Lab 15-7-2023 (Part - 1)

1 Variables

Variables in Solidity play a crucial role in storing and manipulating data within smart contracts. There are three types of variables:

1.1 Local Variables:

- Local variables are declared inside a function and are only accessible within that function's scope.
- These variables are not stored on the blockchain and are used for temporary data storage during the execution of the function.
- Function parameters are also considered local variables, as they are only accessible within the function they are defined in.

```
// SPDX-License-Identifier: MIT

// compiler version must be greater than or equal to 0.8.17 and less than
0.9.0

pragma solidity ^0.8.17;

// Creating a contract

contract LocalVariable {

   // Defining function to show the declaration and

   // scope of local variables

   function getResult() public pure returns(uint){

      // Initializing local variables

      uint local_var1 = 1;

      uint local_var2 = 2;

      uint result = local_var1 + local_var2;

      // Access the local variable

      return result;                                }
```

```
}
```

1.2 State Variables:

- State variables are declared outside any function in a contract and are permanently stored on the blockchain.
- These variables hold values that persist across multiple function calls and throughout the lifetime of the contract.
- State variables are used to maintain contract state and store important data that needs to be accessible globally within the contract.

Reading and Writing to a State Variable:

"You can read state variables, for free, without any transaction fee"

```
// SPDX-License-Identifier: MIT

// compiler version must be greater than or equal to 0.8.17 and less than
0.9.0

pragma solidity ^0.8.17;

contract StateVariable {

   uint public storedData;      // State variable

   constructor() {

      storedData = 10;   // Using State variable

   }

}
```

"In the blockchain, transactions with a higher gas fee are typically prioritized and recorded first."

TASK:

Deploy the above contract and observe the gas for reading the state variable named **'storedData'.**

1.3 Global Variables:

- Global variables are special variables that provide information about the blockchain and transaction properties.
- They exist in the global workspace and can be accessed from any part of the contract.

Name	Returns
blockhash(uint blockNumber) returns (bytes32)	Hash of the given block - only works for 256 most recent, excluding current, blocks
block.coinbase (address payable)	Current block miner's address
block.difficulty (uint)	Current block difficulty
block.gaslimit (uint)	Current block gaslimit
block.number (uint)	Current block number
block.timestamp (uint)	Current block timestamp as seconds since unix epoch
gasleft() returns (uint256)	Remaining gas
msg.data (bytes calldata)	Complete calldata
msg.sender (address payable)	Sender of the message (current caller)
msg.sig (bytes4)	First four bytes of the calldata (function identifier)

msg.value (uint)	Number of wei sent with the message
now (uint)	Current block timestamp

1.4 Important Things to be Considered:

a) It is essential to understand the distinction between these variable types and their scopes when developing Solidity smart contracts.

- ✔ Local variables are used for temporary storage within functions.
- ✔ State variables hold persistent contract data.
- ✔ Global variables provide information about the blockchain and transactions.

b) Solidity is a **statically typed** language, which means that the state or local variable type needs to be specified during declaration. Each declared variable always has a default value based on its type. There is no concept of "undefined" or "null".

c) **You should not use any of the Solidity reserved keywords as a variable name.**

d) Solidity variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character.

e) Solidity variable names are case-sensitive.

2 Constants

In Solidity, constants play an important role in ensuring the immutability and security of values within a contract.

There are two keywords related to constants: constant and immutable, each with its own characteristics and use cases.

2.1 constant

- When a state variable is declared as constant, it must be initialized at the time of declaration and cannot be changed thereafter.
- If an attempt is made to modify a constant variable, the compilation will fail.
- Only numeric variables, such as integers and booleans, can be declared as constant.
- String and bytes variables can be declared as constant, but they are not supported by the immutable keyword.

2.1.1 Benefits

1. Improved security: By ensuring that certain values remain constant, the integrity of the contract logic and data is maintained, reducing the risk of unintended modifications.

2. Gas savings: Since constant variables have fixed values known at compile-time, the compiler can optimize their usage, eliminating the need to read these values from storage during contract execution. This optimization reduces gas costs.

```
    uint256 constant CONSTANT_NUM = 82;

    string constant CONSTANT_STRING = "0xFF";

    bytes constant CONSTANT_BYTES = "Blockchain-GB";

    address constant CONSTANT_ADDRESS =
0x0000000000000000000000000000000000000000;
```

TASK:

Declare a **constant** variable PI of type uint and initialize it with the value of pi (3.14159). Write a function **calculateArea**(uint radius) that calculates and returns the area of a circle using the formula

area = PI * radius * radius.

