

# Vyper (Smart Contract) 101 class

Robbie OH



# Speaker: Robbie OH (오영택)

- HAECHI Labs SW engineer
- Decipher co-founder
- 이드콘 준비위원회

[robbieinertia@gmail.com](mailto:robbieinertia@gmail.com)  
010-3264-3011



# Dapp의 특징

- 오픈소스 기반
- 암호화폐라는 새로운 가치 전송 수단 활용

# 왜 Dapp을 만드는가?

- 미리 약속한 대로 코드 실행이 보장됨
- 퍼블릭 블록체인에 데이터 저장 -> 사후 검증 가능
- 암호화폐라는 새로운 가치 교환 수단을 서비스에 integration할 수 있음.

# (아직) Dapp이 어울리지 않는 예제

- 상호작용이 잣은 앱들
  - 속도 문제보다는 스마트컨트랙트 특성상 당사자가 매번 직접 실행 해야 해서 UX/UI 경험이 안좋다
  - 매 행위마다 수수료 발생 -> fee delegation등의 제안됨
- 로직이 너무 복잡한 앱들
  - 허용된 계산의 최대치가 있다(gas)
  - 여러 offchain solution들이 현재 연구중

# (아직) Dapp이 어울리지 않는 예제

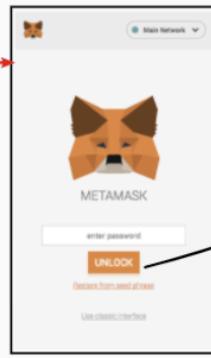
- 상호작용이 잣은 앱들
  - 속도 문제보다는 스마트컨트랙트 특성상 당사자가 매번 직접 실행 해야 해서 UX/UI 경험이 안좋다
  - 매 행위마다 수수료 발생 -> **fee delegation**등의 제안됨
- 로직이 너무 복잡한 앱들
  - 허용된 계산의 최대치가 있다(gas)
  - 여러 offchain solution들이 현재 연구중

# (아직) Dapp이 어울리지 않는 예제

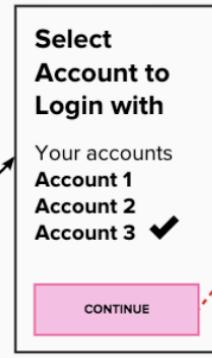
Visit Dapp



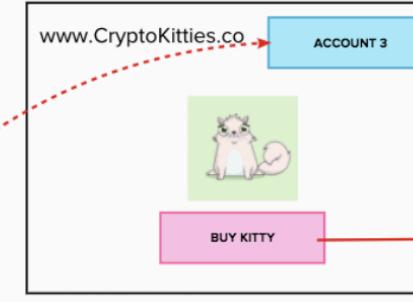
Unlock Popup



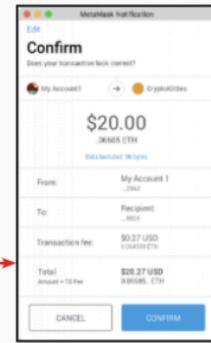
Account Selection Popup



Logged in, see public address



Confirm TX Popup



This works the same way

Clicking "login" from a Dapp would trigger an unlock metamask popup. This removes the extra step of clicking the extension to log in, which would remove the added friction of looking for the extension on the top right corner of your browser.

Another option here might be to open the extension in full screen mode so user can verify the real MM extension is requesting their password, vs a fishing site.

Is fishing for MetaMask passwords really a concern? Seedwords and private keys are a bigger concern.

