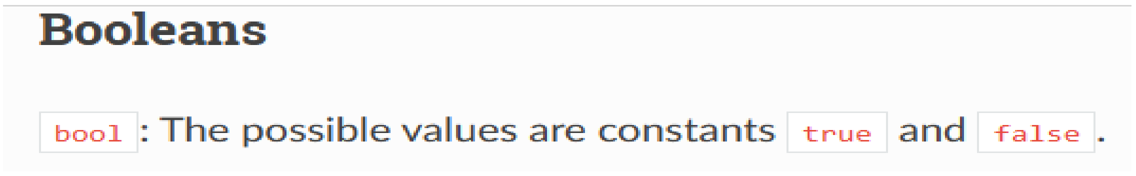
Master: Distributed Information Systems Engineering and Security
Teacher: Mrs. LOUDINI Souad
Subject: Blockchain for finance and banks

Tp N 2

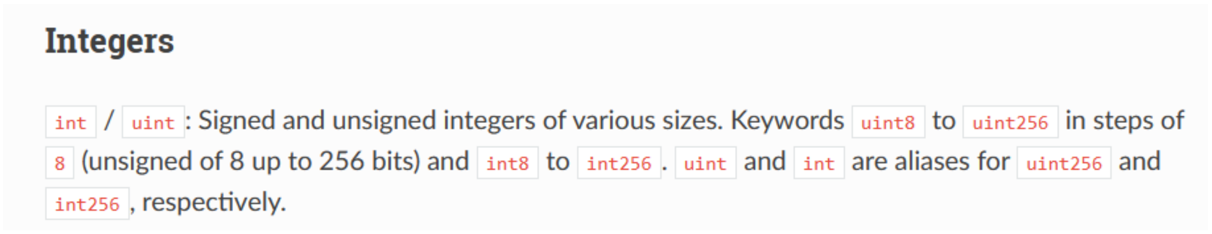
Solutions
- the different data types in Solidity

1) **Variables**



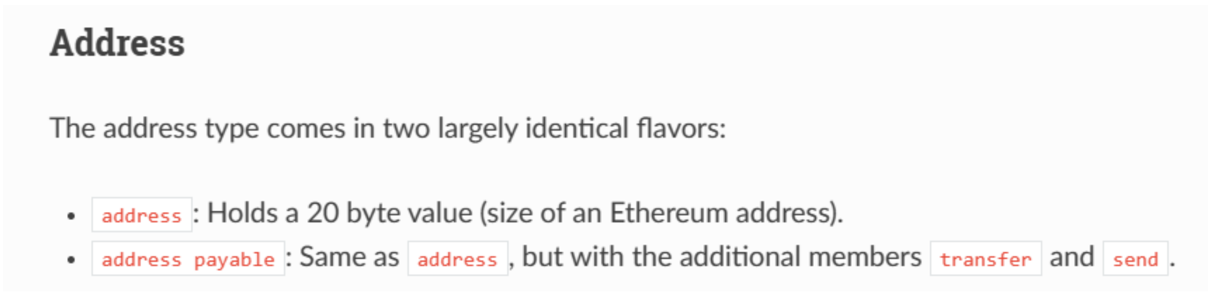
Booleans

`bool` : The possible values are constants `true` and `false` .



