Exercise 8

1. In the lecture, you learned about the oracle design pattern.
   (a) What is an oracle and what is it used for?
   (b) Explain the difference between a synchronous and an asynchronous oracle.
   (c) What are advantages and disadvantages of oracles?
   (d) The provider of a synchronous oracle has a daily budget of 10 USD. Every how many blocks can they update the value of the oracle with this budget? Assume average transaction costs. Use current values from Etherscan.

2. A popular application of smart contracts are voting systems. In this exercise, we investigate such a system, which has a very subtle but common bug. Consider the following smart contract:

```
pragma solidity >=0.5.1 <0.6.0;

contract FlawedVoting {
  mapping (address => uint256) public remainingVotes;
  uint256[] public candidates;
  address owner;
  bool hasEnded = false;

  modifier notEnded() {
    require(!hasEnded);
    _;
  }

  constructor(uint256 amountOfCandidates) public {
    candidates.length = amountOfCandidates;
    owner = msg.sender;
  }

  function buyVotes() public payable notEnded {
    require(msg.value >= 1 ether);
    remainingVotes[msg.sender] += msg.value / 1e18;
    msg.sender.transfer(msg.value % 1e18);
  }

  function vote(uint256 _candidateID, uint256 _amountOfVotes) public notEnded {
    require(_candidateID < candidates.length);
    require(remainingVotes[msg.sender] - _amountOfVotes >= 0);
    remainingVotes[msg.sender] -= _amountOfVotes;
    candidates[_candidateID] += _amountOfVotes;
  }

  function payoutVotes(uint256 _amount) public notEnded {
    require(remainingVotes[msg.sender] >= _amount);
    msg.sender.transfer(_amount * 1e18);
    remainingVotes[msg.sender] -= _amount;
  }

  function endVoting() public notEnded {
    require(msg.sender == owner);
    hasEnded = true;
    msg.sender.transfer(address(this).balance);
  }

  function displayBalanceInEther() public view returns(uint256 balance) {
     uint balanceInEther = address(this).balance / 1e18;
     return balanceInEther;
  }
}
```

   (a) Describe the (intended) basic functionality of this contract.
   (b) What is the problem with the contract? You can assume that the contract creator is benign and does not e.g. end the voting prematurely. If you are having trouble to see the bug, copy the code into the Remix IDE and try calling some functions.

(c) Which Solidity idiom is commonly used to prevent this type of error?

(d) Rewrite the respective code to make the contract work as intended. How exactly does the idiom prevent the error?