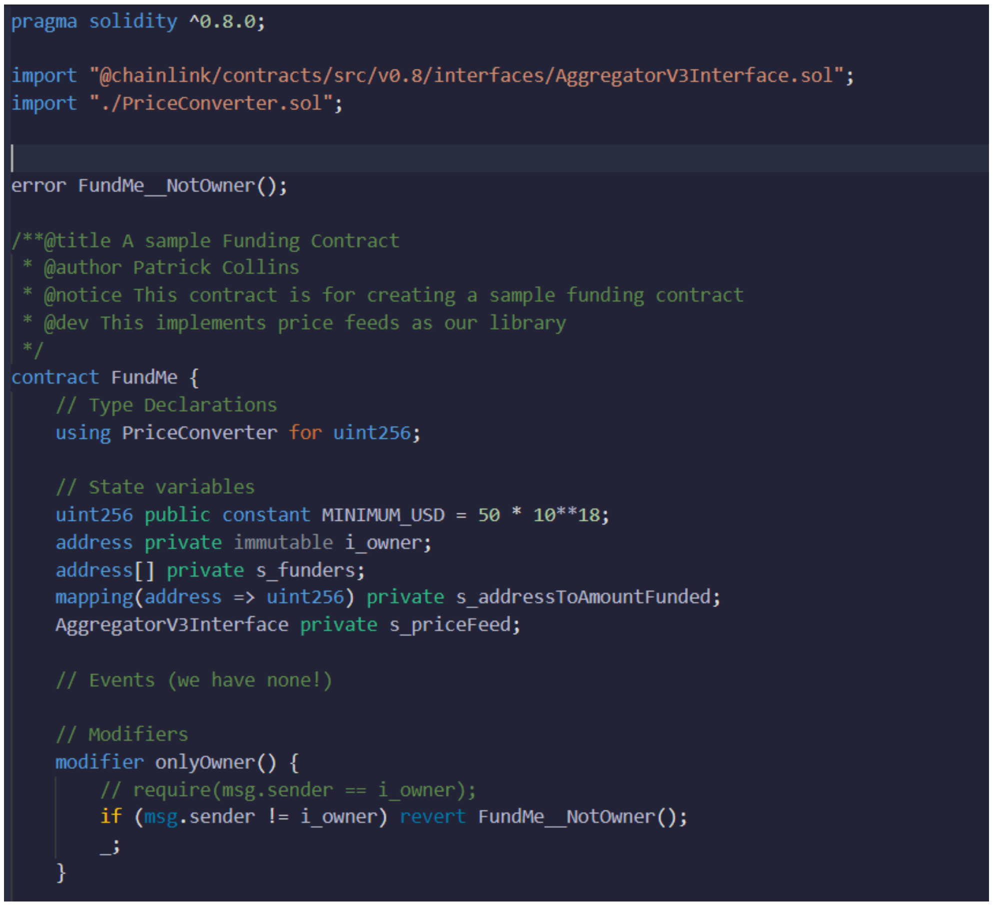
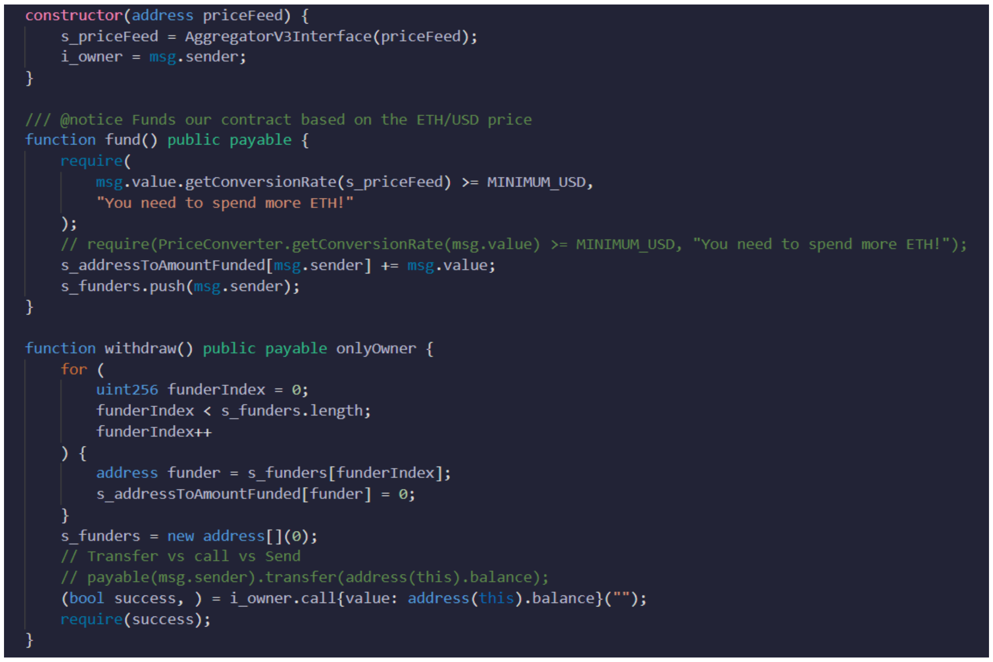
Lesson 7 Documentation