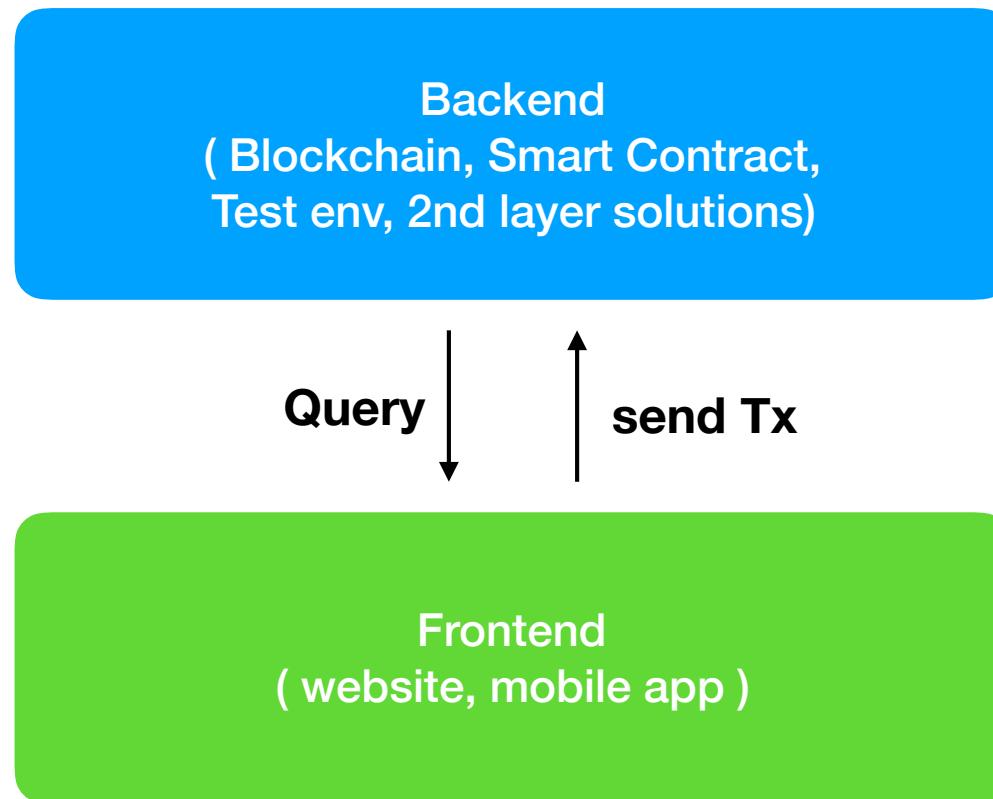
Account selection screen. Right now there's no way to change the account I'm logging in with. So let's add an additional step so the user can select which account to log in with a dapp with.

Can the Dapp auto-detect which is the right account to log in with? That would be cool, so the user doesn't have to choose. If that's the goal of this new feature, we'll need to figure out some additional UX around how to change the ID should the user want to use a different account on a website. Say I have two IDs that I use on CryptoKitties.

