-RSS3- Crossbell OpenBuild

Smart Contract Development

In Solidity

pseudoyu | November 18, 2023

About me

pseudoyu (x.com/pseudo_yu)

Blockchain developer | github.com/pseudoyu

- Tech writer/content creator | pseudoyu.com
- Back-end dev at RSS3
- Smart contract dev at Crossbell
- Solidity courses/tutorials contributor of OpenBuild

什么是智能合约

智能合约是运行在链上的程序

合约开发者可以通过智能合约实现
与链上资产、数据进行交互

用户可以通过自己的链上账户来
调用合约、访问资产与数据

与一般程序的差异

原生支持资产流动

部署与后续写入需要一定费用

存储数据的成本更高

部署后无法更改（可升级合约? ）

…

Solidity

一门面向合约的、为实现智能合约而创建的高级编程语言

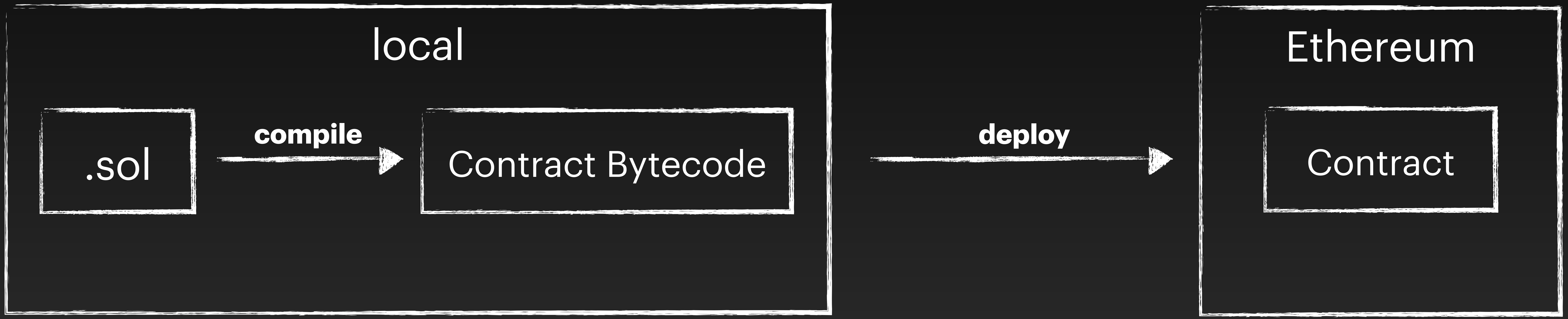
在 EVM 虚拟机上运行

语法整体类似于 javascript

是目前最流行的智能合约语言

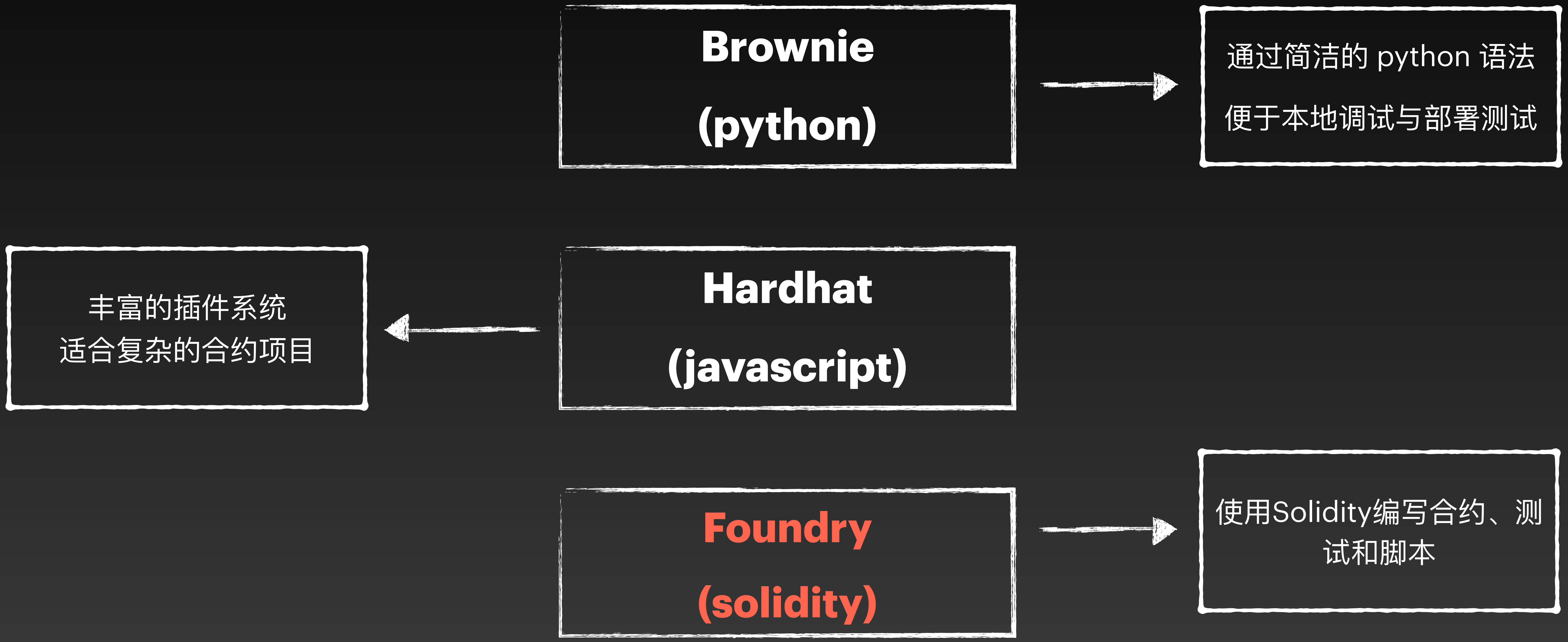
也是入门区块链与 web3 所必须掌握的语言

如何部署智能合约



Solidity 合约是以 **.sol** 为后缀的文件，无法直接执行，需要编译为 **EVM**（**Ethereum Virtual Machine**）可识别的字节码才能在链上运行。

开发框架 & 工具



Brownie

(python)

通过简洁的 python 语法

便于本地调试与部署测试

Hardhat

(javascript)

丰富的插件系统

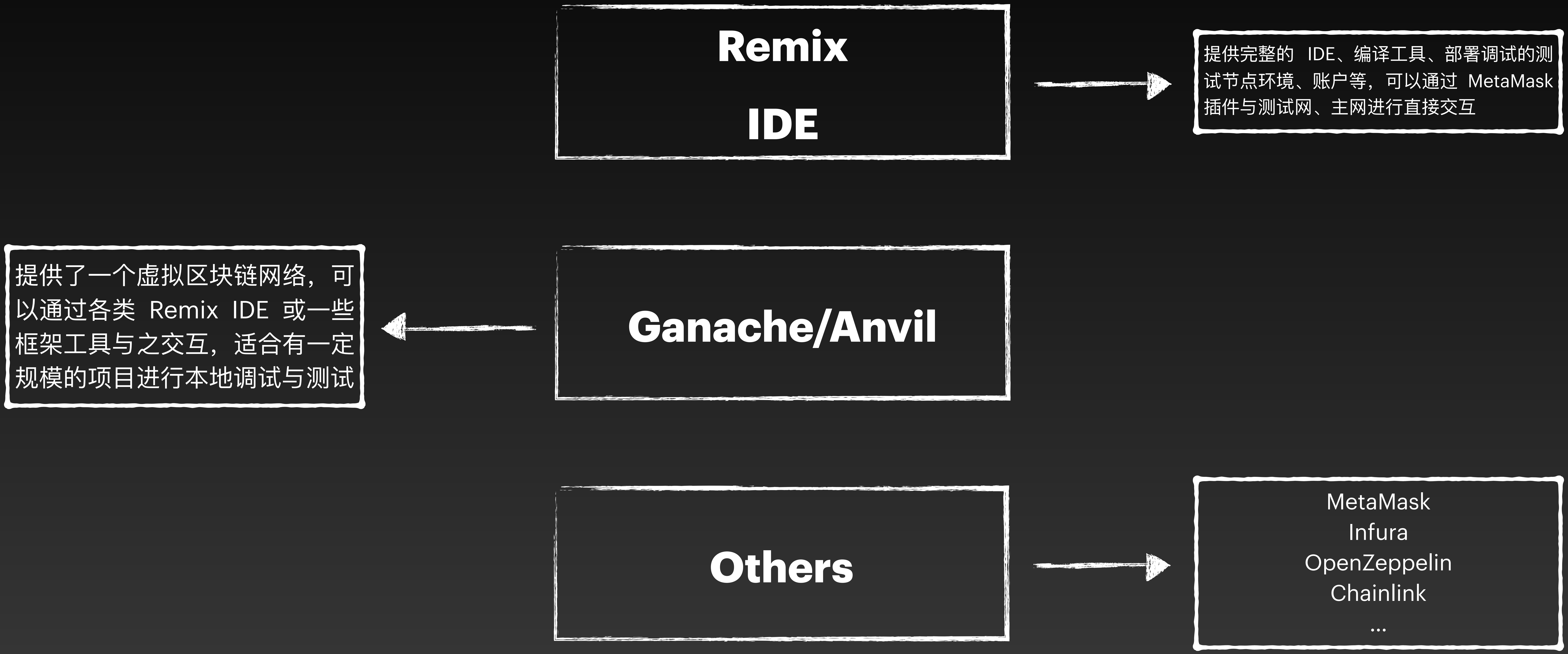
适合复杂的合约项目

Foundry

(solidity)