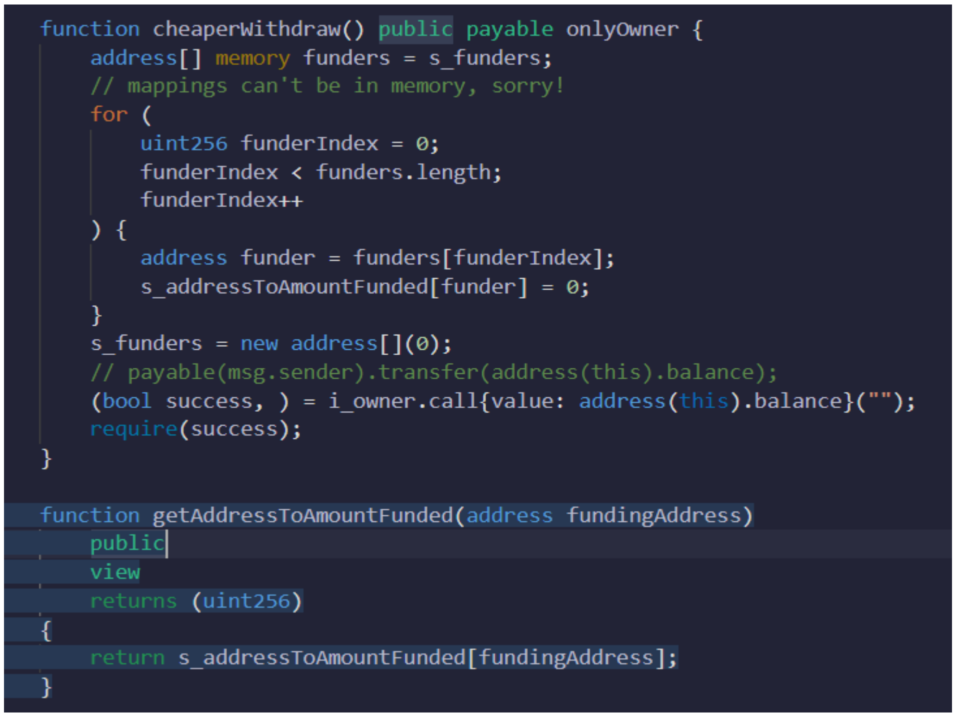
Fundme.sol



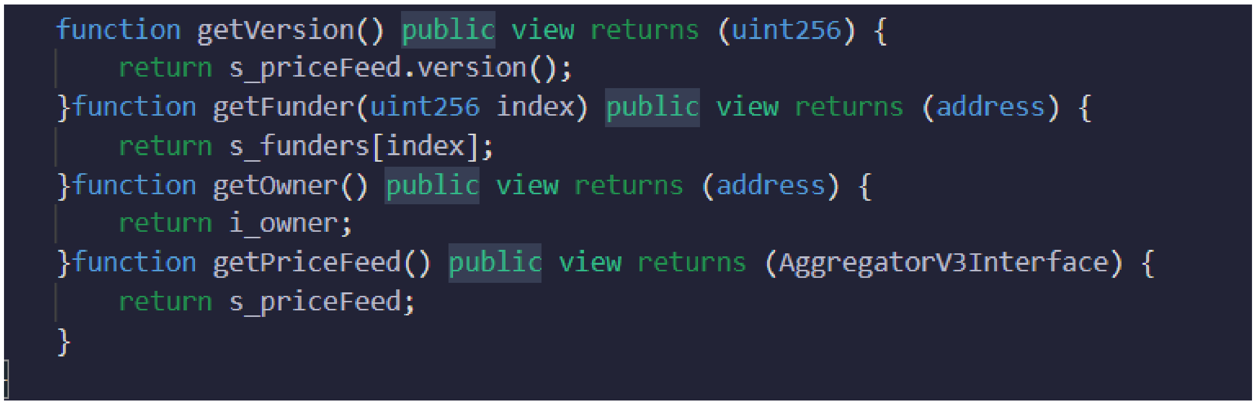
```solidity
pragma solidity ^0.8.0;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
import "./PriceConverter.sol";


error FundMe__NotOwner();

/**@title A sample Funding Contract
 * @author Patrick Collins
 * @notice This contract is for creating a sample funding contract
 * @dev This implements price feeds as our library
 */
contract FundMe {
    // Type Declarations
    using PriceConverter for uint256;

    // State variables
    uint256 public constant MINIMUM_USD = 50 * 10**18;
    address private immutable i_owner;
    address[] private s_funders;
    mapping(address => uint256) private s_addressToAmountFunded;
    AggregatorV3Interface private s_priceFeed;

    // Events (we have none!)

    // Modifiers
    modifier onlyOwner() {
        // require(msg.sender == i_owner);
        if (msg.sender != i_owner) revert FundMe__NotOwner();
        _;
    }
```



```solidity
    constructor(address priceFeed) {
        s_priceFeed = AggregatorV3Interface(priceFeed);
        i_owner = msg.sender;
    }

    /// @notice Funds our contract based on the ETH/USD price
    function fund() public payable {
        require(
            msg.value.getConversionRate(s_priceFeed) >= MINIMUM_USD,
            "You need to spend more ETH!"
        );
        // require(PriceConverter.getConversionRate(msg.value) >= MINIMUM_USD, "You need to spend more ETH!");
        s_addressToAmountFunded[msg.sender] += msg.value;
        s_funders.push(msg.sender);
    }

    function withdraw() public payable onlyOwner {
        for (
            uint256 funderIndex = 0;
            funderIndex < s_funders.length;
            funderIndex++
        ) {
            address funder = s_funders[funderIndex];
            s_addressToAmountFunded[funder] = 0;
        }
        s_funders = new address[](0);
        // Transfer vs call vs Send
        // payable(msg.sender).transfer(address(this).balance);
        (bool success, ) = i_owner.call{value: address(this).balance}("");
        require(success);
    }
```