Hints:

1) You can't store floating point in solidity directly, consider the value of PI as 31415 and make changes in formula accordingly.
2)
```
function calculateArea(uint radius) public pure returns (uint)
```

TASK:

Create a contract Person with state variables firstName and lastName of type string. Implement a function **setFullName**(string _firstName, string _lastName) that allows setting the values of firstName and lastName. Write a function **getFullName**() that returns the full name of the person.

Hints:

1)
```
function setFullName(string memory _firstName, string memory
_lastName) public
```

2)
```
function getFullName() public view returns (string memory)
```

3) To concatenate firstName and lastName, use

```
string(abi.encodePacked(firstName, " ", lastName)
```

2.2 immutable

When a state variable is declared as immutable, it can be initialized either at declaration time or in the constructor.

This flexibility allows for more dynamic initialization compared to constant variables. However, it's important to note that immutable variables cannot be changed or reassigned after initialization.

3.2.1 Benefits

Flexibility in initialization: immutable variables can be initialized based on dynamic values, such as constructor parameters or other runtime information.

Enhanced contract design: By allowing variables to be initialized in the constructor, contracts can be designed to have configurable or customizable values while maintaining immutability once deployed.

```
    uint256 public immutable IMMUTABLE_NUM = 9999999999;

    address public immutable IMMUTABLE_ADDRESS;

    uint256 public immutable IMMUTABLE_BLOCK;

    uint256 public immutable IMMUTABLE_TEST;
```

By leveraging constant and immutable keywords,you can create more robust and secure contracts.

The use of these keywords not only ensures that critical values remain unchanged, but also optimizes **gas consumption** and enhances contract design flexibility.

TASK:

Declare an immutable address variable **ADMIN_Address** and initialize it with your MetaMask Wallet address. Write a function **getAdminAddress()** that returns the value of ADMIN_Address.

TASK:

1) Deploy, observe and explain the following code.
2) Try to re-define the constants by uncommenting the 11-14 lines and see what happens.

```
// SPDX-License-Identifier: MIT
pragma solidity  ^ 0.8.17 ;
contract  Constant {
    // constant variables must be initialized at the time of declaration
and cannot be changed afterwards
    uint256  public constant CONSTANT_NUM =  10 ;
    string  public constant CONSTANT_STRING =  "0xAA" ;
    bytes  public constant CONSTANT_BYTES =  "WTF" ;
    address  public constant CONSTANT_ADDRESS =
0x0000000000000000000000000000000000000000 ;

    // CONSTANT_NUM =  100 ;
    // CONSTANT_STRING =  "0xBA" ;
    // CONSTANT_BYTES =  "Hello" ;
    // CONSTANT_ADDRESS =  0x0000000000000000000000000000000000000001 ;


    // immutable variables can be initialized in the constructor and
cannot be changed afterwards
    uint256  public immutable IMMUTABLE_NUM =  9999999999 ;
    address  public  immutable IMMUTABLE_OWNER;
    address  public  immutable IMMUTABLE_ADDRESS;
    uint256  public  immutable IMMUTABLE_BLOCK;
    uint256  public  immutable IMMUTABLE_TIMESTAMP;
    address  public  immutable IMMUTABLE_COINBASE;
    uint256  public  immutable IMMUTABLE_TEST;


    // Use constructor to initialize immutable variables, so you can use
    constructor (){
        IMMUTABLE_OWNER = msg.sender;
        IMMUTABLE_ADDRESS =  address ( this );
        IMMUTABLE_BLOCK =  block.number ;
        IMMUTABLE_TIMESTAMP = block.timestamp;
        IMMUTABLE_COINBASE =block.coinbase;
        IMMUTABLE_TEST =  test ();
```

```
    }


    function test () public  pure  returns ( uint256 ){
        uint256 what =  9 ;
        return (what);
    }
}
```

4 Reading and Writing State Variable:

1) Run, observe and explain the following code
2) How much transaction fee did you pay to deploy this contract?
3) What are professional terms used for input() and output() functions?

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract SimpleStorage {
    // State variable to store a number
    uint public num;


    function input(uint _num) public {
        num = _num;
    }

    function output() public view returns (uint) {
        return num;
    }
}
```