Solidty 学习笔记 By Alex

1. Use `pragma solidity ^` at the beginning to clarify the solidty complier version
2. `int` is 0 by default, `string` is empty, `bool` is false
3. `unit` stands for `unit256` by default
4. `unit8` goes from 0 to 255, and it would repeat itself in this range (u stands for unsigned).`int8` ranges from -128 to 127. E.g

   ```
   uint8 public myUnit8; // default 0
   myUnit8--;
   // now myUnit = 255
   ```

5. **Solidity 0.8** would automatically take care of overflows and underflows in #4!

   ```
   pragma solidity 0.8.0;
   ...
   uint8 public myUnit8; // default 0
   myUnit8--;
   // now myUnit remain 0!
   ```

6. ufixedM*N or fixedM*N means 128bits with 18 decimal points
7. string is stored as byte code in solidity. In function, you have to specify `string memory _myString` to specify the location of the string
8. String cannot be concatenated, searched or replaced in solidity, and does not have length or index attribute
9. Reference type needs a memory location(memory/storage)

   ```
   function setMyString(string memory _myString) public{
           myString = _myString;
       }
   ```

10. **Boolean**: `bool myVar; myVar != myAnotherVar;` value: `true` or `false`
11. Only **Payable** function can receive money: `function receiveMoney() public payable{}`
12. The smart contract **have to** be triggered by an external account. We cannot use contract account to initiate the smart contract itself.
13. `Address` has two members: `balance` and `transfer()`, both in Wei. Another two class `send()` and `call.gas().value()`.
14. **Msg** Object has 3 properties: `sender`, `value`, `now`.
15. `Msg.sender`: the address of the account initialized the transaction
16. `Msg.value`: How much Ether were sent along?

17. `Msg.now`: The current timestamp, can be influenced by miners, so don't use it for a random number generation.
18. **The constructor** is a *special* function. It is automatically called during Smart Contract deployment. And it can never be called again after that. It can be either public or internal.

    ```
    constructor() {
      owner = payable(msg.sender);
      //set the owner to the deployer of the smart contract
      //assign the owner of the contract at the beginning
    }
    ```

19. **selfdestruct(beneficiaryAddress)** will make the smart contract not be available in any blocks
20. **mapping** is kind of like a hash map. Syntax: `mapping(_KeyType => _ValueType) name;`. The _KeyType can be any elementary type, including bytes and string. The _ValueType can be any type, including mappings.

    ```
    contract MappingExample{
        mapping(uint => bool) public myMapping;
       // bool default value is false, so whatever you input, you would get a false

        function setValue(uint _index) public{
            myMapping[_index] = true;
        }
      // when you call setValue with 1, it indicates myMapping(1) would return true
      // index is just like a key
      // index don't have to be an integer, can be an address
    }
    ```

21. **Public** state variables of mappings, or other variables, would become a getter. Solidity will automatically generate a getter-function for us. That means, if we deploy the Smart Contract, we will automatically have a function that looks technically like this:

    ```
    mapping(uint => bool) public myMapping;

    function myMapping(uint index) returns (bool) {
        return myMapping[index];
    }
    ```

    We don't need to explicitly write the function. Also not for `myAddressMapping`. Since both are public variables, Solditiy will add these auto_magic_ally.

22. **Checks-Effects-Interaction pattern** As a rule of thumb: You interact with outside addresses *last*, no matter what. Unless you have a trusted source. So, first set your Variables to the state you want, as if someone could call back to the Smart Contract before you can execute the next line after .transfer(...).

```
function withdrawBalance(address payable _to) public{
  //_to.transfer(balanceReceived[msg.sender]);
  //balanceReceived[msg.sender] = 0;
  //Checks-Effects-Interaction Pattern
  //Because the control flow would pass over to the malicious contract, it would be able
to call the withdraw() function again, before the first invocation is finished, without
being intercepted by our check. This is because the line of code carrying the effect of
adjusting the balance would not have been reached yet. Therefore, a second transfer of
ether to the attacker would be issued.
  uint balanceToSend = balanceReceived[msg.sender];
  balanceReceived[msg.sender] = 0;
  _to.transfer(balanceToSend);
}
```