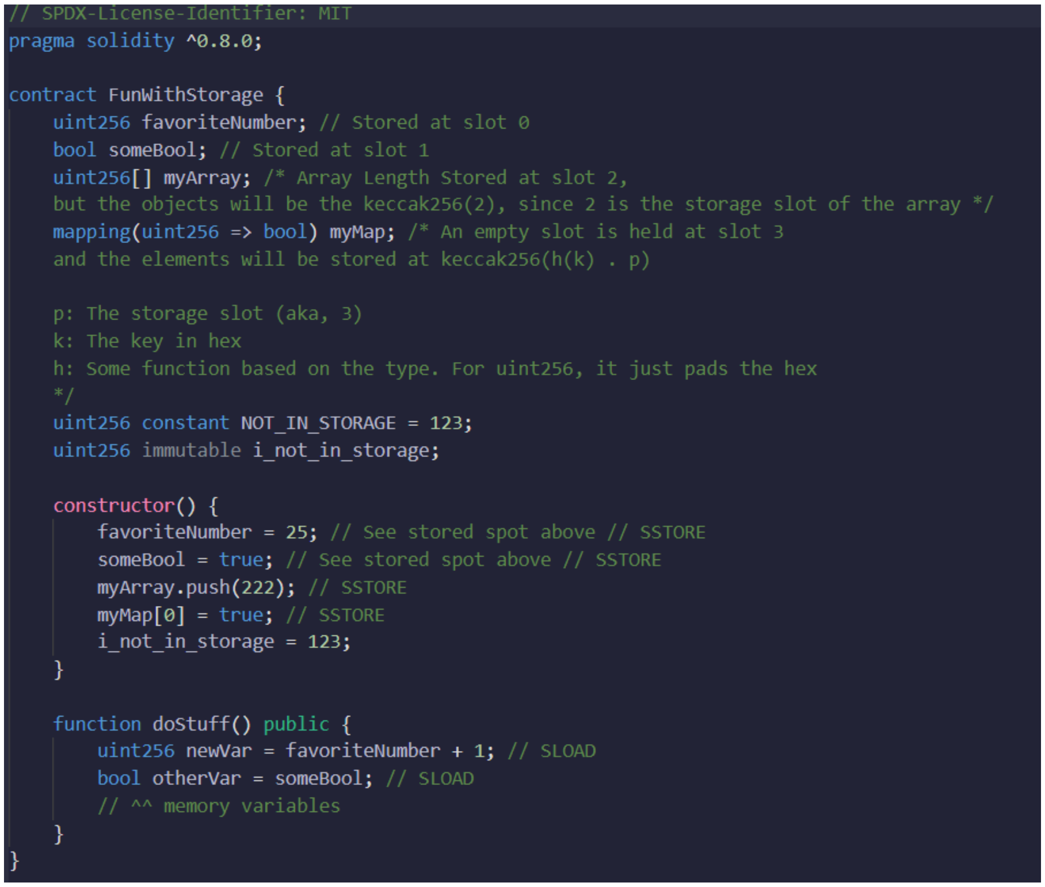


```
function cheaperWithdraw() public payable onlyOwner {
    address[] memory funders = s_funders;
    // mappings can't be in memory, sorry!
    for (
        uint256 funderIndex = 0;
        funderIndex < funders.length;
        funderIndex++
    ) {
        address funder = funders[funderIndex];
        s_addressToAmountFunded[funder] = 0;
    }
    s_funders = new address[](0);
    // payable(msg.sender).transfer(address(this).balance);
    (bool success, ) = i_owner.call{value: address(this).balance}("");
    require(success);
}

function getAddressToAmountFunded(address fundingAddress)
    public
    view
    returns (uint256)
{
    return s_addressToAmountFunded[fundingAddress];
}
```



