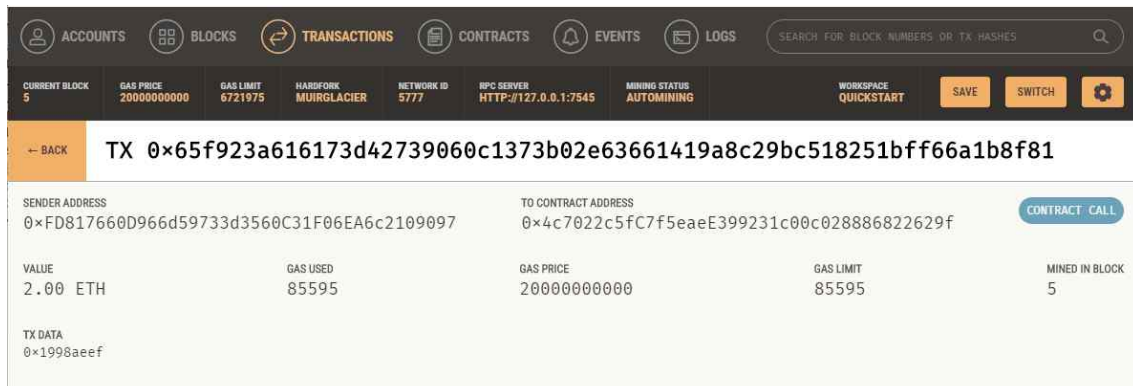


highestBid와 highestBidder 확인해 보면 입찰

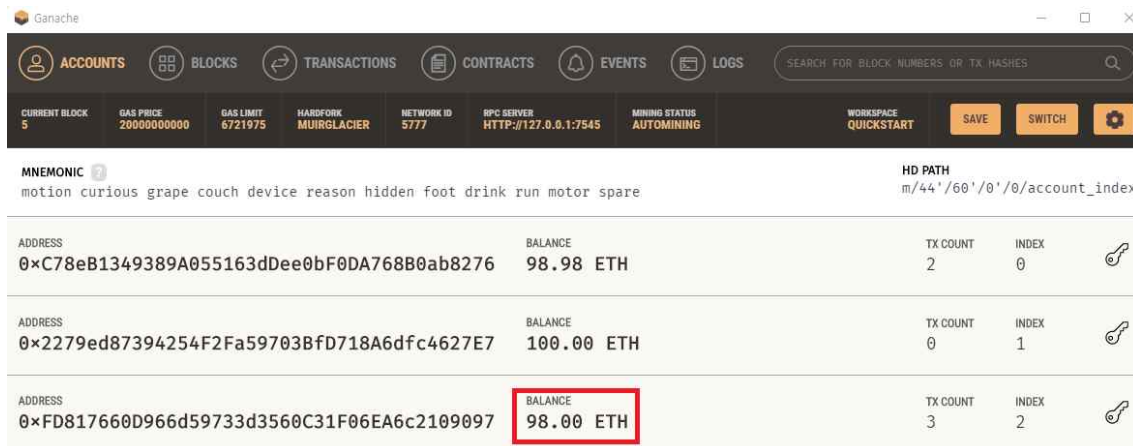


각가 2ether와 입찰자가 변경된 것을 확인할 수 있다.

리믹스에서 확인해보면



트랜잭션이 발생한 것을 알 수 있으면 85595만큼의 가스와 2ether만큼 사용한 것을 알 수 있다.



이제 html을 이용하여 입찰을 해보자.
index.html파일을 만들고 아래 내용을 적어준다.

```
<html>
<head>
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
<!-- <script
src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.6.1/web3.min.js"></script>
-->
<script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@3.0.0/dist/web3.min.js"></script>
<!--script src="bower_components/abi-decoder/dist/abi-decoder.js"></script-->

<script src="./auction.js"></script>
</head>
<body onload="init()">
  <div class="col-lg-12">
    <div class="page-header">
      <h3>Car Auction Dapp</h3>
    </div>
    <div class="col-lg-6">
      <div class="well">
        <div>
          <legend class="lead">Car Details</legend>
          <label class="lead"> Brand: </label><text id='car_brand'></text><br>
          <label class="lead">Registration Number: </label><text
id='registration_number'></text>
        </div>
        <div>
          <legend class="lead">Bid value</legend>
          <small>eg. 100</small>
          <div class="form-group">
            <input class="form-control" type="text" id="value" value="10"
          ></input>
          <text id="valueValidation"></text>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

```

        <button class="btn btn-default" id="transfer" type="button"
onClick="bid()">Bid!</button><br>
        <text id="biding_status"></text>

    </div>
</div>
</div>
<div class="col-lg-6">
    <div class="well">
        <div>
            <legend class="lead">Auction Details</legend>
        </div>
        <div>
            <ul id='transfers'>
                <li><label class="lead">Auction End: </label>
<text id="auction_end"></text></li>
                <li><label class="lead">Auction Highest Bid:
</label> <text id="HighestBid"></text></li>
                <li><label class="lead">My Bid: </label> <text
id="MyBid"></text></li>
                <li><label class="lead">Auction Highest Bider:
</label> <text id="HighestBidder"></text></li>
                <li><label class="lead">Auction Status:
</label> <text id="STATE"></text></li>
            </ul>
        </div>
    <br>
    <div>
        <legend class="lead">Events Logs </legend>

        <text id="eventslog"></text>
    </div>
</div>
</div>
</div>
<div class="col-lg-6" id="auction_owner_operations">
    <div class="well"> <div>
        <legend class="lead">Auction Operations</legend>
        <button class="btn btn-default" id="transfer" type="button"
onClick="cancel_auction()">Cancel auction!</button>
        <button class="btn btn-default" id="transfer" type="button"
onClick="Destruct_auction()">Destruct auction!</button>
    </div></div></div>

