3 장. 스마트 컨트랙트 작성

1. 솔리디티 소스 파일
   - 소스 파일 내에서 pragma solidity 지시자를 사용해 코드가 작성된 컴파일러 버전 언급 가능 => **pragma solidity ^0.4.2;**
2. 데이터 위치
   - 모든 프로그래밍 언어들은 변수 값을 메모리에 저장하는 반면 솔리디티에서는 변수가 컨텍스트에 따라 메모리 또는 파일 시스템에 저장됨.
   - 함수 매개변수는 메모리, 로컬 변수 및 상태 변수는 스토리지에 저장
   - 스토리지 변수와 메모리 변수 사이의 할당은 언제나 독립적 사본 생성
3. 다양한 데이터 유형
   - bool , uint8, uint16, uint24, … , uint256(= uint) , int256(= int)
   - 실수형 : ufixed0x256(= ufixed) , fixed
   - address : 16 진수를 할당해 최대 20 바이트 값 저장(이더리움 주소 저장)
     - balance : 계정의 잔액 확인
     - send : 주소로 이더를 송금하는데 사용 => send 메소드를 호출하는 컨트랙트에서 wei 가 차감
4. 배열
   - 바이트 배열의 유형 : bytes1(= byte), bytes2, bytes3, … , bytes32
   - 배열의 크기를 알아낼 수 있는 length 속성 갖고 있음
     - 이를 이용해 배열 크기 변경 가능(동적 배열만)
   - 동적 배열의 설정되지 않은 인덱스에 접근하려고 하면 예외 발생
5. 문자열
   - bytes(원시 문자열 만듦)와 string(UTF-8 문자열을 만듦)을 이용
   - 문자열의 길이는 언제나 동적
6. delete 연산자
   - 어떤 변수라도 기본값(모든 비트가 0)으로 재설정하기 위해 사용될 수 있음.
   - 동적 배열에 적용 -> 모든 요소 지우고 길이가 0
   - 정적 배열에 적용 -> 모든 인덱스 재설정(특정 인덱스에만 사용 가능)

7. 기본 유형 간의 변환
   - 배열, 문자열, 구조체, 열거형, 맵 이외의 모든 것을 기본 유형이라 부름
   - 의미상 합당, 정보의 유실이 없는 경우, 값 유형 간의 묵시적 변환 가능
   - 예) uint32 a = 0x12345678; => uint16 b = uint16(a);
     => b 는 이제 0x5678 일 것이다.
8. 변수 선언에 var 키워드를 사용한다.
   - 변수 유형은 첫 번째 할당값에 따름
   - 예) int256 x = 12; => var y = x; // y 의 유형은 int256
9. 제어 구조
   - if, else, while, for, break, continue, return, ?을 지원한다.
10. 함수 변경자
   - 변경자 내부에서의 리턴은 즉시 함수를 벗어난다
   - 호출자의 코드 실행이 완료된 이후 기존 변경자의 _; 뒤에 오는 코드가 실행된다.
11. 폴백 함수(fallback function)
   - 한 개의 이름 없는 함수(인자를 가질 수 없고, 아무것도 리턴할 수 없음)
   - 어떠한 함수 호출 없이 컨트랙트가 이더를 수신한 경우, 즉 트랜잭션이 컨트랙트에 이더를 송금했으나 어떠한 메소드도 호출하지 않은 경우 실행됨
   - 가능한 한 싸게 만들어야함.
   - 폴백 함수가 지정되어 있지 않은 컨트랙트는 이더를 받으면 예외가 발생하여 이더를 돌려보내므로, 이더를 받고 싶은 컨트랙트의 경우 폴백 함수를 구현해야 한다.
12. 상속
   - 코드를 카피하는 방식으로 다중상속 지원
   - Contract 가 다수의 다른 Contract 를 상속하더라도 블록체인에는 하나의 Contract 만 생성되며 부모의 코드가 최종 Contract 에 복사됨
13. 추상 컨트랙트
   - 함수의 구현 대신 프로토타입만 갖고 있는 컨트랙트. => 컴파일 X
   - 추상 함수를 상속 받았고 모든 미구현 함수를 재정의해서 구현하지 않았다면 그 자체도 추상이다.

- 이미 배포된 컨트랙트를 참조하고 함수를 호출할 때 유용하다.

14. 라이브러리
    - 컨트랙트와 유사하지만, 특정 주소에 한 번 배포되고 코드가 다양한 컨트랙트에서 재사용되는 것이 목적. 만약 라이브러리 함수가 호출되면 그 코드가 호출한 컨트랙트의 컨텍스트에서 실행됨
    - 상속을 지원하지 않고 이더를 받을 수도 없음
    - 구조체와 열거형 포함 가능
    - 일부 공통된 소스 코드를 가진 많은 계약이 있는 경우 공통된 코드를 라이브러리로 배포 => 가스는 컨트랙트의 크기에 의존적이므로 가스 절약
    - 데이터 유형에 멤버 함수를 추가하는 데 사용 가능