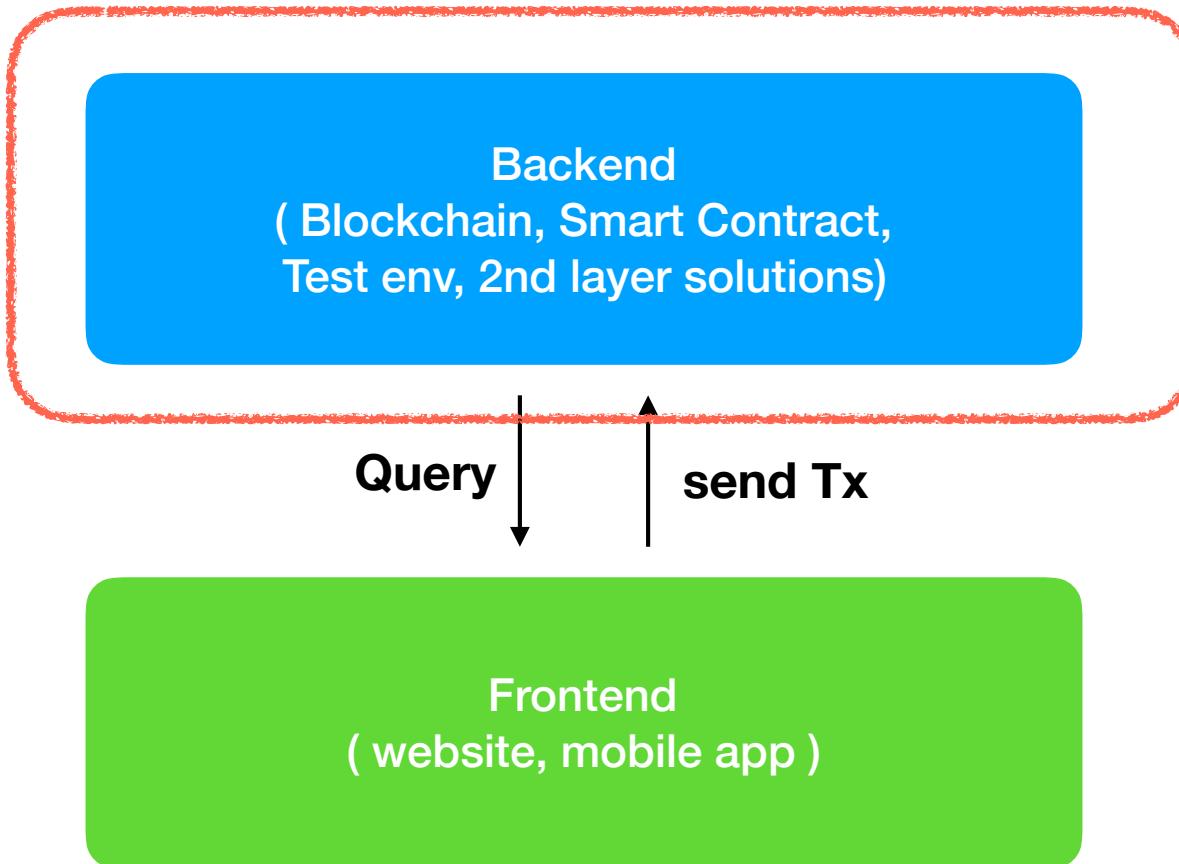
# (아직) Dapp이 어울리지 않는 예제

- 상호작용이 잦은 앱들
  - 속도 문제보다는 스마트컨트랙트 특성상 당사자가 매번 직접 실행 해야 해서 UX/UI 경험이 안좋다
  - 매 행위마다 수수료 발생 -> fee delegation등의 제안됨
- 로직이 너무 복잡한 앱들
  - 허용된 계산의 최대치가 있다(gas)
  - 여러 offchain solution들이 현재 연구중