</body>

```

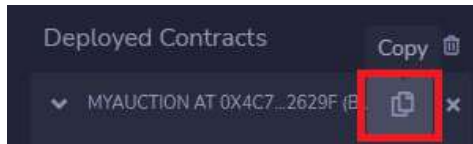
</html>

같은 폴더에 auction.js파일을 만들어 위에 적은 자바스크립트 내용을 적어주면 된다.

auction.js파일에서 contract_address를 바꿔줘야 한다.

abi는 리믹스에서 가져온 것이어서 그대로 사용해도 된다.

contract_address는 리믹스에서 복사해오면 된다.



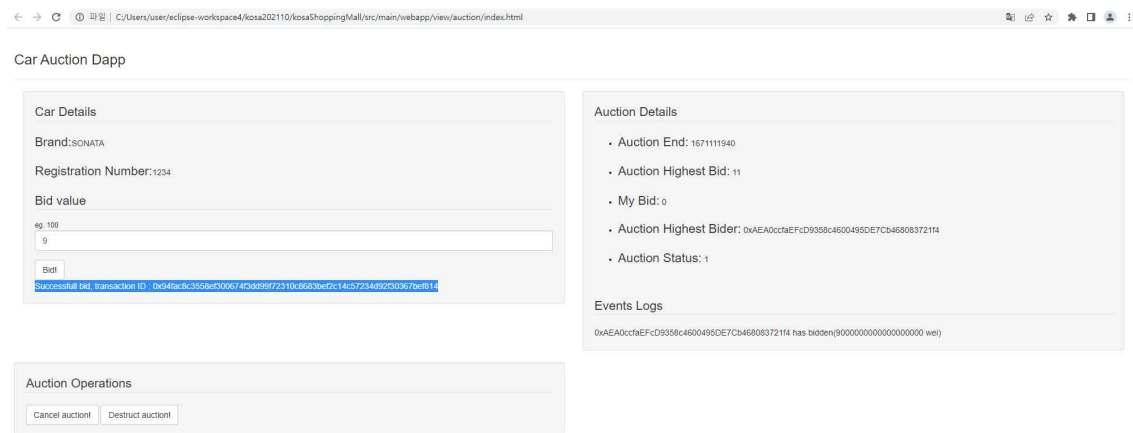
auction.js에서 contractAddress변수를 찾아 리믹스에서 복사해 온 주소를 붙여넣는다.

예)

```
var contractAddress = '0x4c7022c5fC7f5eaeE399231c00c028886822629f';
```

```
auctionContract.options.address = '0x4c7022c5fC7f5eaeE399231c00c028886822629f';
```

그런 후 index.html파일을 실행시킨다.



입찰을 하기 전에 입찰자를 가나슈에 있는 지갑주소 중 하나를 복사해 오고 .

auctionContract.methods.bid().send()에 있는 from에 있는 주소를 바꿔주고 웹 브라우저를 새로고침 해준다.

예)

```
from: '0x2279ed87394254F2Fa59703BfD718A6dfc4627E7'
```

https://github.com/wonyongHwang/kopoSmartContract/tree/main/1_AuctionExample

이더리움에서 토큰 발행하기

<http://wiki.hash.kr/index.php/ERC-20>

<https://docs.openzeppelin.com/contracts/2.x/api/token/erc20>

<https://docs.openzeppelin.com/contracts/3.x/erc20>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol> 에 있는 소스를 보게되면

토큰은 ERC20규약을 따르게 되므로 ERC20규약으로 만든 openzeppelin API를 사용하면된다

각 지갑 주소에 따른 잔고를 가져오기 위한 변수

```
mapping(address => uint256) private _balances;
```

발행량을 가져오는 변수

```
uint256 private _totalSupply;
```

토큰의 이름

```
string private _name;
```

토큰의 심볼(BTC)

```
string private _symbol;
```

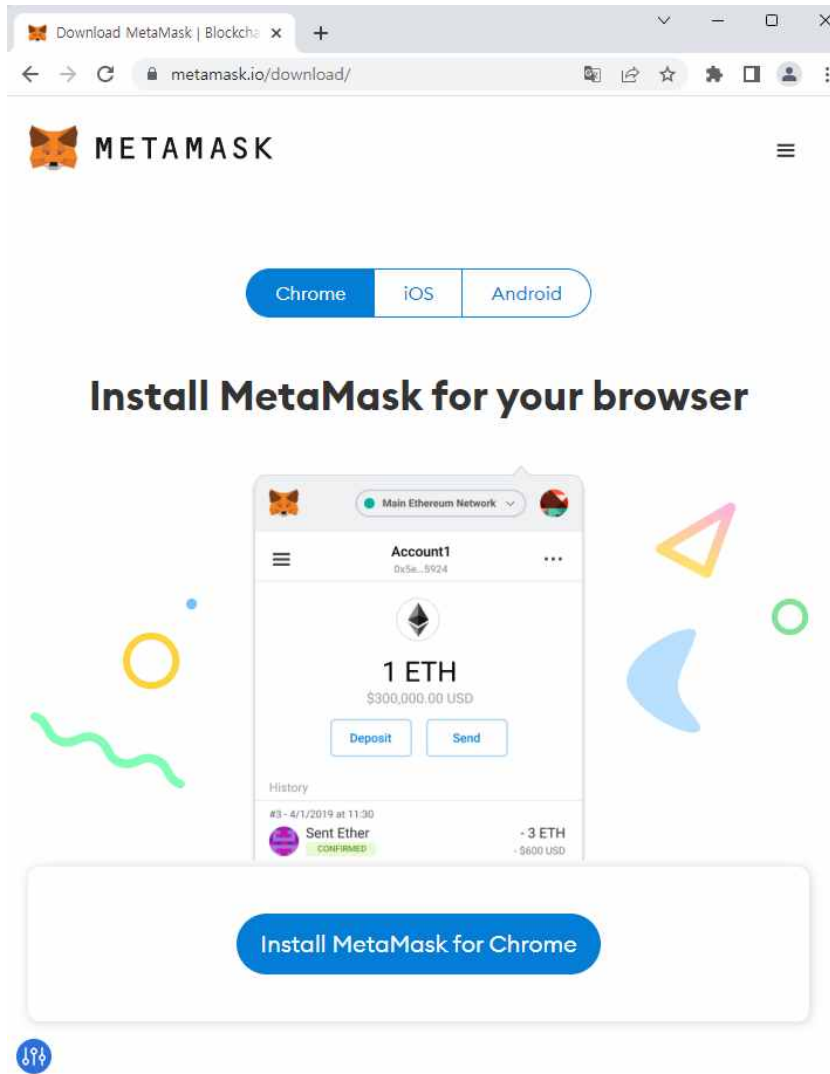
```
function _transfer(
    address from, // 어느 지갑으로 부터
    address to,   // 어느 지갑으로 얼마
    uint256 amount
) internal virtual {
    ...
    unchecked {
        _balances[from] = fromBalance - amount; // 잔고에서 보낸 만큼 차감
        // Overflow not possible: the sum of all balances is capped by
        totalSupply, and the sum is preserved by
        // decrementing then incrementing.
        _balances[to] += amount; // 다른 계좌에 더해준다.
    }
    ..
}
```

// 발행량을 설정할 수 있는 함수

// 예) 10만개를 하고 싶다면 발행량은 $100000 * 10^{18}$ 로 해야 한다


