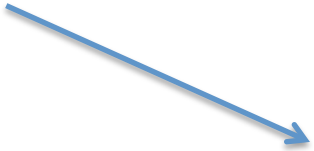
Introduction to Solidity

Murat Osmanoglu

Solidity Basics

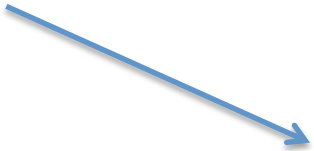
```
pragma solidity >=0.7.0 <0.9.0;
```



specify the Solidity version

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

}
```



keyword to define a smart contract
(similar to 'class' in an object-oriented
programming language)

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    int256 num = 2;
    uint256 num = 1;
    bool value = true;
    string phrase = “storage”;
    address add = 0x71C7656EC7ab88b098defB751B7401B5f6d8976F;


}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    int num;
    uint num;
    bool value;
    string phrase;
    address add;


}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 num;
    function store(uint256 number) public {
        num = number;
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 num;
    function store(uint256 number) public {
        num = number;
    }
}
```

Function Visibility

public : can be called internally or via message calls

external : can be called from other contracts and via transactions (an external cannot be called internally)

internal : can only be called from within the current contract or contracts deriving from it

private : can only be accessible from within the current contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 public num;
    function store(uint256 number) public {
        num = number;
    }
}
```

State Variable Visibility

internal : can only be accessed from within the current contract or contracts deriving from it

public : similar to internal but the compiler automatically generates getter functions for them (it allows other contracts to read their values)

private : can only be accessible from within the current contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 public num;
    function store(uint256 number) public {
        num = number;
    }
}
```

if you don't specify any keyword, it will be automatically set as internal

State Variable Visibility

internal : can only be accessed from within the current contract or contracts deriving from it

public : similar to internal but the compiler automatically generates getter functions for them (it allows other contracts to read their values)

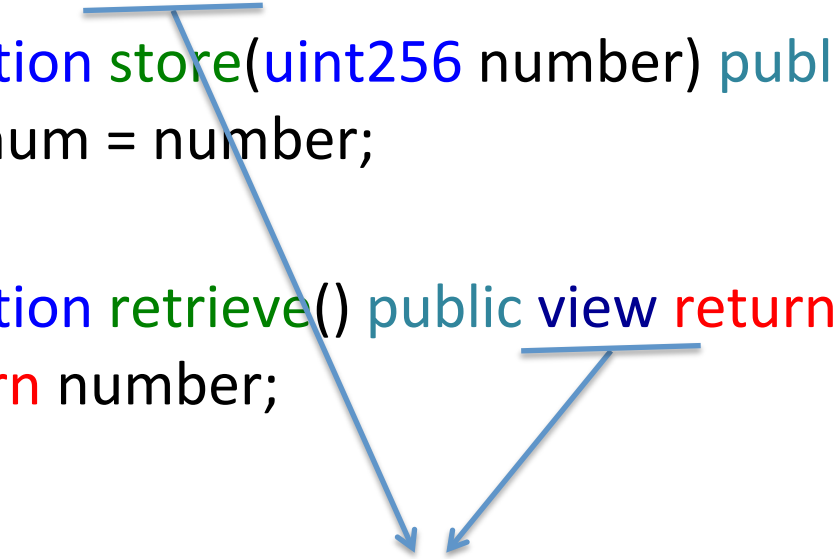
private : can only be accessible from within the current contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 public num;
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
    return number;
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 public num;
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
    return number;
    }
}
```



if you just read some data from blockchain, not requesting any state change through transactions

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 public num;
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
    return number;
    }
}
```

if you just read some data from blockchain, not requesting any state change through transactions

pure : applying some math without making any state change
they don't read or modify the state

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint256 public num;
    struct Farmers {
        uint rep;
        string id;
        uint score;
    }
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint256 public num;
    struct Farmers {
        uint rep;
        string id;
        uint score;
    }
    Farmers public newFarmer = Farmers({rep:8, id:10923, score:7});
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