Integers

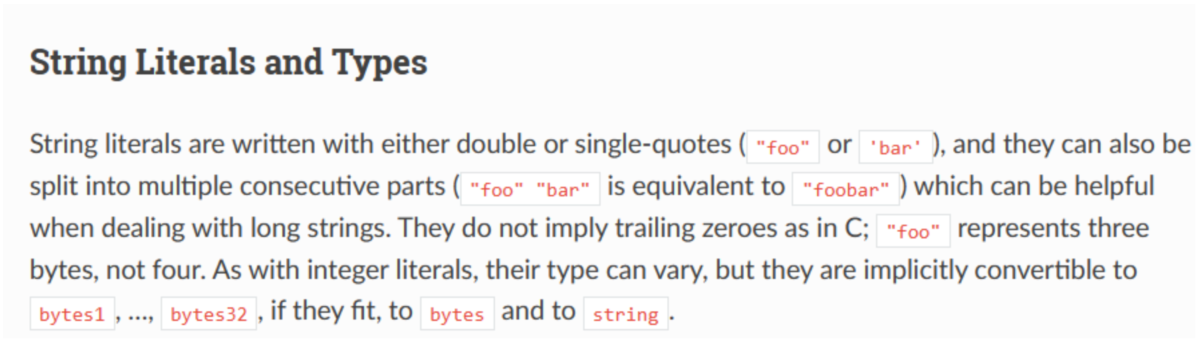
`int` / `uint` : Signed and unsigned integers of various sizes. Keywords `uint8` to `uint256` in steps of `8` (unsigned of 8 up to 256 bits) and `int8` to `int256` . `uint` and `int` are aliases for `uint256` and `int256` , respectively.



Address

The address type comes in two largely identical flavors:

- `address` : Holds a 20 byte value (size of an Ethereum address).
- `address payable` : Same as `address` , but with the additional members `transfer` and `send` .



String Literals and Types

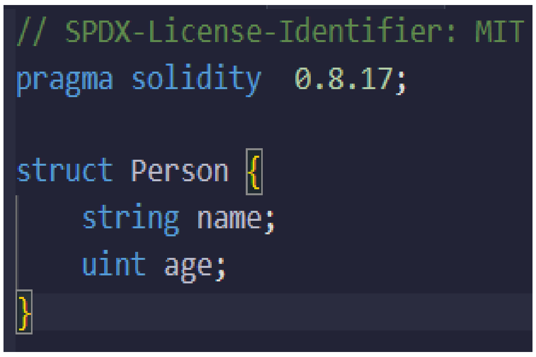
String literals are written with either double or single-quotes (`"foo"` or `'bar'`), and they can also be split into multiple consecutive parts (`"foo" "bar"` is equivalent to `"foobar"`) which can be helpful when dealing with long strings. They do not imply trailing zeroes as in C; `"foo"` represents three bytes, not four. As with integer literals, their type can vary, but they are implicitly convertible to `bytes1` , ..., `bytes32` , if they fit, to `bytes` and to `string` .

❖ There is three different types of variables:

Local Variables	**State Variables**	**Global Variables**
● declared inside the function ● not saved in to the blockchain	● declared outside the function ● stored on the blockchain	● are special variables that exist in the global namespace and provide information about the blockchain.

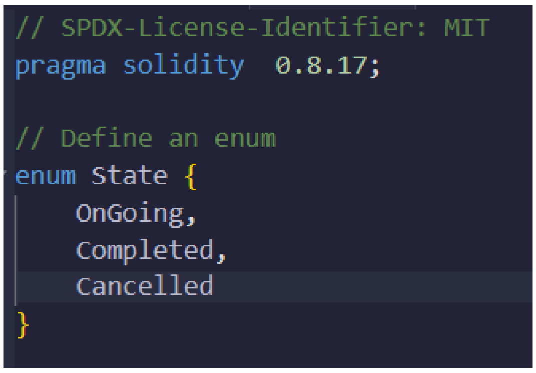
2) **Structs**

- A struct is a user-defined composite data type that allows you to group together a collection of variables with different data types under a single name.



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

struct Person {
    string name;
    uint age;
}
```

- **Enum**: An enum, short for enumeration, is a user-defined data type that represents a set of distinct values.
  Enumerations are useful when you have a fixed set of options or states for a variable.



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

// Define an enum
enum State {
    OnGoing,
    Completed,
    Cancelled
}
```