```
function decimals() public view virtual override returns (uint8) {  
    return 18;  
}
```

테스트넷에 배포하기 위해 MetaMask 설치
<https://metamask.io/download/>



사이트에 들어와서 install MetaMask for Chrome을 클릭해서 설치를 하면된다,



MetaMask가 처음이세요?



아니요, 이미 비밀 복구 구문이 있습니다.

비밀 복구 구문을 사용하여 기존 지갑 가져오기

지갑 가져오기



설정을 시작하죠!

이렇게 하면 새 지갑과 비밀 복구 구문이 만들어집니다

지갑 생성

지갑생성을 클릭한다.



< 뒤로

비밀번호 생성

새 비밀번호(8자 이상)

비밀번호 확인

☐

이용 약관의 내용을 읽고 이에 동의합니다.

생성

비밀번호를 입력한 후 [생성]을 클릭한다.



METAMASK

지갑 보호하기

시작하기 전에 이 짧은 동영상을 보고 비밀 복구 구문과 지갑을 안전하게 보호하는 방법에 대해 알아보세요.



다음

[다음]버튼을 클릭한다.



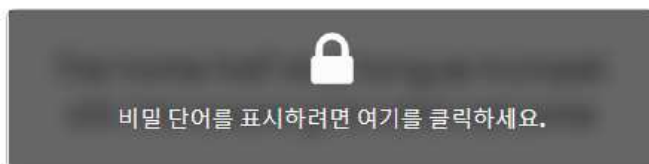
METAMASK

< 뒤로

비밀 복구 구문

비밀 백업 구문을 이용하면 계정을 쉽게 백업하고 복구할 수 있습니다.

경고: 비밀 복구 구문은 절대로 공개하지 마세요. 이 구문이 있는 사람은 귀하의 **Ether**를 영원히 소유할 수 있습니다.



나중에 알림

다음

[비밀 단어를 표시하려면 여기를 클릭하세요]를 클릭하면 비밀글이 나오고 [다음]이 활성화된다. 비밀글은 캡처를 해 놓는 것이 좋다.

'비밀 복구 구문'이란 무엇인가요?

복구 구문은 지갑과 자금의 '마스터 키'입니다.

비밀 복구 구문은 어떻게 저장 하나요?

