1. 이더리움과 비트코인 비교 : 스마트 계약에 결제내용 포함됨
   1. 거래기반의 작업방식 : UTXO 모형-> 비트코인
   2. 계정 기반 : 좀더 직관적이고 이해하기 쉽게 되어있음
2. 이더리움 account
   1. 비트코인
      1. UTXO는 우리가 가지는 지폐라고 보면 된다. (사용자에게는 불편하지만 이중지불문제를 해결하기 쉽다.)
      2. 거래기반의 방식을 사용 -> 더 큰 돈을 주고 거스름돈을 받는 송금구조
      3. Hash 작업증명 수행
      4. 블록생성시간이 10분으로 많은 시간과 에너지가 소비된다. -> fork가 많이 일어나지 않는다.
      5. 이중지불 문제와 거래 기록 생성이 비교적 쉽다.
   2. 이더리움 : 잔액기반의 거래방식 -> 우리가 볼떄는 더 직관적이여서 이해가 쉬움
      1. Account 기반의 거래방식
      2. 다 합처진 상태로 보임
      3. Ethash로 작업증명 수행
      4. 빠른게 장점이지만, 메모리공간이 필요함
      5. 작업증명(pow)에서 지분증명(pos)으로 넘어가는 중 이다.
      6. 블록생성기간이 30초정도로 빠르다.
      7. 블록 생성기간이 빠른데신 fork가 많이 일어난다.
      8. 직관적인 거래내용으로 볼 수 있다. 따라서 계정 간 잔액 조회 및 송금이 편리하다.
      9. 블록의 생성 보다는 거래내용이 갱신되는 것으로 인식됨
3. 이더리움 계정구조
   1. 소유계정과 계약계정으로 나뉜다.
      1. 소유계정 : 개인키로 제어하여 코드를 저장할 수 없다.
      2. 계약계정 : 계약 증명이 있는 계정 (프로그램의 일종으로, 스마트 컨트랙트 코드에 의해 제어됨)
   2. EOA (외부계정): 개인키로 제어, smartcode를 저장할 수 없음
      1. ca를 활성화하기 위해서는 ether가 필요하게 된다.
   3. CA : 계약서들만 이 계정안에 넣어둠.
      1. 자신 스스로는 활성화될 수 없다. 거래에 대한 응답으로만 실행된다.
   4. 따라서 계약관련에 대한 계정과 소유한 금액(Ether) 에 대한 계정을 나눠서 가진다.
   5. Eoa와 Eoa : 금액을 왔다갔다 할 때 사용
   6. Eoa와 ca: 계약관련으로 일을 수행
   7. 스마트 컨트렉트를 활성화하기 위해서는 eoa가 필요하다. 따라서 ca는 자신혼자 거래를 실행할 수 없다
   8. eoa로부터
4. 상태트리 : 계정정보와 이더 잔액 정보를 가지는 것
   1. Nonce : 이중거래를 방지(얼마의 거래가 발생한지 알 수 있는 것) -> 동일한 값이 나오면 동일지불인 것이다. (중요함)
   2. 블록 외부에 보관하고, 블록에는 상태트리의 루트 노드값만 저장
   3. storageRoot는 ca를 가지고있다.
5. 스마트 계약 : 자동적으로 기능을 수행하는 것 (자율 에이전트)
   1. 이더리움 버츄얼머신에 의해 자율 에이전트가 수행됨
   2. 외부세계 : 이더리움 네트워크 안에서의 반응
   3. 계약상태와 이더
6. Pow: 작업증명
   1. EVM 코드를 실행하면서 해쉬값으로 인해서
   2. 블록을 만들기만해도 보상을 재공함
7. 이더리움 버츄얼 머신 : 블록들의 검증절차로 실행되는 것
   1. 중앙 컨트롤러가 필요없게 됨 :
   2. 연산을 수행하는 곳
   3. 검증을 하기위햏서EVM을 실행한다.
   4. 블록체인에 스마트컨트랙트가 있기 때문에 거래를 마음대로 바꾸면 블록체인 모두를 바꿔야한다.
8. 컨트렉트 코드
   1. Solidity 코드로 만들어짐
   2. EVM이 컨트렉트 코드를 컴퓨터언어로 바꿔주는 역할을 한다.
   3. Op code가 EVM안에서 실행된다.
9. EVM 안에 컨트렉트 코드 안에 무한루프가 있으면 안됨
   1. Gas를 적용한다 : 스마트 컨트렉트가 실행될 때마다 필요한 비용( . 특정 명령어마다 gas를 소비. )
   2. 따라서 거래 또는 계약할 때 수수료(gas) 가 요구된다. -> 따라서 계약을 실행하기 위해서는 계속해서 Gas가 필요하기 때문에 무한루프를 하더라도 끝나버린다.