3) ***Functions***

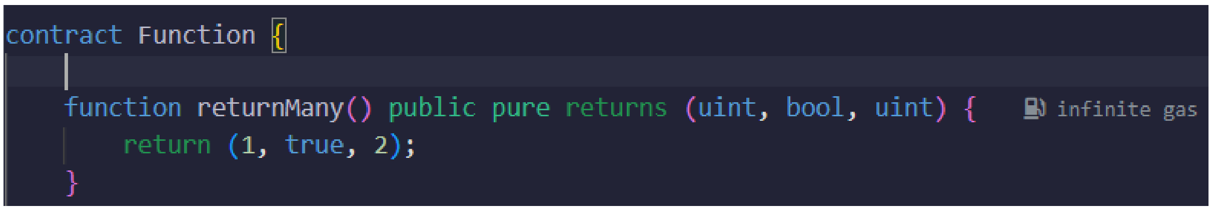
- Functions are the fundamental building blocks of smart contracts. They are reusable pieces of code that perform specific tasks or actions when called. Functions encapsulate logic and behavior,
- Functions can be defined inside and outside of contracts.
- Visibility Modifiers control who can access and execute functions. There are four types of visibility modifiers:

Public	External	Internal	Private
Public functions are accessible from anywhere, both inside and outside the contract.	External functions can only be called from outside the contract. They cannot be called internally	Internal functions can only be called from within the contract. They are not accessible from outside.	Private functions are the most restrictive. They can only be called from within the same function where they are defined.

- State Mutability indicates whether a function modifies the contract's state or not. There are three types of state mutability:

- View: View functions read data from the contract's state but do not modify it. They are useful for retrieving information without affecting the contract's state.
- Pure: Pure functions not only read data from the contract's state but also guarantee that they will not modify it under any circumstances. They are considered the most secure type of function.
- Payable: Payable functions can receive Ether payments when called. They are typically used for transactions or value transfers.

- Parameters and Return Values allow functions to take input data and provide output data. Parameters are the values that are passed to a function when it is called. Return values are the values that the function generates and sends back to the caller.

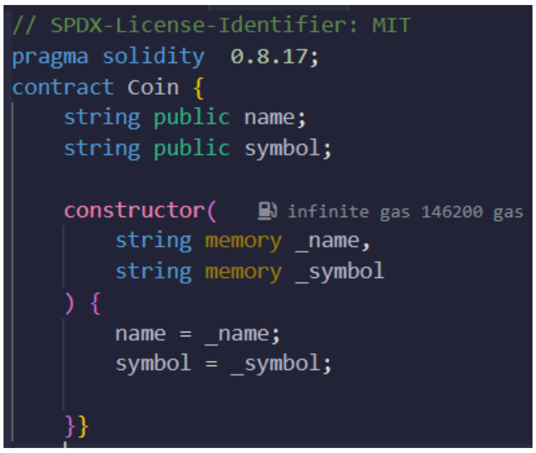


```
contract Function {

    function returnMany() public pure returns (uint, bool, uint) {    infinite gas
        return (1, true, 2);
    }
```