- 비밀번호 관리자에 저장
- 은행 금고에 보관.
- 대여 금고에 보관.
- 적어서 여러 비밀 장소에 보관하세요.

비밀 복구 구문을 공유해야 하나요?

절대로, 누구와도, 심지어 MetaMask와도 비밀 구문을 공유하면 안 됩니다!

복구 구문을 요청하는 사람은 사기를 치려는 것입니다.

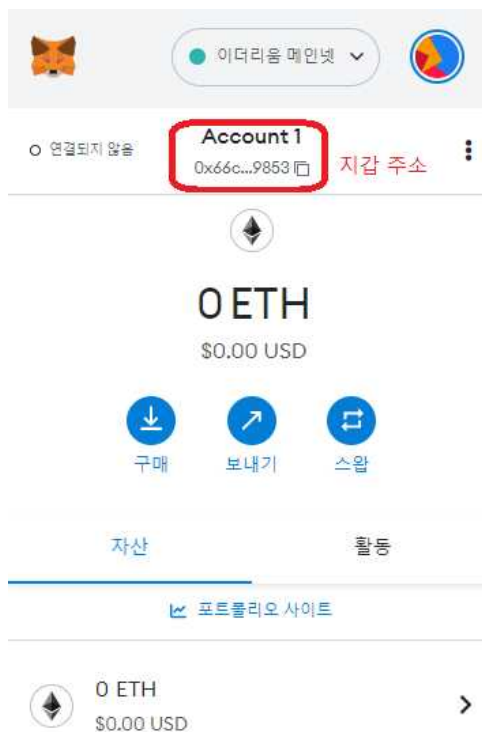
비밀 백업 구문 확인

각 구문을 선택하여 구문이 올바른지 확인하세요.

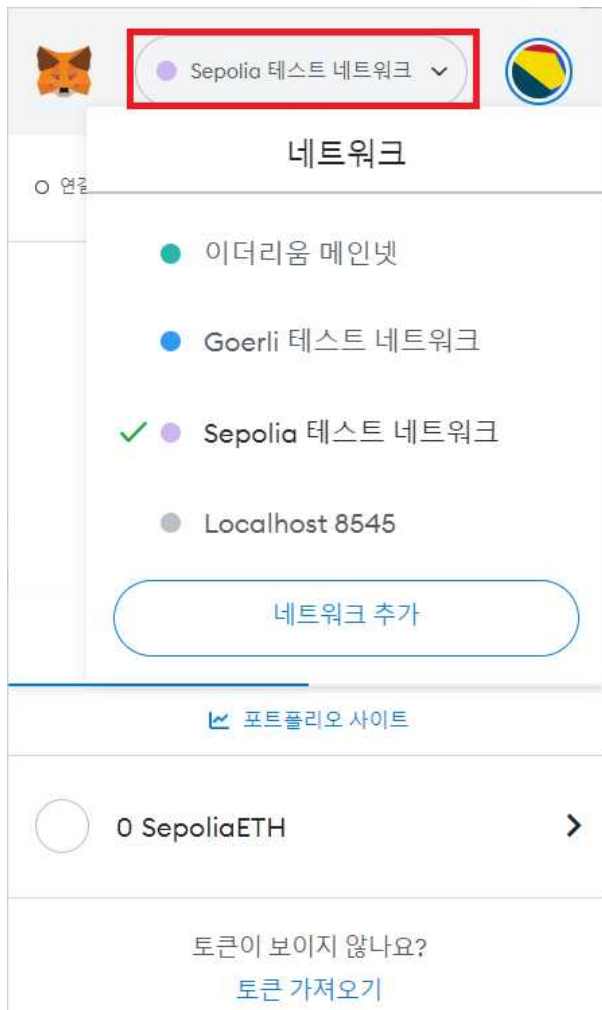
aware	fire	grid	half
home	into	silk	skull
tongue	trumpet	valid	welcome

확인

캡처한 비밀글 순서대로 선택한 후 [확인]



오른쪽 상단에 있는 이미지를 클릭하면 설정으로 들어갈 수 있다.



메뉴중 [고급]으로 들어간다.

테스트넷에 전환 표시

이 항목을 선택하면 테스트넷에 명목 전환이 표시됩니다.

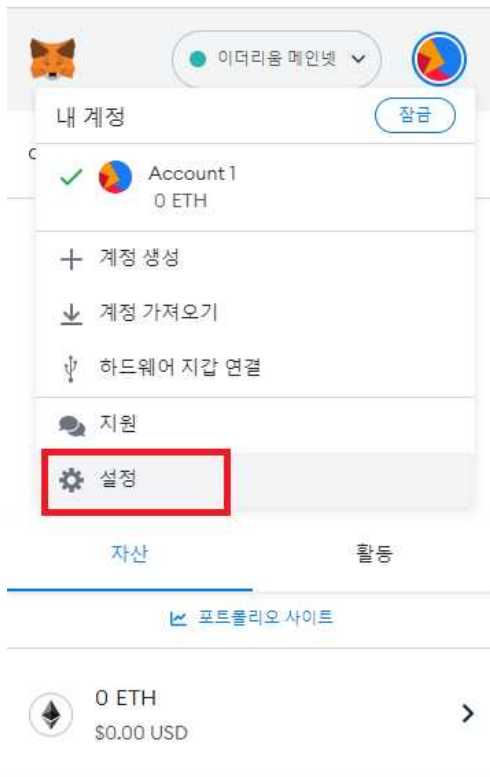
☒ 켜기

테스트 네트워크 보기

네트워크 목록에서 표시하려는 테스트 네트워크를 선택하세요.

☒ 켜기

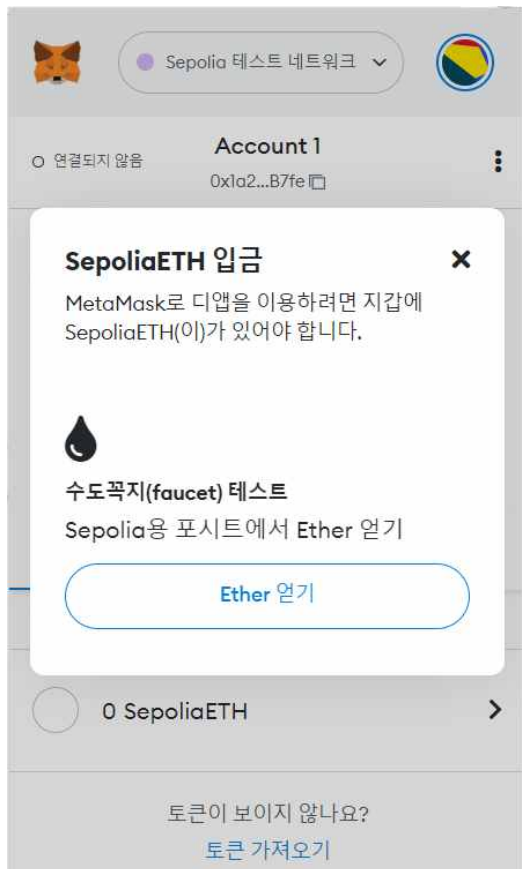
두 개를 켜기로 바꿔준다.



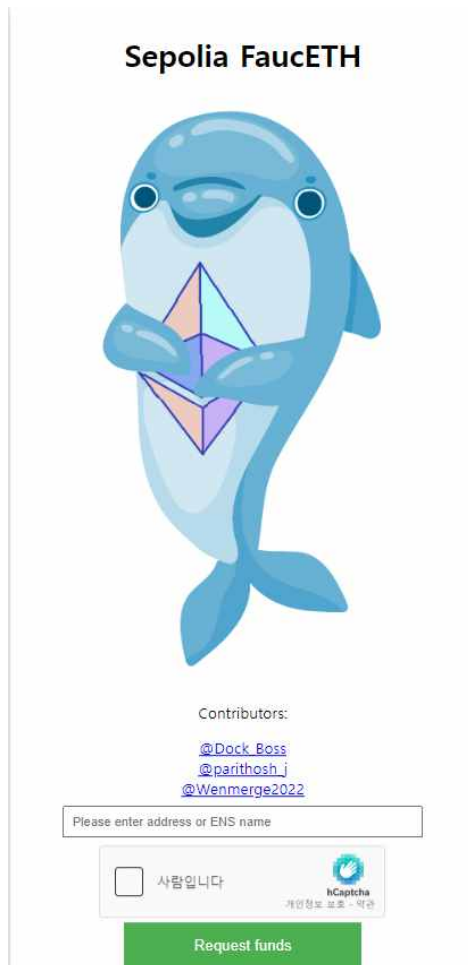
토큰이 보이지 않나요?
테스트넷 중 하나를 선택한다.



위에 있는 지갑 주소를 복사한 후 구매 이미지를 클릭한다.



Ether 얻기를 선택한다.

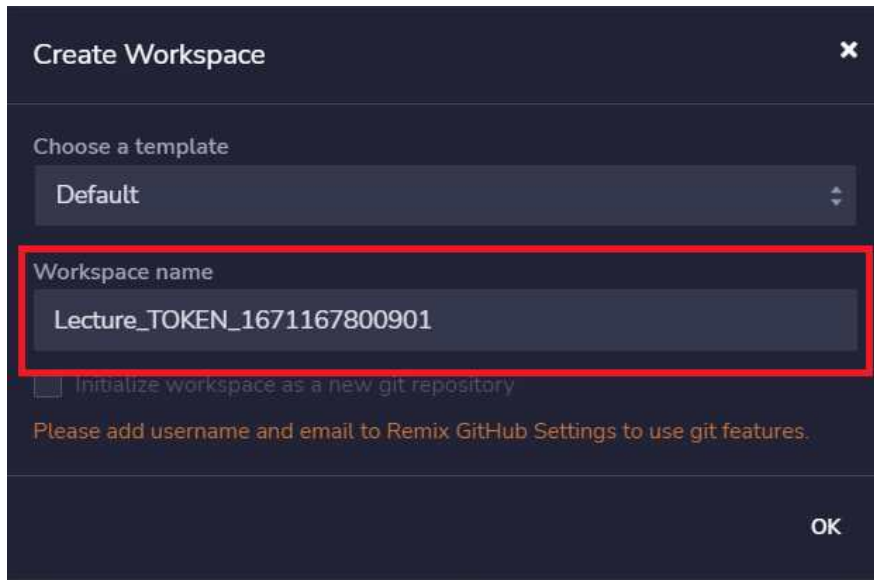


지갑 주소를 입력한 후 사람입니다를 체크 후 Request funds를 클릭한다.

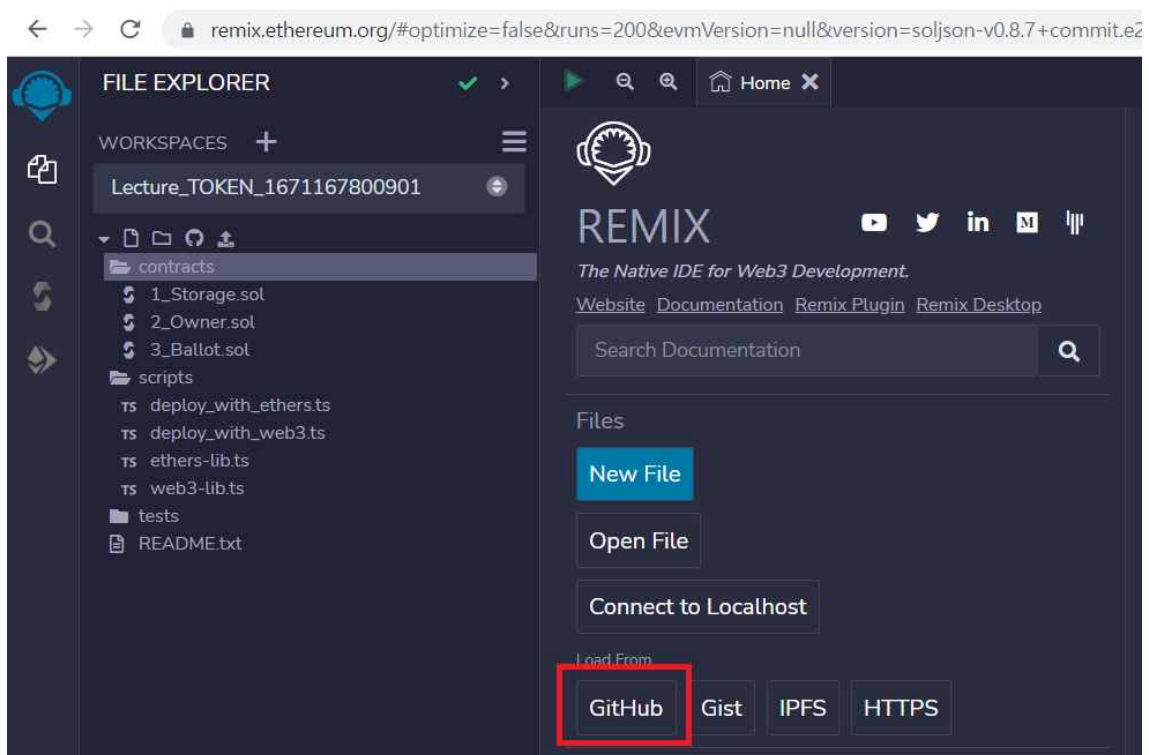


