```
pragma solidity ^0.5.8:

contract Greeter
{
    [          ]  // address 변수 creator 생성
    [          ]  // string을 담을 변수 greeting 생성

    constructor(string memory _greeting) public {  //생성자는 인자로 _greeting 입력

        [          ]  // creator에는 컨트랙트를 배포한 사람의 주소를 대입
        [          ]  // greeting에는 생성자 파라미터로 입력받은 string 값 대입
    }

    function greet() public view returns (string memory)
    {
        [          ]  // greeting 변수를 반환
    }


    function setGreeting (string memory _newgreeting ) public
    {
        [          ]  // greeting 변수에 새로운 파라미터를 대입
    }


}
```

```
pragma solidity ^0.5.8;

contract Lottery{
    [                    ] // address 변수 manager를 public으로
    [                    ] // address 배열 players를 payable과 public으로

    constructor() public{ // 생성자 함수
        [                ] //manager는 컨트랙트를 생성하는 사람의 주소
    }

    function enter() public payable{
        [                ]
        //컨트랙트에 담아 보낸 이더 값이 0.1 ether 이상 인지 아닌지 확인
        players.push(msg.sender);
    }

    function random() private view returns (uint){
        return uint(keccak256(abi.encodePacked(now, msg.sender, players.length)));
    }

    function pickWinner() public restricted{
        [                ]
    //uint형 변수 index에는 random() 함수의 return 값을 players.length로 나눈 나머지
        players[index].transfer(address(this).balance);
        show_players=
    }

    function getPlayers() public view returns([              ]){ //address 배열 반환
        return show_players;
    }

    modifier restricted{
        [                ] //manager가 함수를 호출한 사람이 맞는지 확인
        [    ]  //맞으면 다시 pickWinner함수로 돌아가서 나머지 코드 수행
    }
}
```

```
pragma solidity ^0.5.8;

contract SimpleAuction{

    
//address 유형의 beneficiary 변수를 public, payable으로 선언
    
//uint 유형의 auctionEnd 변수를 public으로 선언

/* 경매의 현재 상태를 나타냄*/
    
//address 유형의 highestBidder 변수 public으로 선언
  (highestBidder는 현재 경매에서 최고가를 제시한 사람)

    
//uint 유형의 highestBid 변수를 public으로 선언
//(highestBid는 현재 경매에서 제시된 최고가)
    
//mapping 유형의 pendingReturns 선언
//(경매에서 진 나머지 입찰자들의 자발적 인출을 하기 위함)
      bool ended;

/*어떤 변화에 따라 발생하는 이벤트 선언*/
    
//HighestBidIncreased 이벤트는 인자로 address유형의 bidder, uint유형의 amount 받음
    
//AuctionEnded 이벤트는 인자로 address유형의 winner, uint 유형의 amount 받음
```

```
/*함수 선언*/

    constructor (uint _biddingTime, address payable _beneficiary) public{
        
//address유형의 _beneficiary를 변수 beneficiary에 대입
        
//auctionEnd(경매가 끝나는시간)= now(현재 블록의 타임 스탬프) + _biddingTime
    }


    {
//bid 함수는 public으로 선언되어 외부에서 접근 가능
//payable modifier를 사용하여 함수 호출 시 Ether를 보낼 수 있다
        
//경매가 끝나면 call을 되돌리도록 require에 now와 aunctionEnd 비교하여 판정
        
//경매 입찰가가 높지 않으면 돈을 되돌려 준다
//require에 입찰가(msg.value)와 현재 최고액(highestBid)를 비교해 반정

        if(highestBid!=0){
            pendingReturns[highestBidder]+=highestBid;
//경매에서 진 나머지 입찰자들의 자발적 인출을 하기 위해 입찰자들의 계정과 던졌던 금액
//을 mapping한다.
        }
        
//새로운 입찰가를 제시한 사람을 교체. bid함수를 호출한 사람
        
//새로운 입찰가를 기존 입찰가와 교체. bid함수를 호출하면서 보내는 이더의 양
        emit HighestBidIncreased(msg.sender, msg.value);
//2018.03.08. v0.4.21에 emit 키워드 도입
//emit (이벤트 이름)으로 호출 가능
//HighestBidIncreased 이벤트 호출
//이벤트가 호출되면 EVM이 블록체인에 트랜잭션 로그를 기록하고 이 로그를
//FrontEnd(Web3.js)에서 사용가능
    }
```

```
/*withdraw 함수는 pendingReturns를 스스로 인출하기 위한 함수*/
    function withdraw() public returns (bool){
        
//uint 유형의 amount 변수에 pendingReturns에 저장된 값을 할당
//저장된 값은 Key가 msg.sender인 값이다
        if(amount>0){
            pendingReturns[msg.sender]=0;
//받는 사람이 'send'가 반환되기 전에 수신 호출의 일부로서 이 함수를 다시 호출할 수 있
기 때문에 이것을 0으로 설정하는 것이 중요!!
                if(!msg.sender.send(amount)){
//돈을 보냈는데 오류가났다
                        pendingReturns[msg.sender]=amount;
                        return false;
// 주어진 양의 wei를 address에 보내고 실패하면 false를 반환
                        }
                }
         }
        return true;

}



   function aunctionEnd() public{
        
//경매가 아직 끝나지 않았음을 판정 (now 사용)
        require(!ended); //이 함수가 이미 호출되었는지 판정

        ended=true;
        emit AuctionEnded(highestBidder, highestBid);
        
//beneficiary의 address로 highestBid(단위 : Wei) 보냄
        }
```