   3. EOA 에서의 value는 ether를 뜻함
10. 거래 수수료 = gas(이더리움 안에서 bytecode의 한가지 명령어를 실행하는데 필요한 것 )
    1. 명령어마다 지불하는 gas가 다르다.
    2. 블록을 생성할 때 수수료를 지급해야함
    3. 연산을 할 때마다 지불
    4. EVM에 연산을 시키게 되는데, 이때 수수료가 발생한다.
    5. Gas의 사용이유는 무한루프를 막기위해 사용된다.
    6. Solidity 특징 -> 튜링 완전 : 무한 반복이 가능한 것 -> 이것을 해결하기 위해 gas를 사용
    7. 필요한 gas 뿐 아니라 충분한 gas가 있는지 확인한 후 거래를 수행한다.
    8. Ether = 10의 18승
    9. 발신자가 가스 한도와 가스 가격을 설정
       1. 최대수수료 = gas price \* gas limit => price 낮게 설정해 놓으면 채굴자들이 잘 안 하려고 해서 오래 걸리게 된다. (250을 보낸다고 하면, 250보다 많은 gas를 보내야 한다. 명령어를 수행하며 gas를 소비한다. 그리고 모두 수행한 후 남은 gas를 다시 받는다.
11. Sender 송신자, operation: 스마트 코드(계속 gas가 소모됨) ,
    1. 전체금액에서 gas비용이 지불되고, Receiver에게 돈을 보낸다. 이때 남은 돈은 다시 송신자에게 간다.
    2. 트랜잭션 실행을 위한 gas가 충분하지 않을 경우 상태 변경이 취소되고, 이때 사용된 gas는 환불되지 않는다.
    3. 튜링 완전의 특징을 가진다. : 무한루프에 빠질 수 있다. 이것을 막기 위해 gas 비용을 설정한다.
    4. Gas는 명령어를 수행하는 값으로 보기 때문에 환불해주지 않는다.
12. 토큰 시스템 구현
    1. 질문에 따라 자문금을 지불한다.
13. Ethash
    1. 메모리를 크게 구성하여, 비트 코인에서hash값을 구하는 것과 동일하지만, 이런 연산을 연산만 하는 장치를 사용하지 못하게 했다.
    2. 메모리가 크고, 연산을 빨리 할 수 있는 장비가 필요하다. -> 그래픽카드를 요구한다.
14. Uncle block
    1. 블록크기가 작고 빨리 만들어진다. -> 전파하는 시간 보다 블록을 생성하는 시간이 더 짧은 경우가 많이 발생하게 된다(fork 발생)
    2. 비트 코인에서는 분기 되어있는 block들에 대하서는 비용을 제공하지 않음
    3. 이더리움에서는 분기 되어있는 block이여도 보상을 지불한다. ( 최장길이라고 하는 룰을 선택하기 보다는, 무게를 중시하는 ghost 프로토콜을 사용하여 블록을 생성한다. -> fork가 난 블록도 공격하는 대상을 저지하는데 기여하므로 비용을 지불한다. )
    4. Pos
       1. ether라는 금액을 가지고 있는 만큼(토큰의 수와 보유기간에 따라) 블록생성 권한에 대해 우선권을 부여한다.
       2. 따라서 많이 가지고 있을수록 더 많은 블록을 생성할 수 있다. -> pow보다 적은 전력이 요구됨
       3. 단점 : 중앙화가 될 수 있다. (빈익빈 부익부가 발생할 수 있다.)
       4. 검증에 대한 서명도 블록에 포함된다.
       5. 장점 : 낮은 전력소비, 공격 확률 감소(51%의 ether를 가지고 있어야 하는데, 거의 불가능),
       6. 단점 : 블록생성시 두개의 분기가 있을 때, 거의 같다면 선택이 어려워진다. (Nothing at stake problem)
          1. 블록을 검증하는 데 있어서, 보증금을 넣고 검증을 하고 다시 돌려받는 방법으로 해결
          2. 이때, 두개의 검증부분 중 한곳에만 보증금을 넣을 수 있고 검증할 수 있다.