23. **struct** the member of a struct cannot be the type of struct
24. Mappings are usually preffered over Arrays because of the cost of Gas
25. `Require` returns remaining gas, used to validate user input. `Assert` consume all gas, used to validate invariants.
26. Modifying the State **costs gas**, is a concurrent operation that requires mining and doesn't return any values. Reading values, on the other hand, is virtually free and doesn't require mining. There are two types of reading functions: **view** and **pure**. Reading does not require any change in data, so it's gas-free.
27. **view**: the deployment is a transaction, while the reading operation is a call. It can access the state variables from the stat or from other view functions but only in a reading way. It can call pure functions but not writing functions.
28. **pure**: Pure functions are functions that are not accessing any state variables. They can call other pure functions, but not view functions or writting functions. State varialbes are variables stored in the smart contract E.g `address public owner`. Writing function can call both view and pure function.
29.

Function Visibility	
Public	Can be called internally and externally, including inheritance
Private	Only for the contract, not externally reachable and not via derived contracts
External	Can only be called externally
Internal	Can only be called by the contract itself or derived contracts

30. **Fallback function**: called when a transaction without a function-call is sent to the smart contract, can only be external.

```
// before solidity 0.6.0

receive() external payable {
  receiveMoney();
}
```

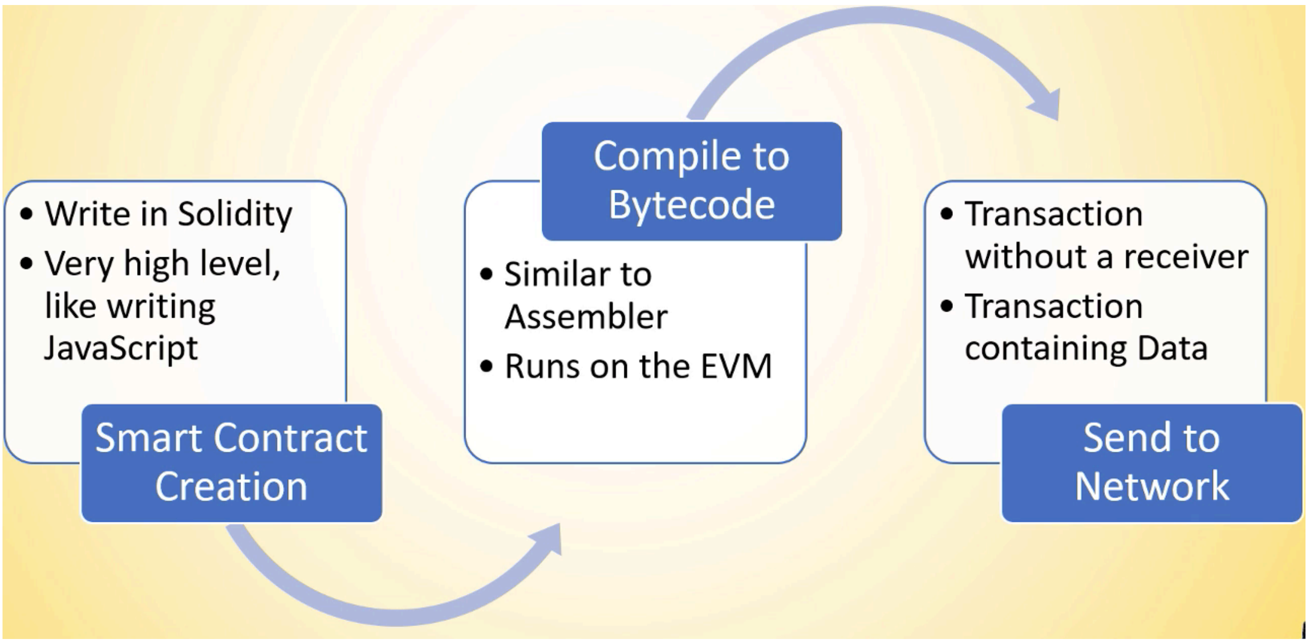
```
// after solidity 0.6.0
contract A {
    event SomeEvent(address _addr, uint _amount);

    /**
     * Will be called when (fallback) is used in Remix
     */
    receive() external payable {
        emit SomeEvent(msg.sender, msg.value);
    }

    /**
     * Will be called when msg.data is not empty or when receive() doesn't exist
     *
     * If not payable => assert-style error on msg.value not empty
     * */
    fallback () external {

    }
}
```