-

4) ***Constructors***

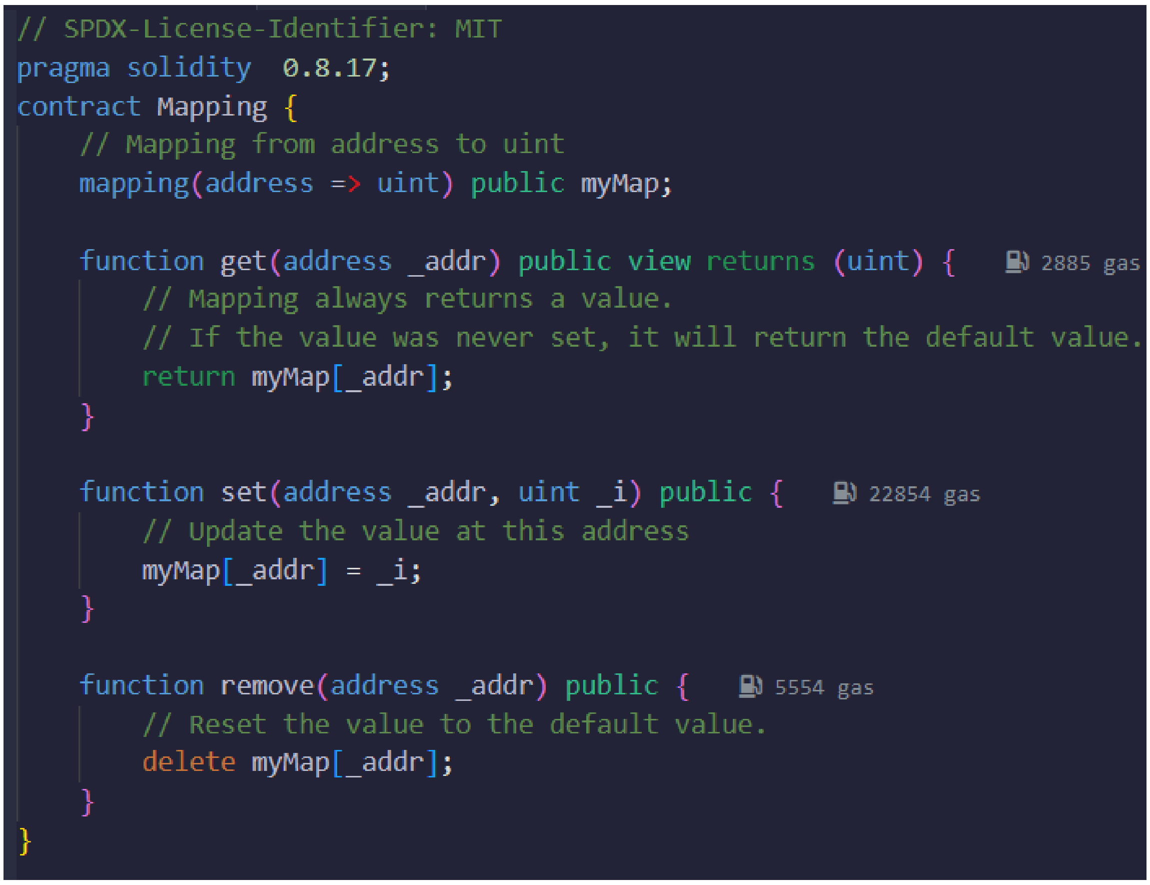
- constructor is a special function that is executed only once when a contract is deployed.
- It is used to initialize the contract's state variables.
- A contract can have only one constructor.



```
// SPDX-License-Identifier: MIT
pragma solidity  0.8.17;
contract Coin {
    string public name;
    string public symbol;

    constructor(    infinite gas 146200 gas
        string memory _name,
        string memory _symbol
    ) {
        name = _name;
        symbol = _symbol;

    }}
```