다소 시간이 걸릴 수 있다.

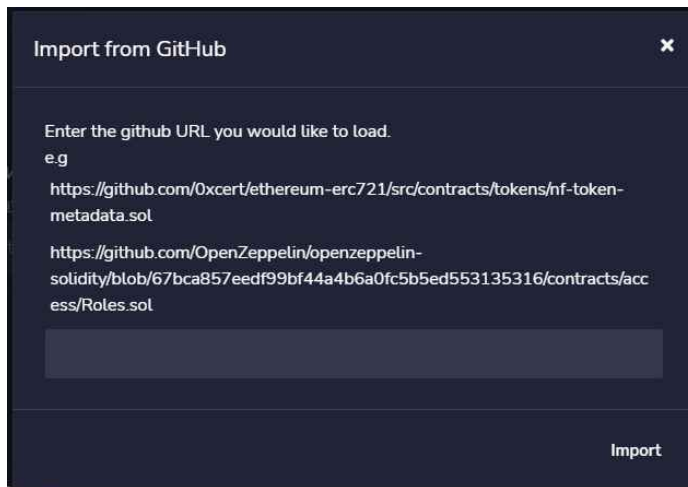
리믹스를 열어서 WorkSpace를 새로 만들어 보자.



OpenZeppelin을 깃허브에서 가져오자.



GitHub를 클릭하면



빈칸에 아래 주소를 넣어주소 import를 한다.

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/token/ERC20/ERC20.sol>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/token/ERC20/IERC20.sol>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/GSN/Context.sol>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/math/SafeMath.sol>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/examples/SimpleToken.sol>

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.5.1/contracts/token/ERC20/ERC20Detailed.sol>

Workspace에서 SimpleToken.sol파일에서

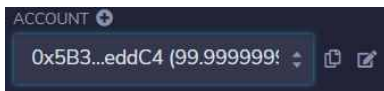
```
constructor () public ERC20Detailed("SimpleToken", "SIM", 18) {  
    _mint(_msgSender(), 10000 * (10 ** uint256(decimals())));  
}
```

에 토큰이름과 심볼 그리고 발행량을 적어줄 수 있다.

