Lab Session 5

# Theme of this Lab Session

Today Lab is split into two parts. This first part involves introducing you to basic and essential web development. This includes writing HTML file and using basic CSS to style or using available templates code available using the open-source CSS framework Bootstrap. The second part involves more advanced questions that are related to the lectures.

This particular Lab's setup is as follows: If you are highly experienced with HTML, CSS, and Bootstrap. Please sit in the back of the class with other colleagues and start working on part two of this Lab. Otherwise, please sit in front and team with another colleague and start working on part one of this Lab.

# Pre-requests skills for this Lab

* Having and running VS code editor with the configuration done in Lab session 1.

**Notes before solving the questions**:

1. If you are stuck in any of the questions,
   1. try looking at the solidity documentation (make sure it’s the latest version - <https://docs.soliditylang.org/en/latest/>)
   2. Most of the common questions can be available on Stackoverflow and simply by searching online.
   3. If any of the above methods did not work, try asking your question using the following [procedure](https://stackoverflow.com/help/how-to-ask) on Piazza and one member of the Teaching team will answer it.
   4. If you struggle to write down the question in an explanatory manner, then please raise your hand so that one of the teaching team can come to provide you with feedback.
2. Please follow the coding style guide of the solidity language provided [here](https://docs.soliditylang.org/en/latest/style-guide.html).

Main Lab Questions

# Part 1: Reading, Understanding, and Modifying moderate Smart Contract using Solidity programming

This question is dedicated to understanding how to read available smart contracts. This practice is vital because it teaches developers how to read smart contracts and eventually write one. In particular, in this question, you will use several provided examples available in Remix and Truffle.

The questions will help you understand the procedural form of reading, interpreting, modifying, and implementing smart contracts.

In Remix IDE, three built-in example contracts are automatically provided in the File Explorers panel (left-hand side icon panel). The below sub-questions will be dedicated for reading and understanding these sub-questions. Before doing these questions, please ensure that you have attended all the pre-recorded lectures designed for this lab. Otherwise, it will be challenging to answer these questions.

## Questions:

1. Navigate to “1\_Storage.sol” file in Remix IDE. <http://remix.ethereum.org>.   
   
   1. Skim the code. The first line of the contract says “uint256 number;”. What does uint256 mean? What is the maximum value that this number can store? Is the variable “number” local variable?

Unsigned integer of length 256 bits.

Max value is 2^256-1.

It’s not a local variable, it is part of the contract’s state.

* 1. What is a public function?

It can be called externally, i.e. by users who initiate transactions.

* 1. What does the function “store” do? Can you demonstrate it using Remix? Why the variable “number” was not defined inside the function?

It updates the ‘number’ variable to the input value of the function.

The variable needs to be declared as an attribute of the contract for it to persist.

* 1. Why is there the word “view” in the function “retrieve”? Is it possible that the “returns” keyword can exist without declaring a keyword that describes the behaviour of the function within a contract?

The keyword ‘view’ declares that the function won’t change any of the state variables (the attributes of the contract).

* 1. What is a private function? Modify this contract so that you include a private function named “ExamplePrivate” that does not have an option “view” and that you call it from the function retrieve.

Can only be called internally from within the contract.

1. Navigate to “2\_Owner.sol” file in Remix IDE.  
   
   1. What is the variable “owner” in that contract? Is it private in the sense no one can access this variable?

The owner variable represents the person (or contract) that deployed the Owner contract.

The owner variable is private. No one can see or access it other than the contract itself.

* 1. What is an event data type in solidity programming? How many input parameters does it take? What does the event “OwnerSet” do?

Events are used to show what is happening in the contract to the outside world. Events are written to the ‘transaction log’ which is stored on the blockchain. These can be viewed by anyone who can see the blockchain blocks (so anyone).

The event OwnerSet emits an event telling the world that the owner has been changed from the ‘oldOwner’ to the ‘newOwner’.

* 1. What does “address (0)” stand for? Write a “pure” function that can not be called from within the contract that may return and view this address (“address (0)”). Give a narrative on the function you wrote.