31. **Inheritance**: **is** as the keyword. A is X,Y,Z, where Z is the most derived contract (When X,Y,Z have the same function, first call Z). Using `super` to access the other base contract.
32. **Event**: interact with the initializer of the transaction. In real blockchain, the smart contract could not return values to the user. It is stored on the site's chain. Through event, you can return values from transactions and it's a cheaper way to store variables.

```
event TokenSent(address _from, address _to, uint _amount);

function sendToken(address _to, uint _amount) public returns(bool) {
  require(tokenBalance[msg.sender] >= _amount, "Not enough tokens");
  assert(tokenBalance[_to] + _amount >= tokenBalance[_to]);
  assert(tokenBalance[msg.sender] - _amount <= tokenBalance[msg.sender]);
  tokenBalance[msg.sender] -= _amount;
  tokenBalance[_to] += _amount;

  emit TokenSent(msg.sender, _to, _amount);

  return true;
}
// Output
[
  {
    "from": "0xA47e90B616C479F553f07D40A3648F4FAf17d1cf",
    "topic": "0x3ddb739c68dd901671f09fbe0bc2344c179ed55f8e8110a7c7a3c5665bd9518d",
    "event": "TokenSent",
    "args": {
      "0": "0xA2Bc1e1A8009aC9aEE70B0DF2A94873fB1201B55",
      "1": "0xA2Bc1e1A8009aC9aEE70B0DF2A94873fB1201B55",
      "2": "2",
      "_from": "0xA2Bc1e1A8009aC9aEE70B0DF2A94873fB1201B55",
      "_to": "0xA2Bc1e1A8009aC9aEE70B0DF2A94873fB1201B55",
      "_amount": "2"
      }
    }
  ]
```

33. **Data Storage**: Storing data on the Ehtereum Blockchain is extremely expensive. The solution would be: a) Store Data off-chain and store only a proof(hash) on the chain b) Store data in IPFS c) Store data in Event logs

34.



35. **Application Binary Interface(ABI) Array** is a json file and contains all the information to interact with the contract.

```
[
  {
    "constant": true,
    "inputs": [],
    "name": "myUint",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
  },
  {
    "constant": false,
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_myuint",
        "type": "uint256"
      }
    ],
    "name": "setMyUint",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
```

36. **Gas cost**: Appendix G. The complexity of the smart contract determines the execution cost. The transaction cost composes the type of the action you are going to do on chain.

37. **Libraries**: stateless, cannot be destroyed, no inheritance, cannot receive money

```
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/utils/math/SafeMath.sol";

using SafeMath for uint;

tokenBalance[msg.sender].sub(_amount);
// sub is from SafeMath Library

library Search {
   function indexOf(uint[] storage self, uint value)
    public
    view
    returns (uint)
   {
    for (uint i = 0; i < self.length; i++)
      if (self[i] == value) return i;
    return uint(-1);
   }
}

//method 1
uint[] data;
uint index = Search.indexOf(data,_old);

//method 2
using Search for uint[];
uint[] data;
uint index = data.indexOf(_old);
```

38. **Array**

```
//works in 0.5.15

contract MyContract {
    uint[] public myUintArray;

    function add(uint _num) public {
        myUintArray.length++;
        myUintArray[myUintArray.length - 1] = _num;
    }

    function removeElement() public {
        myUintArray.length--;
      // automatically delete the last element
    }
}

//works in 0.6.0
```

```
contract MyContract {
    uint[] public myUintArray;

    function add(uint _num) public {
        myUintArray.push(_num);
    }

    function removeElement() public {
        myUintArray.pop();
    }
}
```