```
constructor () public ERC20Detailed("KopoToken", "KPT", 18) {  
    _mint(_msgSender(), 10000 * (10 ** uint256(decimals())));  
}
```

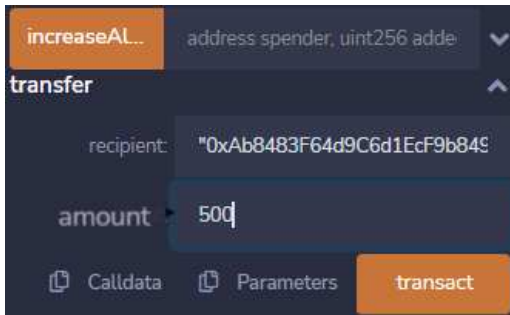
로 변경한다,

SimpleToken을 선택하고 배포를 한다.

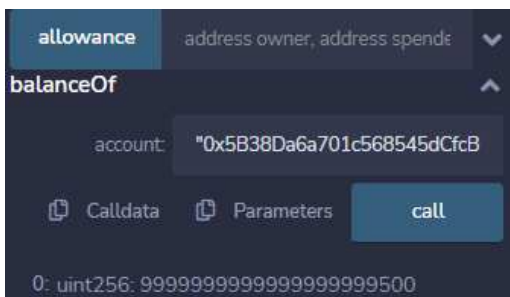


다시 소유자로 계정을 바꿔주어야 한다.

복사한 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2(지갑 주소는 다를 수 있음)주소를

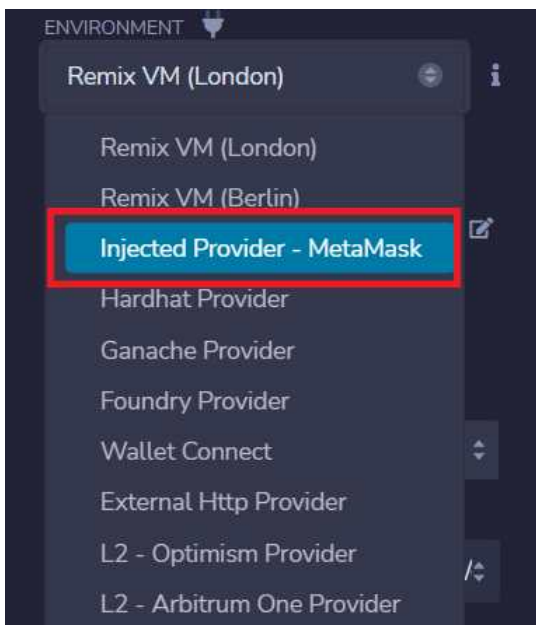


에 복사해 넣고 amount를 입력한다, 여기서는 500을 입력하고 transact를 클릭한다.

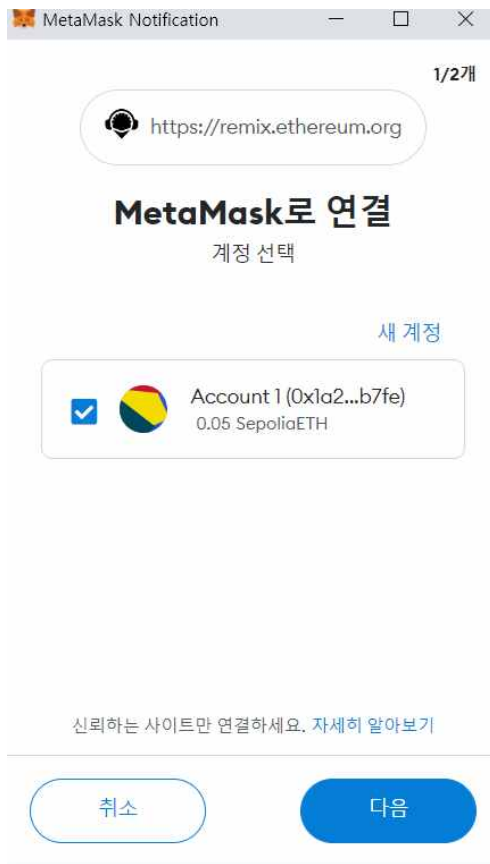


allowance에서 call을 실행시키면 500만큼 차감된 것을 알 수 있다.

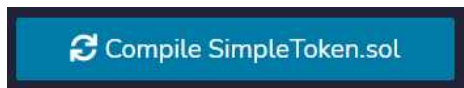
이제 메타마스크에 배포를 해보자.



EVM을 Injected Provider로 선택을 하면

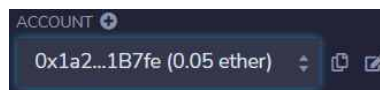


MetaMask에 있는 계정으로 배포 할 것인지 나오면 다음을 클릭한 후



리믹스에서 다시 컴파일을 한다.

메타마스크에 있는 계정인지 확인해 보고 한다.



배포  를

MetaMask Notification

Sepolia 테스트 네트워크

Account 1

→

새 계약

새로운 가스 경험

가스비 견적 산정 방법과 맞춤화 작업을 업데이트했습니다.

[설정에서 항상된 가스 수수료 UI를 활성화](#)

<https://remix.ethereum.org>

계약 배포

세부 정보

데이터

0.00278335

예상 가스 요금

0.002783 SepoliaETH

추천 사이트

최대 요금

0.00278335 SepoliaETH

0.00278335

합계

0.00278335 SepoliaETH

금액 + 가스 요금

최대 금액: 0.00278335 SepoliaETH

거부

확인

가스가 얼마나 소비되는지 확인할 수 있다.확인 버튼을 누른다.

Chrome

Confirmed transaction

Transaction 0 confirmed! View on Etherscan

배포되었다고 메시지가 나오면 MetaMask를 실행해 본다.



Sepolia 테스트 네트워크



< Account 1 / SepoliaETH

0.0472 SepoliaETH

구매

보내기

스왑

계약 배포

계약 배포

-0 SepoliaE...

Dec 16 · remix.ethereum.org

-0 SepoliaETH

받기

받기

0.05 Sepoli...

Dec 16 · 발신: 0xf29...6767

0.05 SepoliaETH

배포된 것을 확인할 수 있다.

계약 배포

상태

확인됨

블록 탐색기에서 보기

트랜잭션 ID 복사

발신

수신

 0x1a2...B7fe

→

새 계약

거래

임시값

0

금액

-0 SepoliaETH

가스 한도 (단위)

1113341

가스 사용됨 (단위)

1113341

기본 수수료(GWEI)

0.000000007

우선 수수료(GWEI)

2.5

총 가스 수수료

0.002783 SepoliaETH

가스당 최대 수수료

0.000000003 SepoliaETH

합계

0.00278335 SepoliaETH

블록 탐색기에서 확인 해볼 수 있다.

Transaction Details < >

Overview Logs (1) State

[This is a Sepolia Testnet transaction only]

⑦ Transaction Hash: 0x5874a8b99dbc0a186e2dfa7cd30b4932fc8b91b52b0f63fa6b42e7994cd45083 [트랜잭션 Id](#)

⑦ Status: Success

⑦ Block: [2489269](#) 44 Block Confirmations

⑦ Timestamp: 8 mins ago (Dec-16-2022 06:36:00 AM +UTC)

⑦ From: [0x1a29e45d9871f61fe1f980704ec13f882f01b7fe](#) [내 계정](#)

⑦ Interacted With (To): [Contract [0xe3bde42c17f1e4dd78e56369525defeaaf183069](#) Created] [Contract Id](#)