It’s a special address that is the default address for any newly created smart contracts.

* 1. What is a constructor function? What does the constructor of the “Owner” contract do?

A function that is invoked when the function is created. This is the only time the function can be called.

It sets the owner and emits an event.

* 1. What is a “modifier”? What is the “isOwner” modifier doing? Can you change the public function “changeOwner” such that the contract still do what it does without the need of the “isOwner” modifier?

A modifier contains a section of code that is ran before the function it’s applied to is invoked. The modifier code usually performs checks, such as checking who owns the contract.

* 1. Explain in simple English what does this code do and why was it written this way? Where do you think that such a concept could be useful in finance?

It gives a user of the contract the ability to give the contract an owner. The owner can then set someone else as the owner.

1. The last example provided in Remix is 3-Ballot.sol contract. In English, a ballot is defined as a system of secret voting.[[1]](#footnote-2)   
     
     
     
   1. When reading a contract for the first time, what is the recommendation that the developer needs to follow in order to understand the contract’s logic?

Start by reading the name of the contract. What does it tell you about what the contract does?

Next, read the variable names, what do they tell you about the contract?

Next, read the function names, what does the contract allow you to do?

Only at the end should you start reading any of the code. The variable names should tell you all you need to know about the functionality of a contract.

* 1. In the constructor function, you can see that it takes an input array whereby each entry represents a 32-byte value. This array is represented by the variable “proposalNames”. Given that you know the basic set-up of a ballot and basic solidity language, answer the following questions
     1. What are proposal names in a ballot contract? Do you think that the variable’s name “proposalNames” capture this concept?

We initialize the contract with a list of proposals, the names of which we pass in to the constructor. Each one is initialized with 0 votes.

* + 1. It is known that Solidity compilers have to do more work i.e. generating more bytecode when data is not in chunks of 32 bytes. Therefore, having a data type different from 32 bytes may lead to higher gas and higher costs. To see this, consider the following contract code.  
         
       1. Write this code on a new file in Remix and save it as “TestGas.sol”.
       2. Deploy this contract and run the “getString” function on the interface. Define and record the transaction cost of this function.
       3. Run the “getByte32” function on the interface. Record the transaction cost of this function. How does it compare to the previous function?
       4. In the above contract, the functions were declared as “payable”. What does this mean? Do non-payable functions incur a transaction cost or only an execution cost? What is an “execution cost”? Demonstrate this idea via a code.
       5. In the above code, you can see that the keyword “constant” is being used as opposed to “immutable”. What are these keywords used for and why it is important to use them? Could we use “immutable” state variables inside “pure” functions? Would removing these keywords add transaction or execution cost to our codes? Demonstrate this by modifying the above code.

Constants cannot be changed after they are created. Immutable variables can be changed only by the constructor. Immutable state variables CAN be accessed by pure functions.

The gas cost for constant and immutable variables is lower than for regular variables.

* + - 1. In the function “getString” you could see that the return string has the keyword “memory”. Why is this keyword important?

Without the ‘memory’ keyword, variables are written to storage. Without it, they are written to memory (like the Solidity equivalent of RAM).

* + 1. Now back to “3-Ballot.sol” constructor, why 32-byte is being used instead of a string for collecting “proposalNames”?

Uses less gas to use byte-32 types.

* + 1. What is the purpose of using the “memory” keyword in the input argument of the constructor function? Furthermore, although the input variable can not be stored in the storage of the contract, how could a new caller (voter) of the contract vote for those proposed names?
    2. Explain each line of the constructor function and shed lights on the below comments in your explanation.
       1. Why is it important that the chairperson variable be public?
       2. What does proposal variable important in a ballot.

