Contract 2

from vyper.interfaces import ERC20

implements: ERC20

event Transfer:

sender: indexed(address)

receiver: indexed(address)

value: uint256

event Approval:

owner: indexed(address)

spender: indexed(address)

value: uint256

allowance: public(HashMap[address, HashMap[address, uint256]])

balanceOf: public(HashMap[address, uint256])

totalSupply: public(uint256)

nonces: public(HashMap[address, uint256])

DOMAIN\_SEPARATOR: public(bytes32)

DOMAIN\_TYPE\_HASH: constant(bytes32) = keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)')

PERMIT\_TYPE\_HASH: constant(bytes32) = keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)")

YFI: constant(address) = 0xdc7844Aaaf7a63F9b281ef0b35B3a4C9fd821c87

@external

def \_\_init\_\_():

self.DOMAIN\_SEPARATOR = keccak256(

concat(

DOMAIN\_TYPE\_HASH,

keccak256(convert("Woofy", Bytes[5])),

keccak256(convert("1", Bytes[1])),

convert(chain.id, bytes32),

convert(self, bytes32)

)

)

@view

@external

def name() -> String[5]:

return "Woofy"

@view

@external

def symbol() -> String[5]:

return "WOOFY"

@view

@external

def decimals() -> uint256:

return 12

@internal

def \_mint(receiver: address, amount: uint256):

assert not receiver in [self, ZERO\_ADDRESS]

self.balanceOf[receiver] += amount

self.totalSupply += amount

log Transfer(ZERO\_ADDRESS, receiver, amount)

@internal

def \_burn(sender: address, amount: uint256):

self.balanceOf[sender] -= amount

self.totalSupply -= amount

log Transfer(sender, ZERO\_ADDRESS, amount)

@internal

def \_transfer(sender: address, receiver: address, amount: uint256):

assert not receiver in [self, ZERO\_ADDRESS]

self.balanceOf[sender] -= amount

self.balanceOf[receiver] += amount

log Transfer(sender, receiver, amount)

@external

def transfer(receiver: address, amount: uint256) -> bool:

self.\_transfer(msg.sender, receiver, amount)

return True

@external

def transferFrom(sender: address, receiver: address, amount: uint256) -> bool:

self.allowance[sender][msg.sender] -= amount

self.\_transfer(sender, receiver, amount)

return True

@external

def approve(spender: address, amount: uint256) -> bool:

self.allowance[msg.sender][spender] = amount

log Approval(msg.sender, spender, amount)

return True

@external

def woof(amount: uint256 = MAX\_UINT256, receiver: address = msg.sender) -> bool:

mint\_amount: uint256 = min(amount, ERC20(YFI).balanceOf(msg.sender))

assert ERC20(YFI).transferFrom(msg.sender, self, mint\_amount)

self.\_mint(receiver, mint\_amount)

return True

@external

def unwoof(amount: uint256 = MAX\_UINT256, receiver: address = msg.sender) -> bool:

burn\_amount: uint256 = min(amount, self.balanceOf[msg.sender])

self.\_burn(msg.sender, burn\_amount)

assert ERC20(YFI).transfer(receiver, burn\_amount)

return True

@external

def permit(owner: address, spender: address, amount: uint256, expiry: uint256, signature: Bytes[65]) -> bool:

assert owner != ZERO\_ADDRESS # dev: invalid owner

assert expiry == 0 or expiry >= block.timestamp # dev: permit expired

nonce: uint256 = self.nonces[owner]

digest: bytes32 = keccak256(

concat(

b'\x19\x01',

self.DOMAIN\_SEPARATOR,

keccak256(

concat(

PERMIT\_TYPE\_HASH,

convert(owner, bytes32),

convert(spender, bytes32),

convert(amount, bytes32),

convert(nonce, bytes32),

convert(expiry, bytes32),

)

)

)

)

# NOTE: signature is packed as r, s, v

r: uint256 = convert(slice(signature, 0, 32), uint256)

s: uint256 = convert(slice(signature, 32, 32), uint256)

v: uint256 = convert(slice(signature, 64, 1), uint256)

assert ecrecover(digest, v, r, s) == owner # dev: invalid signature

self.allowance[owner][spender] = amount

self.nonces[owner] = nonce + 1

log Approval(owner, spender, amount)

return True