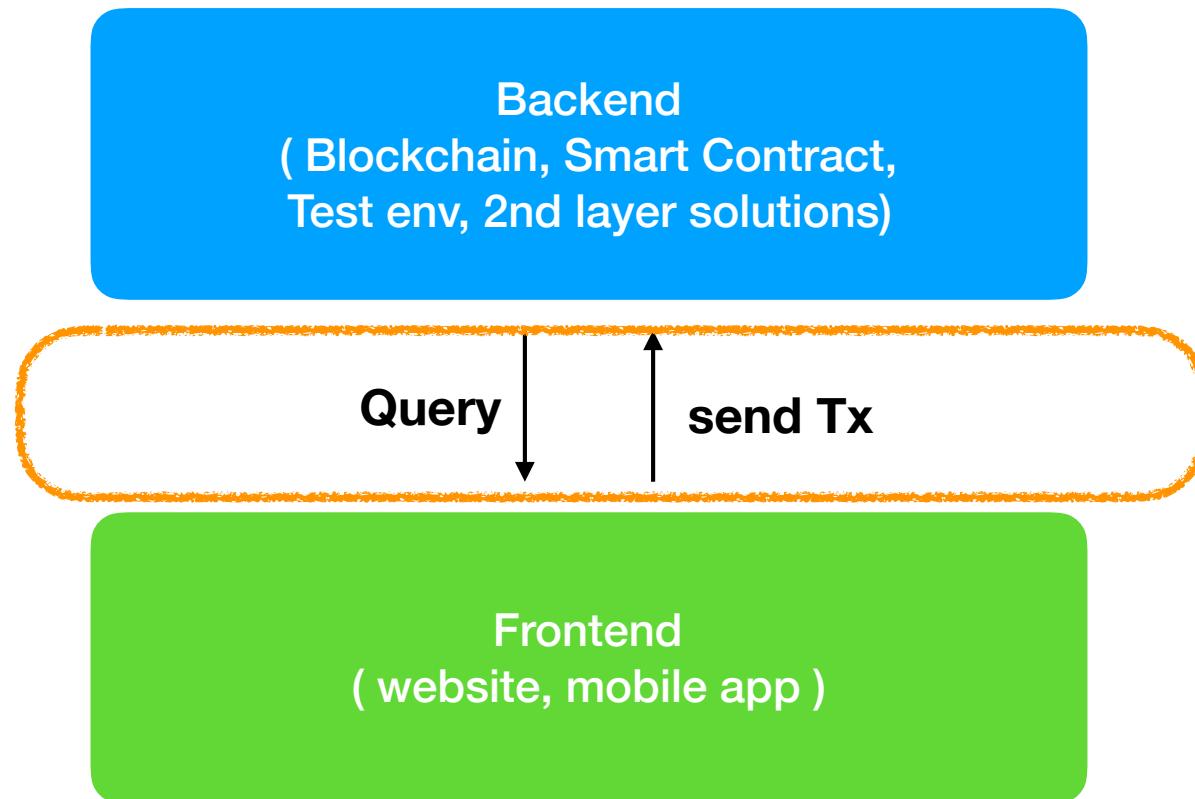
# Ethereum Dapp 개발환경 이해 하기



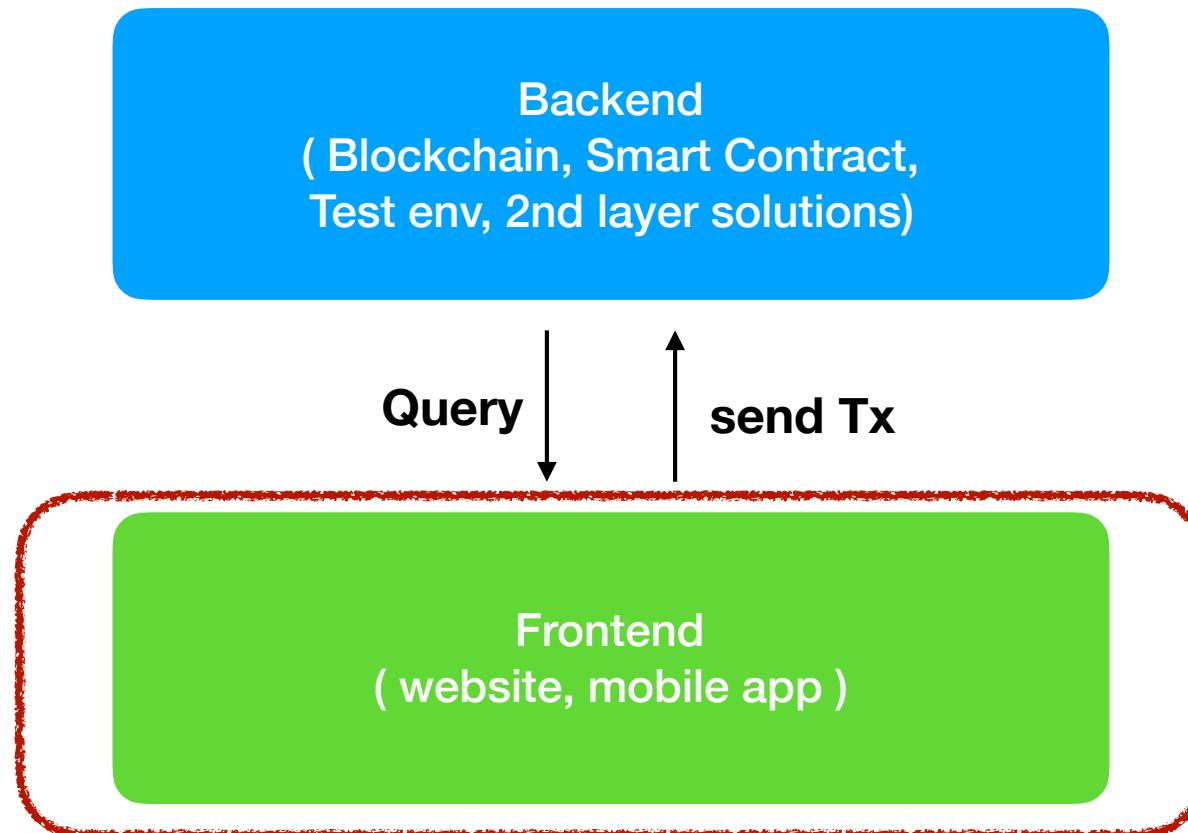
# Ethereum Dapp 개발환경 이해 하기



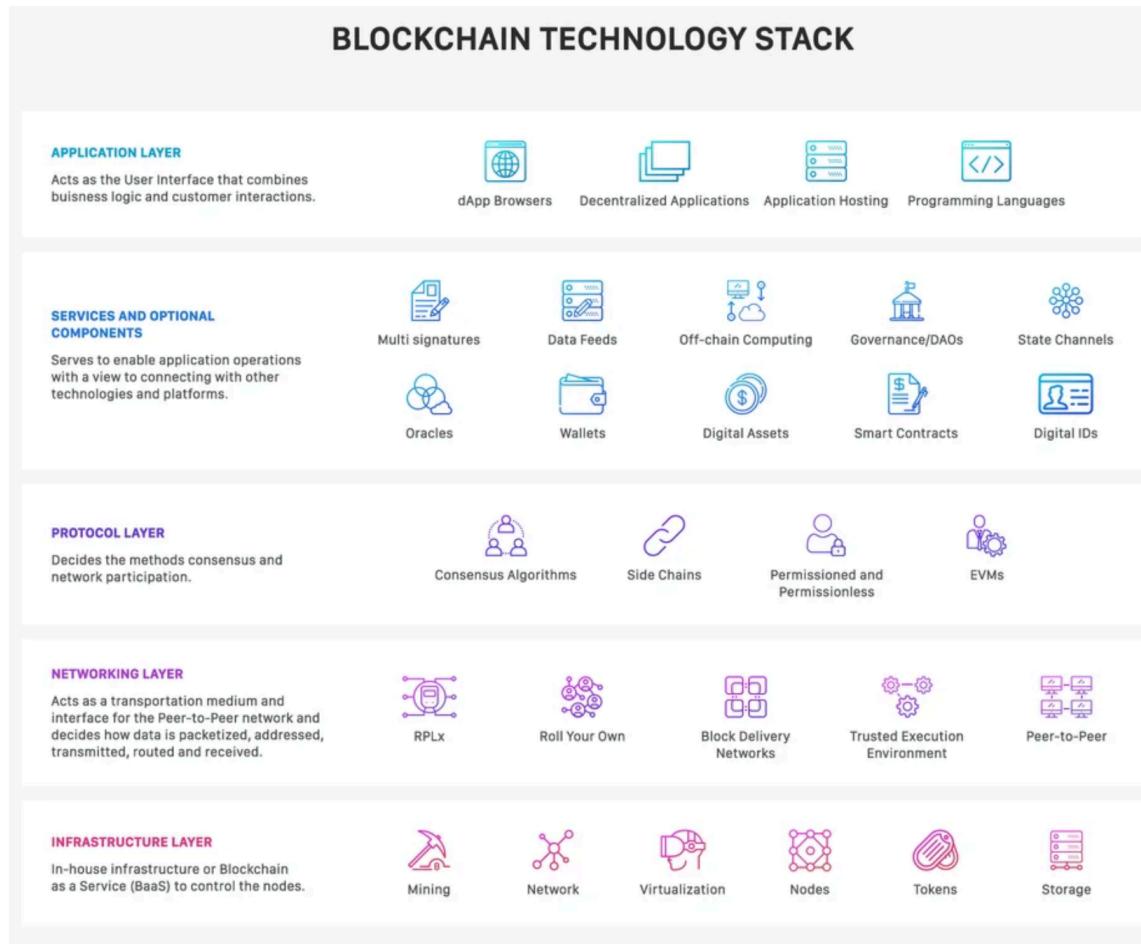
# Ethereum Dapp 개발환경 이해 하기



# Ethereum Dapp 개발환경 이해 하기



# Ethereum Dapp 개발환경 이해하기



[https://www.reddit.com/r/BlockChain/comments/9slpmf/infographic\\_blockchain\\_technology\\_stack/](https://www.reddit.com/r/BlockChain/comments/9slpmf/infographic_blockchain_technology_stack/)

# What is Programming Language?

- 컴퓨터: 전기 신호( on, off/ 0, 1)의 조합으로 일반적인 연산을 하는 기계
- 기계어: 전기 신호와 1:1 매칭 되는 언어
- 프로그래밍 언어: (좁은 의미) 사람이 이해할 수 있는 수준의 문법을 컴퓨터가 이해할 수 있는 기계어로 쉽게 변환이 가능하도록 만든 체계. (문법, 번역기, 실행기 등등)
  - 프로그래밍 언어의 목표: 쉬운 언어 -> 기계어

# What is Programming Language?

- 이더리움: 가상의 컴퓨터(virtual machine) 모델 이용
- 가상의 컴퓨터란?
  - CPU등 하드웨어와 무관한 기계어로 동작하는 가상의 기계
  - “표준화” 호환성을 확보하기에 좋음
- 이더리움도 EVM이라는 가상의 컴퓨터에서 실행 가능한 기계어가 있음.

# What is Programming Language?

- 이더리움에서 언어: solidity, vyper, ...
- **목표:** EVM bytecode로 번역 가능하면서 사람이 쉽게 읽을 수 있도록 편한 개발환경을 제공!

