# Exercise 6

1. (a) Name and describe the four use case examples of smart contracts from the lecture.
   (b) There are many use cases for smart contracts. Think of three more and explain them as well.
   (c) Find concrete addresses of contracts for five of the use cases from the previous two subtasks on the Ethereum mainnet. Explain briefly why a contract falls into a specific category. Hint: `https://etherscan.io/labelcloud` might help.

2. (a) What is gas in the context of Ethereum?
   (b) What happens with spare gas if too much gas was provided for a transaction?
   (c) Rank the following EVM instructions by their gas cost. Note: the `CREATE` instruction creates a new smart contract.
   - `SHA3`
   - `PUSH4 0x00000000`
   - `SELFDESTRUCT` (if no new account is created)
   - `JUMP`
   - `CREATE`
   - `CALL`
   - `PUSH1 0xff`
   - `STOP`

3. In contrast to the transaction based ledger of Bitcoin, Ethereum uses an account based ledger to maintain the world state. In Bitcoin double spend attacks are prevented by UTXOs: Once a UTXO is spent, it cannot be spent again which creates a chain of coin ownership. How does Ethereum prevent double spends?

4. In Ethereum there are two types of transactions. Whenever a wallet calls a method on a contract, a transaction is sent. Whenever a contract calls a method on another contract, a virtual message is sent (Sometimes referred as internal transactions). Messages exist "virtually" and are not written to the blockchain.

   Why are messages not published to blockchain? How could the blockchain be in the same state among all nodes without writing messages to the blockchain?

5. (a) Name four reasons why a transaction that is sent to the Ethereum network might not get mined.
   (b) Name two different ways how a mined transaction can fail.
   (c) Why does a mined transaction that fails still cost gas?

6. In this exercise, we take a closer look at storing data on the Ethereum blockchain.
   (a) What is the difference between memory and storage in Ethereum?
   (b) At a very low level, storing data to storage is performed by the EVM instruction `SSTORE`. According to the yellow paper, the gas cost of an `SSTORE` operation is 20,000 "when the storage value is set to non-zero from zero" and 5,000 "when the storage value's zeroness remains unchanged or is set to zero." Why is it cheaper to set a value to 0 than to set it from 0 to any other value?
   (c) Calculate an estimation of how much it approximately costs (in USD) to store 1 KiB of data on the Ethereum blockchain at the moment. For simplicity, consider only the cost of `SSTORE` operations and disregard everything else. Use current usual values from Etherscan for your calculation.

(d) Remix is an emulator to run and test Solidity code in browsers. Run the following test program in the Remix Emulator (`https://remix.ethereum.org/`). Empirically determine the real transaction gas cost of storing 1 KiB of data on the blockchain.

To do that copy and paste the code to Remix editor. Compile the code and click to Run section. Click "Deploy" button to deploy the smart contract to the blockchain. In the console below (in gray background) you will see the receipt of the deploy operation.

On below right you will notice deployed contracts. If you expand the section you can see the methods you can call in those contracts.

Now send a transaction using "put" method writing 1 KiB of data in the contract. The put method accepts an array of 32bytes. Therefore you will write 32 items. Go ahead and fill the following array to the parameter field of the `put()` method in Remix. [1]

Press put and find the transaction cost. Determine the price in USD as well.

```
pragma solidity ^0.5.1;

contract StorageTest {

    bytes32[] b;
    function put(bytes32[] memory _b) public {
        b = _b;
    }
}
```

(e) What additional cost factors are responsible that the real gas price determined in subtask (d) is higher than the estimation from task (c)? Name three factors.

7. In this exercise, we investigate one of the biggest hacks in the history of Ethereum, the first Parity multi-sig wallet hack from July 2017.

(a) What is a multi-sig wallet? What is Parity?

(b) The hackers exploited a bug in the `initWallet()` function of the multi-sig wallet in order to wrongfully transfer funds from the affected contracts to their own account. What is the problem with this Solidity code? How could the bug have been prevented?

```
1  // constructor - just pass on the owner array to the multiowned and
2  // the limit to daylimit
3  function initWallet(address[] _owners, uint _required, uint _daylimit) {
4    initDaylimit(_daylimit);
5    initMultiowned(_owners, _required);
6  }
```

[1] ["0x0100000000000000000000000000000000000000000000000000000000000000","0x0200000000000000000000000000000000000000000000000000000000000000",
"0x0300000000000000000000000000000000000000000000000000000000000000", "0x0400000000000000000000000000000000000000000000000000000000000000",
"0x0500000000000000000000000000000000000000000000000000000000000000", "0x0600000000000000000000000000000000000000000000000000000000000000",
"0x0700000000000000000000000000000000000000000000000000000000000000", "0x0800000000000000000000000000000000000000000000000000000000000000",
"0x0900000000000000000000000000000000000000000000000000000000000000", "0x0a00000000000000000000000000000000000000000000000000000000000000",
"0x0b00000000000000000000000000000000000000000000000000000000000000", "0x0c00000000000000000000000000000000000000000000000000000000000000",
"0x0d00000000000000000000000000000000000000000000000000000000000000", "0x0e00000000000000000000000000000000000000000000000000000000000000",
"0x0f00000000000000000000000000000000000000000000000000000000000000", "0x1000000000000000000000000000000000000000000000000000000000000000",
"0x1100000000000000000000000000000000000000000000000000000000000000", "0x1200000000000000000000000000000000000000000000000000000000000000",
"0x1300000000000000000000000000000000000000000000000000000000000000", "0x1400000000000000000000000000000000000000000000000000000000000000",
"0x1500000000000000000000000000000000000000000000000000000000000000", "0x1600000000000000000000000000000000000000000000000000000000000000",
"0x1700000000000000000000000000000000000000000000000000000000000000", "0x1800000000000000000000000000000000000000000000000000000000000000",
"0x1900000000000000000000000000000000000000000000000000000000000000", "0x1a00000000000000000000000000000000000000000000000000000000000000",
"0x1b00000000000000000000000000000000000000000000000000000000000000", "0x1c00000000000000000000000000000000000000000000000000000000000000",
"0x1d00000000000000000000000000000000000000000000000000000000000000", "0x1e00000000000000000000000000000000000000000000000000000000000000",
"0x1f00000000000000000000000000000000000000000000000000000000000000", "0x2000000000000000000000000000000000000000000000000000000000000000"

(c) The address of the hackers is `0xB3764761E297D6f121e79C32A65829Cd1dDb4D32`. Go to `https://etherscan.io` and investigate the oldest three transactions that this account was involved in. What steps did the hackers perform to steal Ether? Which smart contract was exploited first? Which functions were called? How much Ether did they obtain in this first step?

(d) How much ETH did the hackers steal in total? How many contracts were involved? Use Etherscan to determine how much it was worth in USD at that time.