3️⃣

Basics III: Structures & Libraries

Constructors

- A constructor function is executed once the smart contract is created.
- The constructor and the methods it uses don’t take place in the final code deployed to the blockchain.
- Constructors determine the initial state of the contracts.

```
contract Member {
    string name;
    uint age;

    constructor (string memory _name, uint _age) {
        name = _name;
        age = _age;
    }
}

contract Teacher is Member //("Rachel", 28)
{
    constructor (string memory n, uint a) Member(n, a) {}

    function getName() public view returns (string memory) {
        return name;
    }
}
```

Arrays

- An array is a collection of variables of the same type.
- `push` method adds one or more elements to the end of the array and returns the new length of the array.
- `pop` method removes the last element of the array and returns the value to the caller.
- `length` method returns the length of the array.
- `delete` function makes the element in the specified index zero. This function does not change the array length.
  `arr = [1, 2, 3] --> delete arr[2] --> arr = [1, 2, 0]`

```
contract LearnArrays {

    constructor () {
        fillArray;
    }

    // several ways to initialise an array
    uint[] public arrayOne;
    uint[] public arrayTwo = [1, 3, 5, 7];
    uint[10] public arrayThree;     // Fixed sized array, all elements initialise to 0

    function addValue(uint number) public {
        arrayOne.push(number); // appends user provided value to arrayOne
    }

    function removeLastElement() public {
        arrayTwo.pop(); // removes last item from arrayTwo
    }

    function getLength() public view returns (uint) {
        return arrayThree.length;
    }

    function fillArray() internal {
        arrayThree = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    }

    function removeArbitraryElement(uint index) public {
        delete arrayThree[index]; // removes (makes zero) the element in the specified index
    }
}
```

Enums

Enums are **user-defined data types that restrict the variable to have only one of the predefined values.**

> Enums are data types, not variables!

The default value for a variable of the defined enum type is the first element of the enum unless the index numbers are arbitrarily assigned by the developer.

A quick coffee shop app to practice enums:

```
contract EnumsLearn {

    // Variables of coffeeCupSize type can only be assigned SMALL, MEDIUM OR LARGE
    enum coffeeCupSize {SMALL, MEDIUM, LARGE}

    // assign MEDIUM to defaultChoice
    coffeeCupSize constant defaultChoice = coffeeCupSize.MEDIUM;

    // initialise a variable
    coffeeCupSize choice = defaultChoice;
```

```
    function getChoice() public view returns (coffeeCupSize) {
        return choice;
    }

    function makeChoice(uint i) public {
        require (i == 0 || i == 1 || i == 2, "Invalid choice!");
        if (i == 0) {
            choice = coffeeCupSize.SMALL;
        }
        else if (i == 1) {
            choice = coffeeCupSize.MEDIUM; // This line can be removed as the it's the default value
        }
        else {
            choice = coffeeCupSize.LARGE;
        }
    }
}
```

Structs

Struct types are used to represent a record. They include a batch of information for the same type of objects/assets.

```
contract LearnStructs {

    struct Film {
        string name;
        string director;
        uint id;
    }

    Film sherlock_holmes;

    function addFilm() public {
        sherlock_holmes = Film("Sherlock Holmes", "Guy Ritchie", 1);
    }

    function getDirector() public view returns (string memory) {
        return sherlock_holmes.director;
    }

    function getID() public view returns (uint) {
        return sherlock_holmes.id;
    }
}
```

Mappings

Mappings are used to store the data in the form of key-value pairs, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type.

Mappings are mostly used to associate the unique Ethereum address with the associated value type.

> Mapping in Solidity is alike a dictionary in Python.

Look how we associate an Ethereum wallet address with an integer:

```
contract LearnMappings {
    mapping (address => uint) public wallets;

    function setAddress(address _addr, uint _i) public {
        wallets[_addr] = _i;
    }

    function getValue(address _addr) public view returns (uint) {
        return wallets[_addr];
    }

    function removeAddress(address _addr) public {
        delete wallets[_addr];
    }
}
```

Examine This Code for A Better Understanding

```
contract StructMapping {
    struct Film {
        string name;
        string director;
    }

    mapping (uint => Film) public mapFilm;

    function addFilm(string memory _name, string memory _director, uint _id) public {
        mapFilm[_id] = Film(_name, _director);
    }

    function getDirector(uint _id) public view returns (string memory) {
        return mapFilm[_id].director;
    }

    function getFilm(uint _id) public view returns (Film memory) {
        return mapFilm[_id];
    }
}
```

Nested Mappings

Mapping a key to a value of another mapping is called nested mapping.

> `msg.sender` is a global variable accessible through Solidity that captures the address of the caller of the contract.

Sample Code 1

```
contract StructNestedMapping {
    struct Film {
        string name;
        string director;
    }

    mapping (address => mapping(uint => Film)) public mapFilm;

    function addFilm(string memory _name, string memory _director, uint _id) public {
        mapFilm[msg.sender][_id] = Film(_name, _director);
    }

    function getDirector(uint _id) public view returns (string memory) {
        return mapFilm[msg.sender][_id].director;
    }

    function getFilm(uint _id) public view returns (Film memory) {
        return mapFilm[msg.sender][_id];
    }
}
```

Sample Code 2

```
contract Allowance {

    mapping(address => mapping(address => uint)) public allowance;

    //this function removes the spenders allowance
    function remove(address _addrOwner, address _addrSpender) public {
        delete allowance[_addrOwner][_addrSpender];
    }

    function pairUp(address _addrOwner, address _addrSpender, uint _allowance) public {
        allowance[_addrOwner][_addrSpender] = _allowance;
    }

    function getAllowance(address _addrOwner, address _addrSpender) public view returns (uint) {
        return allowance[_addrOwner][_addrSpender];
    }
}
```