⑦ ERC-20 Tokens Transferred: ▶ From [0x00](#) To [0x1a29e45d9871f61fe1f980704ec13f882f01b7fe](#) For 10,000 [KopoToken \(KPT\)](#)

⑦ Value: 0 Ether (\$0.00)

⑦ Transaction Fee: 0.002783352507793387 Ether (\$0.00)

⑦ Gas Price: 0.000000002500000007 Ether (2.500000007 Gwei)

이제 MetaMask에서 토큰을 가져 온다.

 Sepolia 테스트 네트워크 

 연결됨 Account 1 
0x1a2...B7fe 

0.0472 SepoliaETH

 구매  보내기  스왑

자산 활동

[포트폴리오 사이트](#)

 0.0472 SepoliaETH >

토큰이 보이지 않나요?
[토큰 가져오기](#)

토큰 가져오기를 클릭한다.



토큰 가져오기 ×

커스텀 토큰

[안 위험](#) [에 대해 알아보기](#)

토큰 계약 주소

0xE3Bde42C17f1E4DD78e56369525C

토큰 기호

[편집](#)

KPT

토큰 소수점

18

커스텀 토큰 추가

토큰 계약 주소에 Deployed Contracts의 주소를 복사해서 넣어준다.

기다리고 있으면 나머지는 자동으로 등록이 된다.

토큰추가하면 토큰이 추가할 수 있게 된다.



토큰 가져오기

이 토큰을 추가할까요?

토큰

잔액



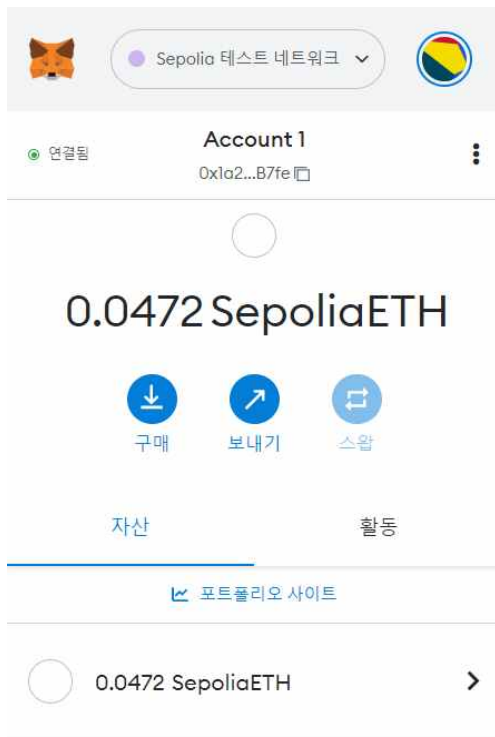
KPT

10000 KPT

토큰 잔액을 알 수 있다.

[뒤로](#)

[토큰 가져오기](#)

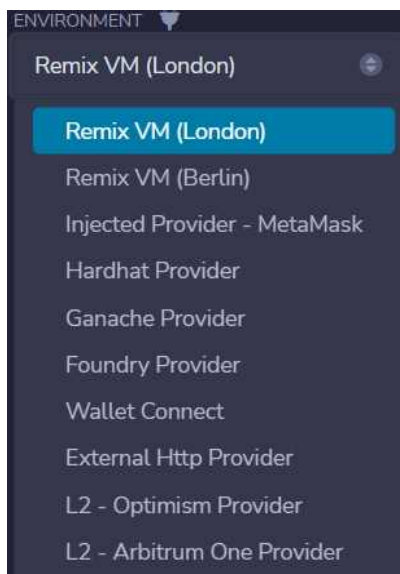


메타마스크로부터 받은 Ether

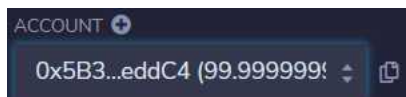
내가 발행한 토큰

을 같이 볼 수 있다.

다시 로컬네트워크에 연결한다.



현재 배포한 계정을 확인해보자.



계정명은 컨트랙트마다 다름을 주의

approve란?

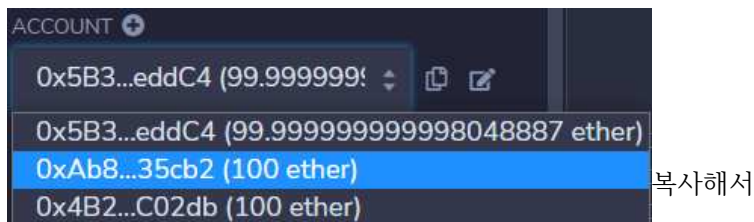
이제 계정1에서 계정2에게 1000토큰을 사용할 수 있는 권한을 부여해보자.

계정2는 계정1로부터 가져와서 계정3에게 토큰을 주자. 주로 구독 서비스에서 사용을 한다.

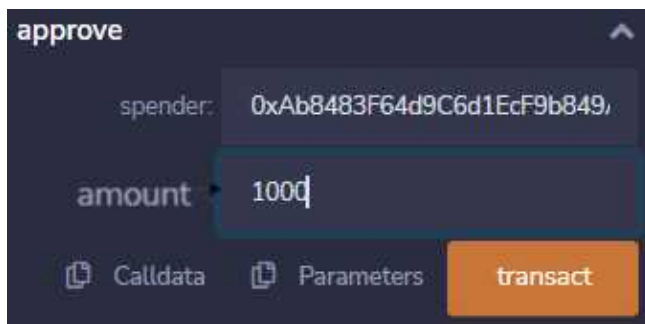
구독 서비스란 내 계좌에서 돈을 빼갈 수 있는 권한을 부여 해주는 것이다,

즉 계정1이 계정2에게 계정1에 있는 토큰을 빼 갈 수 있는 권한을 부여해주는 것이다.

계정1에서 계정2에 approve를 주기 위해 계정2의 주소를 복사해와서 approve에 붙여 넣는다.



approve의 spender에 붙여주기 amount는 1000정도 주도록 한다.



transact를 눌러 실행을 한다,

allowance에서 권한이 주어졌는지 확인을 해보자.



1000토큰만큼을 사용할 수 있는 권한이 부여가 된 것을 확인할 수 있다.


이제 계정3에 부여받은 1000토큰의 일부를 계정3에 주기 위해 transferFrom을 한다,




계정1에서 계정3에 보내는 것이라 sender는 계정1이되어야 하고 recipient는 계정 3이 되어야 한다

A screenshot of a debugger window showing a function call for 'transferFrom'. The function has three arguments: 'sender' with value '0x5B38Da6a701c56854dCfBC', 'recipient' with value '0x4B20993Bc481177ec7E8f571', and 'amount' with value '500'. The 'amount' field is highlighted with a blue selection box. Below the arguments, there are buttons for 'Calldata', 'Parameters', and a red 'transact' button. To the right of the debugger, there is a text explanation in Korean: '계정1' (Account 1) for sender, '계정3' (Account 3) for recipient, and '계정2가 계정1에서 계정3에 보낼 금액을 500' (Account 2 sends 500 to Account 3 from Account 1) for the amount.