1. A delegate is a politician who speaks on behalf of a group of people. In the “3-Ballot.sol” contract there is a function called “delegate” that takes an input address named “to”
   1. What exactly does this function do?
   2. Can you find anything wrong about this function’s logic?
   3. If you found out some logical mistakes, can you fix them?
2. To fully understand the ballot, it is important to test it. Deploy the “3-Ballot.sol” into the browser blockchain in Remix. Answer the following questions.
   1. As you have realized to deploy this contract into a blockchain, it is important that you input the input variable “proposalNames”. Suppose you have two proposed names: “Bassel” and “Theo”. What is the best way to input these values into the “proposalNames” array? Using this information deploy the contract into Remix’s browser sandbox blockchain. (hint: Using Nodejs and the package ethers)
   2. Once the contract is deployed, you may see the following on the left-hand side panel  
      A screenshot of a computer screen

      Description automatically generated with medium confidence  
        
      Now, denote the address of the deployer as D1, address of another account as A1, and address of another account as A2. Each address is different from the other. To test your contract, particularly the “delegate” function do the following
      1. Change the account to A1 and in the delegate paste A2. What does this return? Have you expected this when looking at the code? Please reflect on your thoughts.
      2. Let D1 give the right for A1 to vote using the “giveRightToVote”. Does this work? Would you have expected this by simply reading this function?
      3. Let D1 give the right for A2 to vote. What happens?
      4. Let D1 vote to nominee 0. Let A2 vote for nominee 1. Press on winnerName. Who won? What lessons could you extract from this experimental test?
3. In the above questions, there are many ideas that you have acquired. Based on these concepts, indicate whether the following statements are true or false. Justify your decision by referring to the previous questions' answers or providing additional resources/codes that explain your line of thought.  
   1. A private or an internal function can be a payable function.
   2. Pure functions do not incur transaction costs.
   3. A payable function always incur transaction costs.
   4. An internal function can be declared as “view”.
   5. A private function can be declared as “view”.
   6. A pure function can return a state variable.
   7. A pure function can have a view characteristic.
   8. A “memory” keyword can only be used in a method (function).
   9. A “memory” keyword can be used at a contract level.
   10. When writing a struct, it is an important practice to understand the limit boundaries of each element inside of that struct in order to write an efficient code.
   11. When the keyword “storage” is used (like for example in the ballot.sol contract), it usually acts as a pointer into variables that are State variables.
   12. Whenever using the keyword “storage” inside a function, this implies that the developer is defining a new state variable but not on a contract level.

# Part 2: Creating a CoinFlip Smart Contract

In this question, we will create a simple binary betting smart contract (in Ethereum). This contract is very important as we shall use it as the backend development of our future distributed application (DApp) in the following lab’s session.

The binary betting application that we will create is similar to a real-life coin flip betting game. Notably, it involves two parties (the players) that bet on the outcome of a seemingly random event (the coin flip). The outcome of such a game is that there will always be only one winner.

In a fair betting game, it is assumed that both players are given the same set of rules (protocol) and that there is

* No asymmetric information by either of the parties. That is, no player has more knowledge about the other player.
* No Moral hazard. This means that both players have entered into a contract in good faith or has not provided misleading information about their assets (wealth).

When such games happen in the current financial world, a third party is usually involved to ensure that the protocol is being implemented. These could be organizations such as law enforcement institutions, governments, central banks, rating agencies, casinos, or other potential third parties.

Now the coin-flipping game has been here since the time of Julius Caesar himself. Even many ancient societies, including ancient China, has used to play this game. This game is so well-known that mathematicians wrote a full paper to state that a physical coin toss is not 50-50 chance. Hence, the player who picks the more probable side has more chance of success. In this case, both moral hazard and asymmetric information have been violated.

When this game was shifted to be an electronic game, there were many algorithms available for developers. Such game developers were generating pseudo-random numbers that are mapped into a head or tail (binary outcome). However, good computer scientists could predict these pseudo numbers (i.e. not genuine; artificial) if they could access the code’s algorithm. Therefore, such online games (web2) can have players with asymmetric information depending on the hosting server, violating fair game rules.

Moreover, even if both players have symmetric information about the rules of the game. Such online games are run on a centralized server whereby the systems administrator has the ultimate power, and hence there is room for moral hazard in the system as it can not be fully trusted.

Therefore, it seems logical to ask whether it is possible to create a smart contract that is governed by a protocol that ensures that players will play a fair game. This is precisely the concept of a smart contract, which allows two parties who do not trust one another to interact in a clear, unbiased, and uniform way.