# Vyper History

- Serpent 가 deprecated되면서 탄생(문법은 유사)
- Serpent가 못했던 것을 보안
  - type checking
  - high level structure
  - check safety

# Vyper vs Solidity

- Solidity
  - 개발 편의성, 재사용성 초점
  - javascript 문법 차용
  - 풍부한 개발 인프라
- Vyper
  - 보안, 코드 가독성 중심
  - python 문법 차용
  - (아직은) 부족한 개발 인프라

# Vyper Features

- 감사 가능성과 가독성이 재사용성과 프로그램 편의성보다 높다
- modifier 제거
- infinite length 를 가지는 자료구조 제거(except map)
- function scope(public, private) 강제
- reentrancy 가능성 최소화

# Vyper Features

- Fixed point decimals( 고정소수점)
- Built-in Overflow
- 상속 금지
- 인라인 어셈블리 금지

# Types

- uint256

```
@public
def test() -> bytes[132]:
    a: uint256 = 1
    b: int128 = -3
    c: bytes[100] = "hello world"
    d: bytes32 = convert("hello galaxy", bytes32)

    return concat(d, c)
```

- int128

- Byte array(fix size)

- Bytes32

- Address

# Maps

- key-value
- global scope만 가능
- infinite length

```
registry: address[bytes[100]]  
  
@public  
def register(name: bytes[100], owner: address):  
    assert self.registry[name] == ZERO_ADDRESS, "already set!"  
    self.registry[name] = owner  
  
@public  
@constant  
def lookup(name: bytes[100]) -> address:  
    return self.registry[name]
```

# Lists

- 단일 타입만 가능

```
@public
def foo():
    x: int128[3] = [1, 2, 3]
    x = [4, 5, 6]
    x[1] = 1
```

# Constants

- global scope only

- built-in

- ZERO\_ADDRESS
  - MIN\_INT128
  - {MIN,MAX}\_INT128
  - MAX\_UINT256

# Assertions

- solidity처럼 message도
- 보낼 수 있음

```
@public
def foo():
    assert 1 == 2
```

```
@public
def test(a: int128) -> int128:
    assert a > 1, "larger than one please"
    return 1 + a
```

# Units

```
units: {
    cm: "centimeter",
    km: "kilometer"
}

a: int128(cm) # global storage

@public
def test() -> int128(km):
    b: int128(km)
    b = 100
    return b
```

# Functions

- Decorator
  - @private
  - @public
  - @constant
  - @payable

# Functions

- Constructor
  - `__init__(*args)`
- Default Function
  - `__default__`: transaction failure 났을때 기본적으로 실행되는 함수. fallback함수 @solidity

# Functions

- Built in(<https://vyper.readthedocs.io/en/v0.1.0-beta.8/built-in-functions.html>)
  - convert: type 바꾸는 것
  - as\_wei\_value: ether 단위( 1wei등) 으로 uint256, int128 타입을 바꿔줌
  - concat: 여러개의 bytes,bytes32 변수를 한 개의 bytes 객체로 만듦
  - send(address, value): ~에게 이더리움을 보냄
  - as\_unitless\_number: unit을 제거
  - RLPList: RLP encoding된 bytes를 decode하여 list로 반환
  - floor/ceil: 반내림/ 반올림
  - keccak256: sha3
  - ecrecover: hash값과 v,r,s 값을 받아 address값을 복원하는 함수
  - ecmul: 두 개의 타원곡선을 곱함
  - ecadd: 두 개의 타원곡선을 더함

# Events

- Event(Log)

```
Assigned: event({variable: int128})
```

```
@public
def foo(i: int128) -> int128:
    variable : int128 = i
    log.Assigned(variable)
    return variable
```

# Contract Calls

- 같은 컨트랙트의 메서드

- self로 참조

- 다른 컨트랙트

- contract keyword

- modifying: 상태를 바꾸는 함수

- constant: 상태를 바꾸지 않는 함수

```
class Foo():
    def foo(arg1: int128) -> int128: constant
```

```
@public
def bar(arg1: address, arg2: int128) -> int128:
    return Foo(arg1).foo(arg2)
```

```
class Bar():
    def foo() -> int128: modifying
    def array() -> bytes[3]: constant
```

```
@public
def bar(arg1: address) -> int128:
    return Bar(arg1).foo()
```

# Tutorial

- Vyper + Remix로 ERC20 컨트랙트 구현
  - example code: <https://bit.ly/2W7MQa4>
- 배포(javascript VM)
- review

# ERC20 explained

- 토큰 거래의 표준화를 위해 일정한 인터페이스를 제안!
  - (구현 자체는 표준이 아님)
- 주요 파라메터:
  - totalSupply
  - name
  - symbol
  - decimals
- mint, burn등은 표준은 아님!
- 유명한 구현: open zeppelin, consensys

# ERC20 explained

- 자주 하는 실수:
  - 토큰 발행량 단위 실수(decimal을 놓침)
  - 규격 이외에 추가적인 custom logic( lock, mint 등)을 넣을 때

**Thanks!**