실행하기 위해 transact를 누른다.

단, 보내는 사람이 계정2이므로 Account는 계정2가 되어야 한다.

ACCOUNT 

0xAb8...35cb2 (100 ether)			
0x5B3...eddC4 (99.99999999998002661 ether)			
0xAb8...35cb2 (100 ether)			

계정1에 얼마가 남았는지 balanceOf를 확인해보자.

[illegible]

account에 계정1을 넣고 call을하면 500이 차감된 것을 확인할 수 있다,

계정2의 상태도 확인하기 위해 계정2의 주소를 balanceOf의 account에 적어준다.



변화가 없음을 알 수 있다.

그러면 이제 계정3을 확인해보자. 계정2가 계정1의 토큰을 계정3에 주었으므로 증가가 되어야 한다,

decreaseAllowance

spender: 0xAb8483F64d9C6d1EcF9b849,

subtractedValue: 300



 Calldata  Parameters **transact**

계정1이 계정2에서 남은 금액 500중 300만큼 권한을 차감한다.
계정2는 200이 남을 것이다.

allowance

owner: 0x5B38Da6a701c568545dCfcBC

spender: 0xAb8483F64d9C6d1EcF9b849,

 Calldata  Parameters **call**

0: uint256: 200

이더리움에서 NFT발행하기

- Solidity
- Remix
- MetaData
- OpenZeppelin

NFT도 하나의 토큰이다.

ERC-20으로 토큰을 만들었다면 NFT는 ERC-721규약으로 만들게 된다.

ERC-721

- 대체불가능 토큰
- ID별 소유권을 추적한다,
- 디지털 소유권
- 소유권 변경

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/token/ERC721Token.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/token/ERC721.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/token/ERC721BasicToken.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/token/ERC721Basic.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/token/ERC721Receiver.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/math/SafeMath.sol>

<https://github.com/district0x/memefactory-contracts/blob/master/contracts/Utils/AddressUtils.sol>

<https://github.com/OpenZeppelin/openzeppelin-sdk/blob/master/packages/lib/contracts/Initializable.sol>

솔리디티 버전을 pragma solidity ^0.4.24;로 변경한다,
ERC721BasicToken.sol에서 7행을 주석 처리하고 6행의 주석을 푼다.
ERC721Token.sol에서 38행부터 41행의 주석을 푼다.

ERC721Token.sol에 있는

111번 행에서

```
function _setTokenURI(uint256 _tokenId, address _uri) internal {  
    require(exists(_tokenId));  
    tokenURIs[_tokenId] = _uri;  
}
```

에서 address _uri를 string _uri로 변경

33번 꺾개의

mapping(uint256 => address) internal tokenURIs;를
mapping(uint256 => string) internal tokenURIs;로 변경

70번행에

```
function tokenURI(uint256 _tokenId) public view returns (address) {  
    require(exists(_tokenId));  
    return tokenURIs[_tokenId];  
}  
}에서 반환 형을 string으로 변환  
function tokenURI(uint256 _tokenId) public view returns (string) {}
```

ERC721.sol에서

22행을

function tokenURI(uint256 _tokenId) public view returns (string); 로 변경