5) ***Mapping***

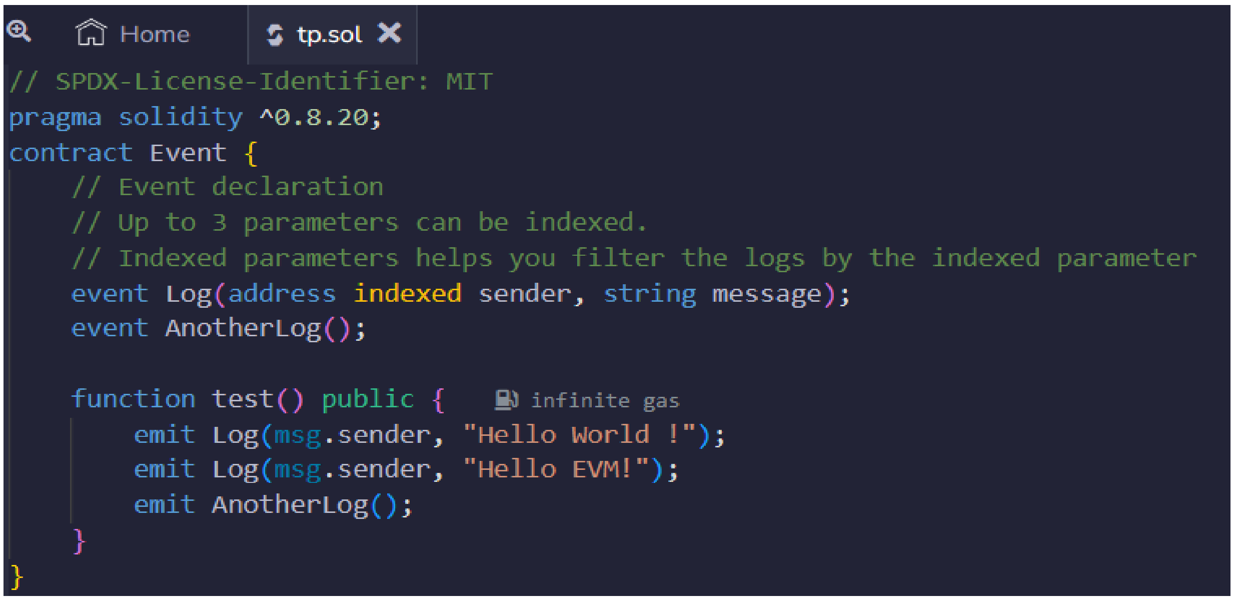
- a mapping is a data structure similar to a HashMap in C++ and Java or a dictionary (Dict) in Python
- Mapping is a (key, value) data structure, where keys and values can have different types.



```
// SPDX-License-Identifier: MIT
pragma solidity  0.8.17;
contract Mapping {
    // Mapping from address to uint
    mapping(address => uint) public myMap;

    function get(address _addr) public view returns (uint) {    2885 gas
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return myMap[_addr];
    }

    function set(address _addr, uint _i) public {    22854 gas
        // Update the value at this address
        myMap[_addr] = _i;
    }

    function remove(address _addr) public {    5554 gas
        // Reset the value to the default value.
        delete myMap[_addr];
    }
}
```

6) ***Events***

Events in Solidity serve as a way to log important information on the Ethereum blockchain. They are defined within a smart contract using the event keyword at the contract level. These events are particularly valuable for the front-end or user interface of decentralized applications, as they allow applications to listen to changes in the contract.