custom defined types that can group several variables

how to initialize a struct

Solidity Basics

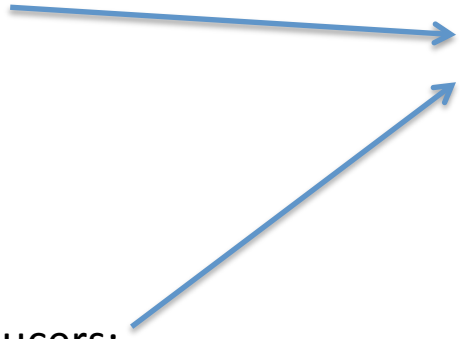
```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint256 public num;
    struct Farmers {
        uint rep;    // index 0
        string id;     // index 1
        uint score;  // index 2
    }
    // Farmers public newFarmer = Farmers({rep:8, id:10923, score:7});
    Farmers public newFarmer = Farmers(8, 10923, 7);
    function store(uint256 number) public {
        num = number;
    }
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

custom defined types that can group several variables

how to initialize a struct

Solidity Basics

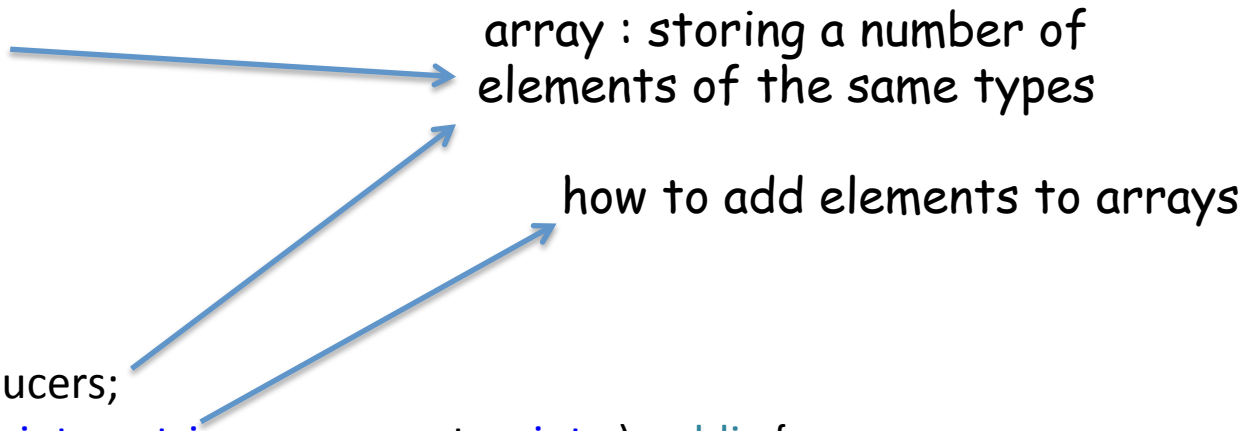
```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
        uint256 number;
        uint256[5] number1;
        struct Farmers {
                uint rep;
                string id;
                uint score;
        }
        Farmers[] public producers;

        function store(uint256 num) public {
                number = num;
        }
        function retrieve() public view returns (uint256){
                return number;
        }
}
```



array : storing a number of elements of the same types

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 number;
    uint256[5] number1;
    struct Farmers {
        uint rep;
        string id;
        uint score;
    }
    Farmers[] public producers;
    function addFarmer (uint x,  string memory str, uint y) public {
        producers.push(Farmers(x, str, y));
    }
    function store(uint256 num) public {
        number = num;
    }
    function retrieve() public view returns (uint256){
        return number;
    }
}
```



Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Storage {
    uint256 number;
    uint256[5] number1;
    struct Farmers {
        uint rep;
        string id;
        uint score;
    }
    Farmers[] public producers;
    function addFarmer (uint x,  string memory str, uint y) public {
        producers.push(Farmers(x, str, y));
    }
    funct
        f
    }
    funct
        r
    }
}
```

array : storing a number of elements of the same types

how to add elements to arrays

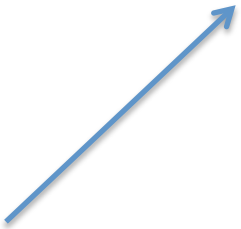
storage : each account has a data area called storage
costly to read or initialize or modify storage
state variables always in storage
local variables of struct, array or mapping storing in storage by default

memory : holds the data in it till the execution of the function
relatively cheaper
function arguments always in memory

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    struct Farmers {
        uint rep;
        string id;
        uint score;
    }

    Farmers[] public producers;

    mapping(string => uint256) public idToReputation;

    function addFarmer (uint x,  string memory str, uint y) public {
        producers.push(Farmers(x, str, y));
    }
}
```

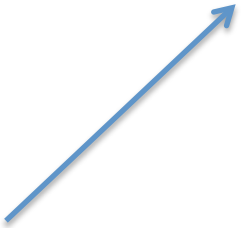
mapping types can be considered as hash tables that map bytes, string, or any contract to any type



Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    struct Farmers {
        uint rep;
        string id;
        uint score;
    }

    Farmers[] public producers;

    mapping(string => uint256) public idToReputation;

    function addFarmer (uint x,  string memory str, uint y) public {
        producers.push(Farmers(x, str, y));
        idToReputation[str] = x;
    }
}
```

mapping types can be considered as hash tables that map bytes, string, or any contract to any type

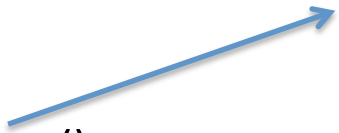


Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint public num;
    function setNum(uint x) public {
        num = x;}

    function retrieve() public view returns (uint256){
        return num + 1;}
}

contract Storage2 {


}
```