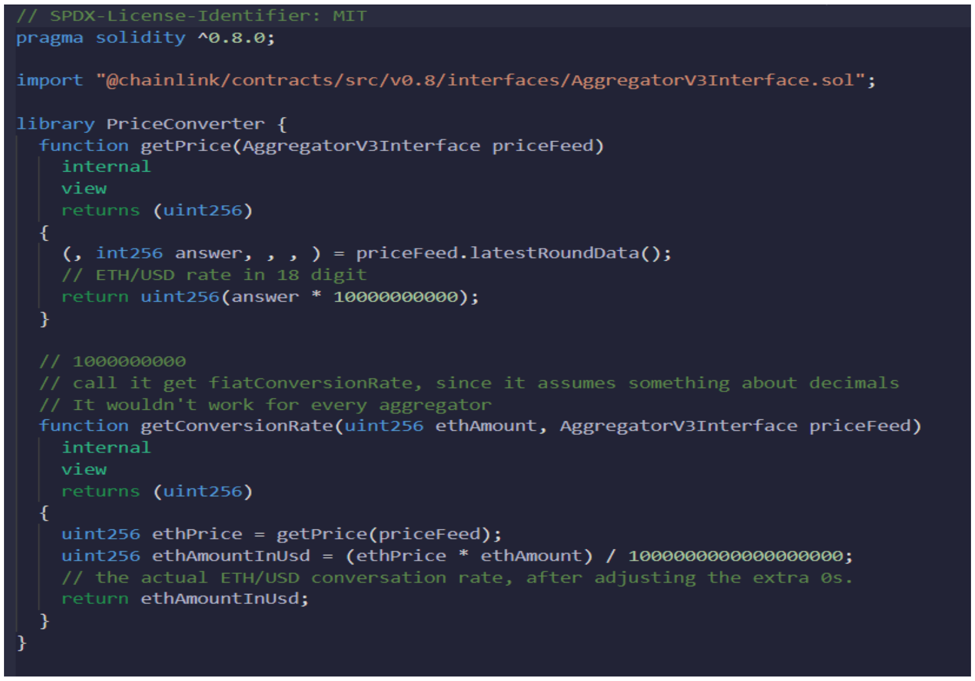
```
function getVersion() public view returns (uint256) {
    return s_priceFeed.version();
}function getFunder(uint256 index) public view returns (address) {
    return s_funders[index];
}function getOwner() public view returns (address) {
    return i_owner;
}function getPriceFeed() public view returns (AggregatorV3Interface) {
    return s_priceFeed;
}
```

Funwithstorage.sol



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract FunWithStorage {
    uint256 favoriteNumber; // Stored at slot 0
    bool someBool; // Stored at slot 1
    uint256[] myArray; /* Array Length Stored at slot 2,
    but the objects will be the keccak256(2), since 2 is the storage slot of the array */
    mapping(uint256 => bool) myMap; /* An empty slot is held at slot 3
    and the elements will be stored at keccak256(h(k) . p)

    p: The storage slot (aka, 3)
    k: The key in hex
    h: Some function based on the type. For uint256, it just pads the hex
    */
    uint256 constant NOT_IN_STORAGE = 123;
    uint256 immutable i_not_in_storage;

    constructor() {
        favoriteNumber = 25; // See stored spot above // SSTORE
        someBool = true; // See stored spot above // SSTORE
        myArray.push(222); // SSTORE
        myMap[0] = true; // SSTORE
        i_not_in_storage = 123;
    }

    function doStuff() public {
        uint256 newVar = favoriteNumber + 1; // SLOAD
        bool otherVar = someBool; // SLOAD
        // ^^ memory variables
    }
}
```

Priceconverter.sol



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

library PriceConverter {
  function getPrice(AggregatorV3Interface priceFeed)
    internal
    view
    returns (uint256)
  {
    (, int256 answer, , , ) = priceFeed.latestRoundData();
    // ETH/USD rate in 18 digit
    return uint256(answer * 10000000000);
  }

  // 1000000000
  // call it get fiatConversionRate, since it assumes something about decimals
  // It wouldn't work for every aggregator
  function getConversionRate(uint256 ethAmount, AggregatorV3Interface priceFeed)
    internal
    view
    returns (uint256)
  {
    uint256 ethPrice = getPrice(priceFeed);
    uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000;
    // the actual ETH/USD conversation rate, after adjusting the extra 0s.
    return ethAmountInUsd;
  }
}
```