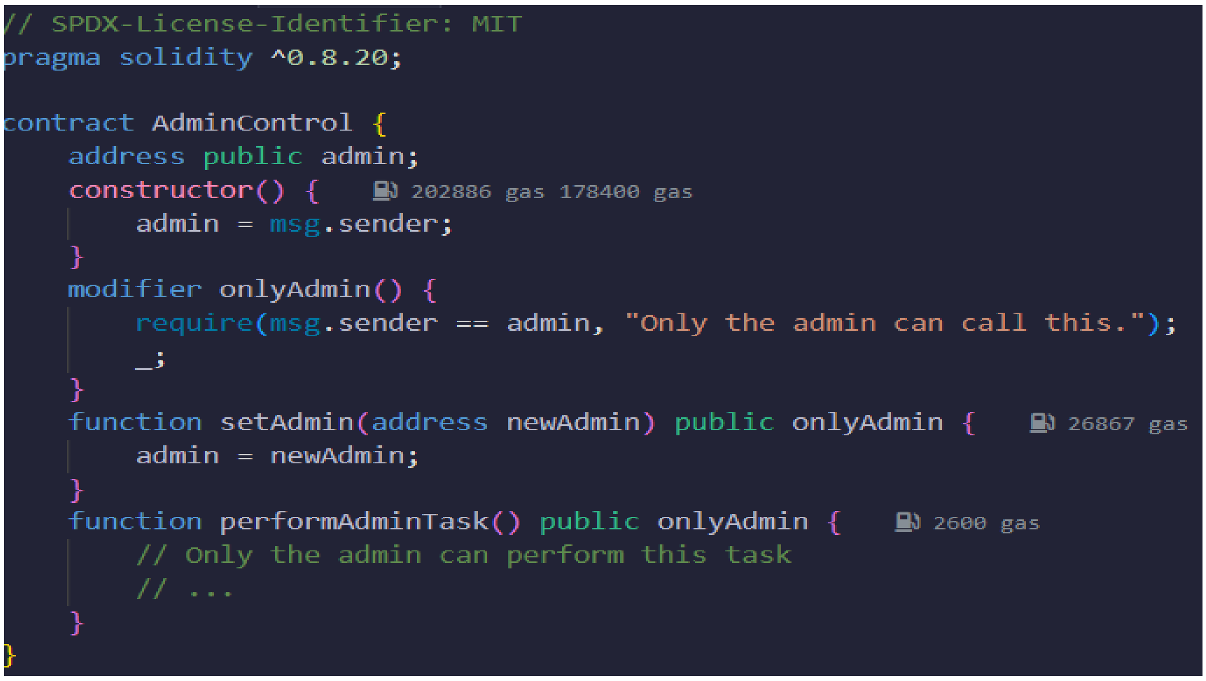


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
contract Event {
    // Event declaration
    // Up to 3 parameters can be indexed.
    // Indexed parameters helps you filter the logs by the indexed parameter
    event Log(address indexed sender, string message);
    event AnotherLog();

    function test() public {    infinite gas
        emit Log(msg.sender, "Hello World !");
        emit Log(msg.sender, "Hello EVM!");
        emit AnotherLog();
    }
}
```

7) **<u>Modifiers</u>**

Function behavior can be changed using function modifiers. Function modifier can be used to automatically check the condition prior to executing the function. These can be created for many different use cases. Function modifier can be executed before or after the function executes its code.

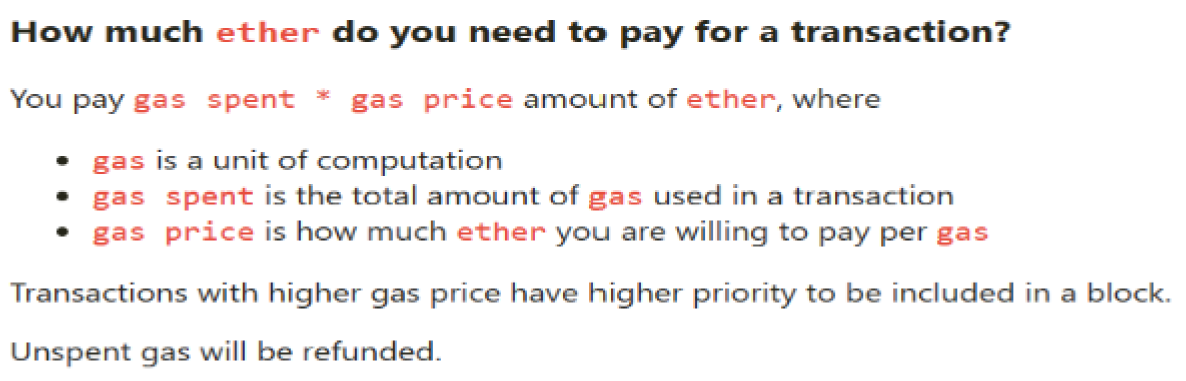
- The modifiers can be used when there is a need to verify the condition automatically before executing a particular function.
- If the given condition is not satisfied, then the function will not get executed.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract AdminControl {
    address public admin;
    constructor() {     202886 gas 178400 gas
        admin = msg.sender;
    }
    modifier onlyAdmin() {
        require(msg.sender == admin, "Only the admin can call this.");
        _;
    }
    function setAdmin(address newAdmin) public onlyAdmin {     26867 gas
        admin = newAdmin;
    }
    function performAdminTask() public onlyAdmin {     2600 gas
        // Only the admin can perform this task
        // ...
    }
}
```

8) <u>Gas</u>

Gas is a unit of computational effort on blockchain networks, representing the cost of executing operations within smart contracts.



How much ether do you need to pay for a transaction?

You pay `gas spent * gas price` amount of `ether`, where

- `gas` is a unit of computation
- `gas spent` is the total amount of `gas` used in a transaction
- `gas price` is how much `ether` you are willing to pay per `gas`

Transactions with higher gas price have higher priority to be included in a block.

Unspent gas will be refunded.

To learn more about Solidity . Visit this website : [Here]()