To interact with the contract, the contract’s code needs to be available for all developers. This means that the randomness element can possibly be generated by an artificial method. However, it is known that computers can generate truly random numbers by observing some outside data, like mouse movements or fan noise, which is not predictable and creating data from it. This will lead to a fair game. Therefore, in this line of thought, rather than creating a flip coin game where a random number generates the randomness, it should be generated by something observed outside such that both players trust that this is random. We shall talk about this later in the problem.

Let us develop this game together using both logical thinking and solidity programming.

## Questions:

1. Developers are often faced with many questions when developing smart contracts. The following will contain an abstract of the "flipping coin" game amended to fit the smart contract settings and interactions. Please read this passage in order to be able to answer the following sub-questions.   
     
     
   1. How could a developer write a code that ensures that there is no turning back once the first player selects her choice? Write a sketch function called “makeBet” that enables you to demonstrate this idea.
   2. In the above passage, it has been said that "hash key acts as an identifier to player 2... that P1 has chosen an outcome". Why would a 32-byte number act as proof for P2? Could player 2, know the choice by reading this number? How could this concept be related to the concept of Hashcash idea? Do you think incorporating this idea into this game will enable the game to be a fair game? Justify your answer.
   3. Some questions that are indirectly related to the main question.
      1. What is an ABI of a smart contract? What is the relationship between ABI seen in previous labs and bytecode? What about ABI of a parameter? How could developers get the abi of a parameter in Solidity language (and JS)?
      2. Suppose you have two variables that you want to concatenate. What is the best way to concatenate two variables using solidity language? In your answer, highlight the importance of using the method abi.encode() function.
   4. As mentioned in the introduction of Part 2, P1 must find a way to generate a truly random number whereby P2 cannot know. One way of doing so is by combining P1's coin's choice with a truly random number and then getting the hash value of such a concatenated parameter (you can use abi.encode() method). This hash value needs to be provided to P2 (why?). Therefore, once P2 plays, P1 needs to show this input to produce the hash value shared at the beginning. Having said this, amend the sketched "makeBet" given this information.
   5. Write a sketch function "takeBet" whereby only P2 can decide to enter this game. This function needs to input the choice of player 2 and the amount of money he deposits to the contract should be equal to that of P1. Then, the contract needs to ensure that P2 player address is stored.
   6. Write a sketch function "reveal" that reveals the choice and the random number to the contract. The contract then computes the hash and transfers the money from the contract address to the winner address.
   7. Remember that P1 can only call the reveal function as he only knows the hash key. Consequently, it might be possible that she never shows up. This makes the game unended, and the money will be stuck in the contract account. Please make the necessary adjustment to the written functions and add functions to ensure that this game is fair and that it ends in one way or another.
   8. Currently the contract can only be used correctly once. Alter the necessary functions to reset the contract after they are called so that the contract can be reused many times.

# Part 3: Creating advanced Smart contracts

This part will extend our simple coin flipping contract to showcase some more advanced solidity features that you might find useful later when working on your projects.

## Questions

1. Data Structures: Until now, we have been using simple variables in our contract. This [documentation](https://docs.soliditylang.org/en/latest/types.html) provides you with a good and useful reference for data structures in Solidity language.
   1. Declare a structure to hold the player selections (P1’s commitment and P2’s guess).
   2. Accoringly, alter your contract so that it uses this struct instead of just plain variables.

# Inspired References

* Mastering Ethereum Building Smart Contracts and Dapps
* GeeksforGeeks website
* Shared links
* Calls vs. transactions in Ethereum smart contracts: <https://blog.b9lab.com/calls-vs-transactions-in-ethereum-smart-contracts-62d6b17d0bc2>

1. As a matter of fact, the ballot comes from the Italian root ballotta, which means "small ball". Ballotta were used in Venice by members of the great council as a way to determine nominating committees for state officeholders.([link](https://en.wiktionary.org/wiki/ballot)) [↑](#footnote-ref-2)