solidity-기본-3

태그	
날짜	@2023년 6월 19일

view : function 밖의 변수들을 읽을 수 있으나 변경 불가능

pure : function 밖의 변수들을 읽지 못하고, 변경도 불가능

view와 pure 둘다 명시 안할 때 : function 밖의 변수들을 읽어서, 변경해야함

```
1. view

uint256 public a = 1;
function read_a() public view returns(uint256) {
    return a+2;
  }
}

배포 후
a - 0 : uint256 : 1
read_a - 0 : uint256 : 3

-------------------------------------------------------------------------------

2. pure
function read_b() public pure returns(uint256) {
    uint256 b = 1;
    return 4 + 2 + b;
  }
}

배포 후
a - 0 : uint256 : 1
read_a - 0 : uint256 : 3
read_b - 0 : uint256 : 7

-------------------------------------------------------------------------------
3. view pure 사용 안할 때

function read_c() public returns(uint256) {
    a = 13;
    return a;
  }
}
```

```
배포 후
read_c : return을 통해 로그로 출력
```

---

string

- storage : 대부분의 변수, 함수들이 저장되며, 영속적으로 저장이되어 가스 비용이 비싸다.
- memory : 함수의 파라미터, 리턴값, 레퍼런스 타입이 주로 저장이 된다. 그러나, storage처럼 영속적이지 않고, 함수 내에서만 유효하기에 storage보다 가스 비용이 싸다.
- colldata : 주로 external function 의 파라미터에서 사용된다.
- stack : EVM(ethereum virtual machine) 에서 stack data를 관리할 때 쓰는 영역인데 1024Mb 제한적이다.

함수 밖의 변수들, 함수 그 자체가 storage에 저장된다.

```
function -string

function get_string(string memory _str) public pure returns(string memory){
  return _str;
}
```

---

instance

A와 B 컨트랙을 연결할 때 주로 사용한다.

```
contract A{
    uint256 public a = 5;
    function change(uint256 _value) public{
        a = _value;
    }
}

contract B{
    A instance = new A(); // instance 생성

    function get_a() publice view returns(uint256){
      return instance.a();
    }
    function change_A(uint256 _value) public {
        instance.change(uint256 _value);
    }

}
```

---

constructor (생성자)

: 변수의 값을 초기화 할때 주로 사용한다.

```
contract A{
    string public name;
    uint256 public age;

    constructor(string memory _name, uint256 _age{
        name = _name;
        age = _age;
    }

    function change(string memory _name, uint256 _age) public{
        name = _name;
        age = _age;
    }
}

contract B{
    A instance = new A("Alice", 52);

    function change(string memory _name, uint256 _age){
        name = _name;
```

```
        age = _age;
    }

    function get() public view returns(string memory, uint256) {
        return (instance.name(), instance.age());
    }
}
```

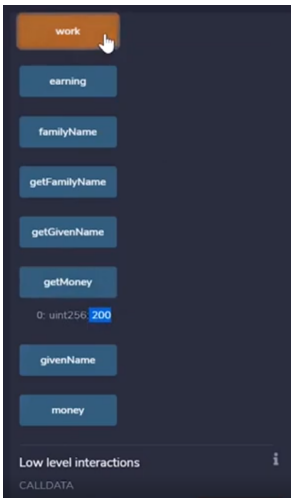
상속

```
contract Father{
  string public familyName = "Kim";
  string public givenName = "Jung";
  uint256 public money = 100;

  constructor(string memory _givenName) public{
      givenName = _givenName;
  }

  function getFamilyName() view public returns(string memory){
      return familyName;
  }
  function getGivenName() view public returns(string memory){
    return givenName;
  }

  function getMoney() view public returns(uint256){
    return money;
  }
}

contract Son is Father("James"){ // constructor 가 추가되어 이름까지 설정했을때


}
```

overriding

: 간단하게 덮어씌우기 라고 생각하면 쉽다.

```
contract Father{
  string public familyName = "Kim";
  string public givenName = "Jung";
  uint256 public money = 100;

  constructor(string memory _givenName) public{
      givenName = _givenName;
  }

  function getFamilyName() view public returns(string memory){
      return familyName;
  }
  function getGivenName() view public returns(string memory){
    return givenName;
  }

  function getMoney() view virtual public returns(uint256){ // virtual 명시
    return money;
  }
}

contract Son is Father{

  constructor() Father("Jame"){

  }

  uint256 public earning = 0;
  function work() public{
    earning += 100;
  }

  function getMoney() view override public returns(uint256){ // fn이름도 같아야한다.
    //override 명시
    return money+earning;
  }

}
```