Solidity Basics

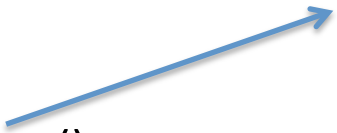
```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint public num;
    function setNum(uint x) public {
        num = x;}

    function retrieve() public view returns (uint256){
        return num + 1;}
}

contract Storage2 {
    Storage stor = new Storage();
}
```



how you initialize a contract in another contract, or how you deploy a contract in another contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Storage {

    uint public num;
    function setNum(uint x) public {
        num = x;}

    function retrieve() public view returns (uint256){
        return num + 1;}
}

contract Storage2 {
    Storage stor = new Storage();

    function setNumber(uint x) public {
        stor.setNum(x) ;}
    function getNumber() public view returns (uint256){
        return stor.retrieve();}
}
```



how you initialize a contract in another contract, or how you deploy a contract in another contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

import "./Storage.sol";

contract Storage2 {
    Storage stor = new Storage();
    function setNumber(uint x) public {
        stor.setNum(x) ;}
    function getNumber() public view returns (uint256){
        return stor.retrieve();}
}
```

how you import a contract

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

import "./Storage.sol";

contract Storage2 {
    Storage[] public storageArray;

    function createStorageContract () public {
        Storage stor = new Storage();
        storageArray.push(stor) ;
    }
}
```

with each call, it creates another Storage contract and adds it to the array

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

import "./Storage.sol";

contract Storage2 {
    Storage[] public storageArray;
    function createStorageContract () public {
        Storage stor = new Storage();
        storageArray.push(stor) ;}
    function setNumber(uint256 index, uint256 number) public {
        Storage stor = Storage(address(storageArray[index]));
        stor.setNumber(number);}
    function getNumber(uint256 index) public view returns (uint256) {
        Storage stor = Storage(address(storageArray[index]));
        return stor.retrieve();}

}
```

with each call, it creates another Storage contract and adds it to the array

Solidity Basics

how to create a derived contract (derived contract inherits all non-private members of the original contract)

```
pragma solidity >=0.7.0 <0.9.0;

import "./Storage.sol";

contract Storage2 is Storage {
    Storage[] public storageArray;
    function createStorageContract () public {
        Storage stor = new Storage();
        storageArray.push(stor) ;}
    function setNumber(uint256 index, uint256 number) public {
        Storage stor = Storage(address(storageArray[index]));
        stor.setNumber(number);}
    function getNumber(uint256 index) public view returns (uint256) {
        Storage stor = Storage(address(storageArray[index]));
        return stor.retrieve();}

}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract Cont1 {
    uint private num;
    function math1(uint256 x) private pure returns (uint) { return x + 1; }
    function setData(uint256 x) public { num = x; }
    function getData() public view returns (uint) { return num; }
    function math2(uint256 x, uint256 y) internal pure returns (uint) { return x + y; }
}
contract Cont2 {
    function readData() public {
        Cont1 cont = new Cont1();
        uint x = cont.math1(5);
        cont.setData(1);
        x = cont.getData();
        x = cont.math2(2, 3);}
}
contract Cont3 is Cont1 {
    function func() public {
        Cont1 cont = new Cont1();
        uint x = math2(2, 3);}
}
```

since math1 is private, it cannot be accessed from other contracts

since math2 is internal, it cannot be accessed from other contracts

since Cont3 is child contract of Cont1, math2 can be accessed from Cont3

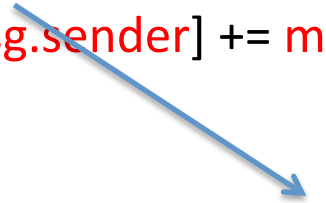
Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;
contract Cont1 {
    uint256 public num = 5;
}

contract Cont2 {
    Cont1 cont = new Cont1();
    function getData() public view returns (uint) {
        return cont.num;
    }
}
```

since the state variable num is public, it can be accessed from other contracts

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        addressToFunding[msg.sender] += msg.value;
    }
}
```

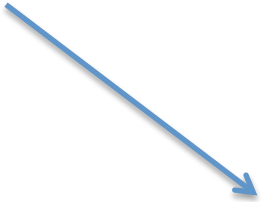


payable address or function
can send or receive ETH

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        addressToFunding[msg.sender] += msg.value;
    }
}
```

the address issuing the transaction to contact with the contract



the amount of wei sent in the transaction

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        uint256 min = 10*10**18;
        if(msg.value < min){ revert? }
        addressToFunding[msg.sender] += msg.value;
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;
    }
    function withdraw() public payable {
        msg.sender.transfer(address(this).balance);
    }

}
```

the address' balance

address.transfer(x) : sending x amount of ETH from the contract's balance to address

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;
    }
    function withdraw() public payable {
        payable(msg.sender).transfer(address(this).balance);
    }

}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;
    }
    function withdraw() public payable {
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
}
```

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    address public owner;
    constructor() public {
        owner = msg.sender;
    }
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;
    }
    function withdraw() public payable {
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
}
```

executed instantly when the contract deployed

it executes the initialization code

Solidity Basics

```
pragma solidity >=0.7.0 <0.9.0;

contract FundMe {

    mapping(address => uint256) public addressToFunding;
    address public owner;
    constructor() public {
        owner = msg.sender;
    }
    function fund() public payable {
        uint256 min = 10*10**18;
        require(msg.value >= min);
        addressToFunding[msg.sender] += msg.value;}
    modifier onlyOwner {
        require(msg.sender == owner);
        _;}
    function withdraw() public onlyOwner payable {
        payable(msg.sender).transfer(address(this).balance);}
}
```

changing behaviour of a function