2개 이상의 스타트 컨트랙 상속

```
contract Father{
  uint256 public fatherMoney = 100;
  function getFatherName() public pure returns(string memory){
    return "KimJung"
  }

  function getMoney() public virtual view returns(uint256){
    return fatherMoney;
  }
}

contract Mother{
  uint256 public motherMoney = 500;
  function getMotherName() public pure returns(string memory){
    return "LeeSol"
  }

  function getMoney() public virtual view returns(uint256){
    return motherMoney;
  }
}

// 두개 이상의 스마트 컨트랙을 상속할 때 같은 이름의 fn이 있다면 override 해주어야 한다.
contract Son is Father, Mother{
    function getMoney() public view override(Father, Mother) returns(uint256){
```

```
        return fatherMoney + motherMoney;
    }
}
```

---

event

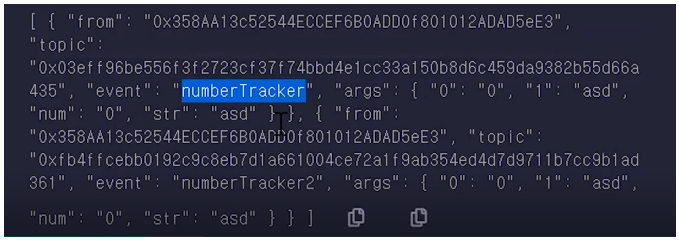
```
contract lec13{

  event info(string name, uint256 money);

  function sendMoeny() public{
    emit info("LeeMinHo", 1000);
  }

}
```

---

indexed

: event의 키워드, event내에서만 사용, 특정한 이벤트의 값들을 가져올 때 사용

```
contract lec14{
    event numberTracker(uint256 rum, string str);
    event numberTractker2(uint256 indexed num, string str);
//uint256 indexed num 특정 이벤트의 값을 가져올 수 있음

    uint256 num = 0;
    function PushEvent(string memory _str) public{
      emit numberTracker(num, _str);
      emit numberTracker2(num, _str);
      num ++;
    }
}
```



출력만 한 상태

vscode

num: [2,1] - 2나 1일때 가져오도록

fromBlock : 1, - 첫번째부터

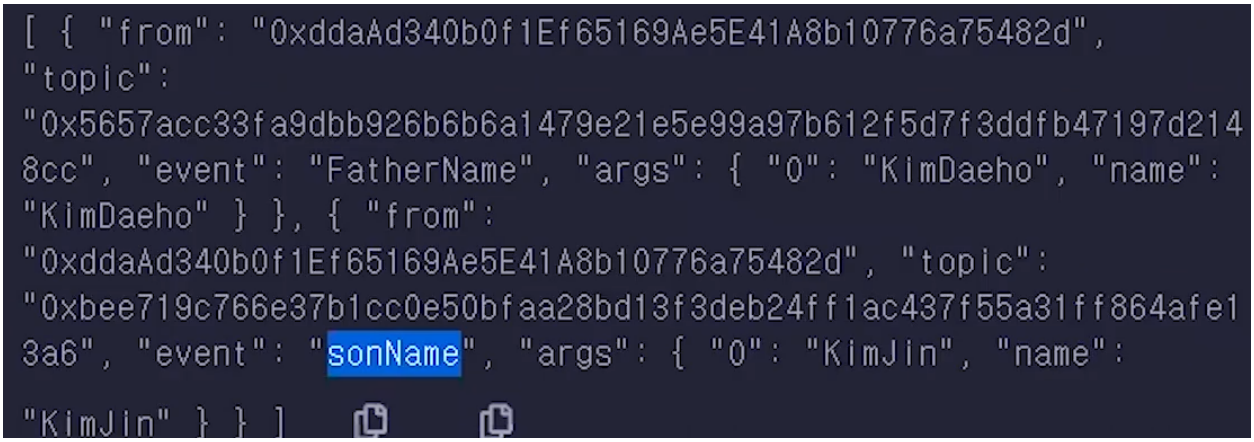
---

super

: 함수를 overriding 할때 사용

```
contract Father{
    event FatherName(string name);
    function who() public virtual{
      emit FatherName("LeeMinHo");
    }
}

contract Son is Father{
  event sonName(string name);
  function who() public override{
    super.who();
    emit sonName("LeeHongSi");
```

```
  }
}
```



상속의 순서

```
contract Father {
    event FatherName(string name);
    function who() public virtual{
      emit FatherName("KimDaeHo");
    }
}

contract Mother {
    event MotherName(string name);
    function who() public virtual{
      emit MotherName("LeeSol");
    }
}

contract Son is Father, Mother{ // << 여기서 뒤에 것이 최신, 상속시 제일 오른쪽부터 최신이다.
    function who() public override (Father, Mother){
      super.who(); // mother와 father중 어떤 who를 가져오게 될까?
      // Mother의 who를 가져오게 된다.
      // 이유는 Mother의 스마트 컨트렉트가 가장 최신이기 때문이다.
    }
}
```