使用Solidity编写合约、测试和脚本

开发框架 & 工具



Remix IDE

提供完整的 IDE、编译工具、部署调试的测试节点环境、账户等，可以通过 MetaMask 插件与测试网、主网进行直接交互

Ganache/Anvil

提供了一个虚拟区块链网络，可以通过各类 Remix IDE 或一些框架工具与之交互，适合有一定规模的项目进行本地调试与测试

Others

MetaMask
Infura
OpenZeppelin
Chainlink
...

Solidity 核心语法讲解

数据类型

基本

boolean

int

uint

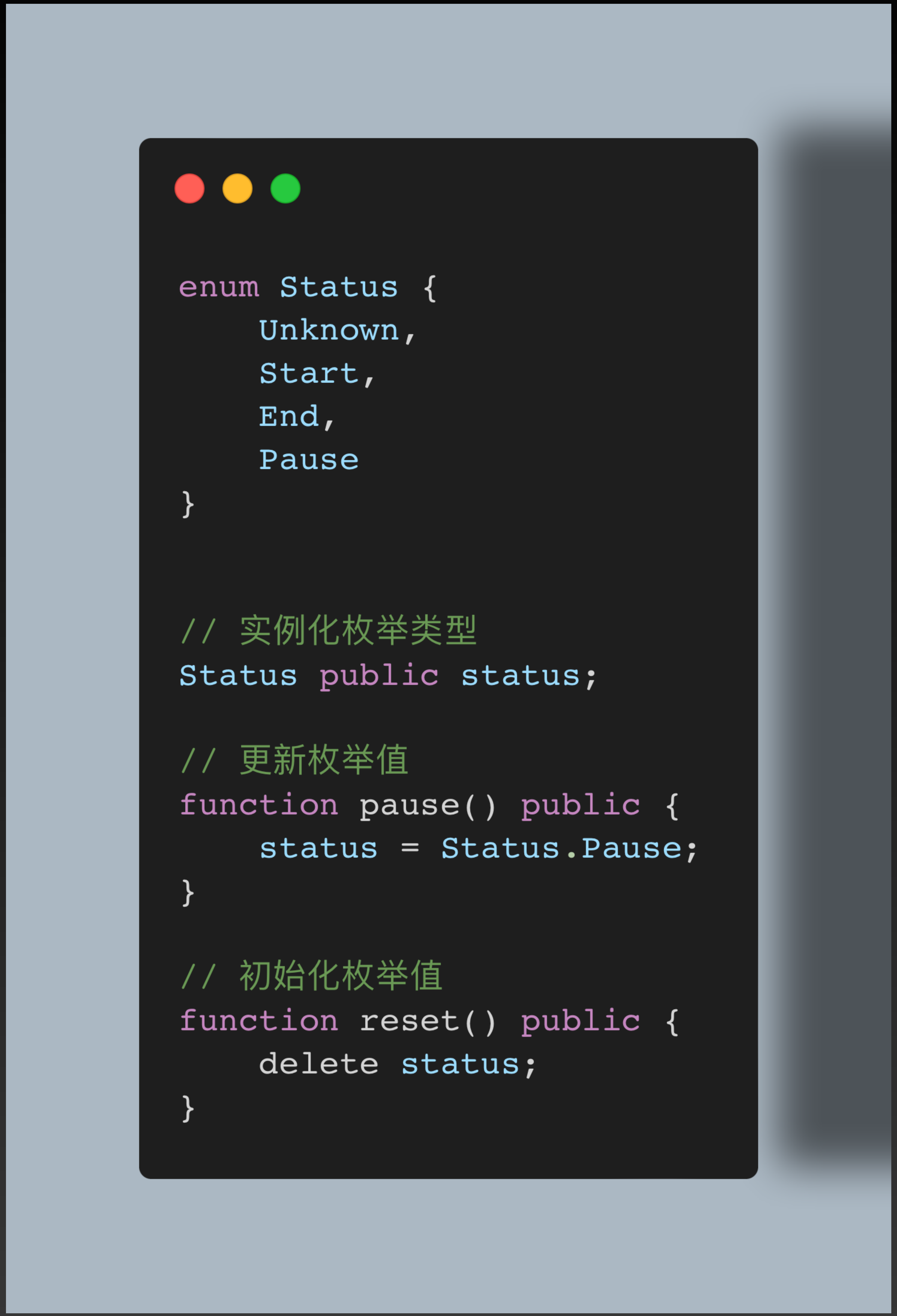
address

bytes

…

数据类型

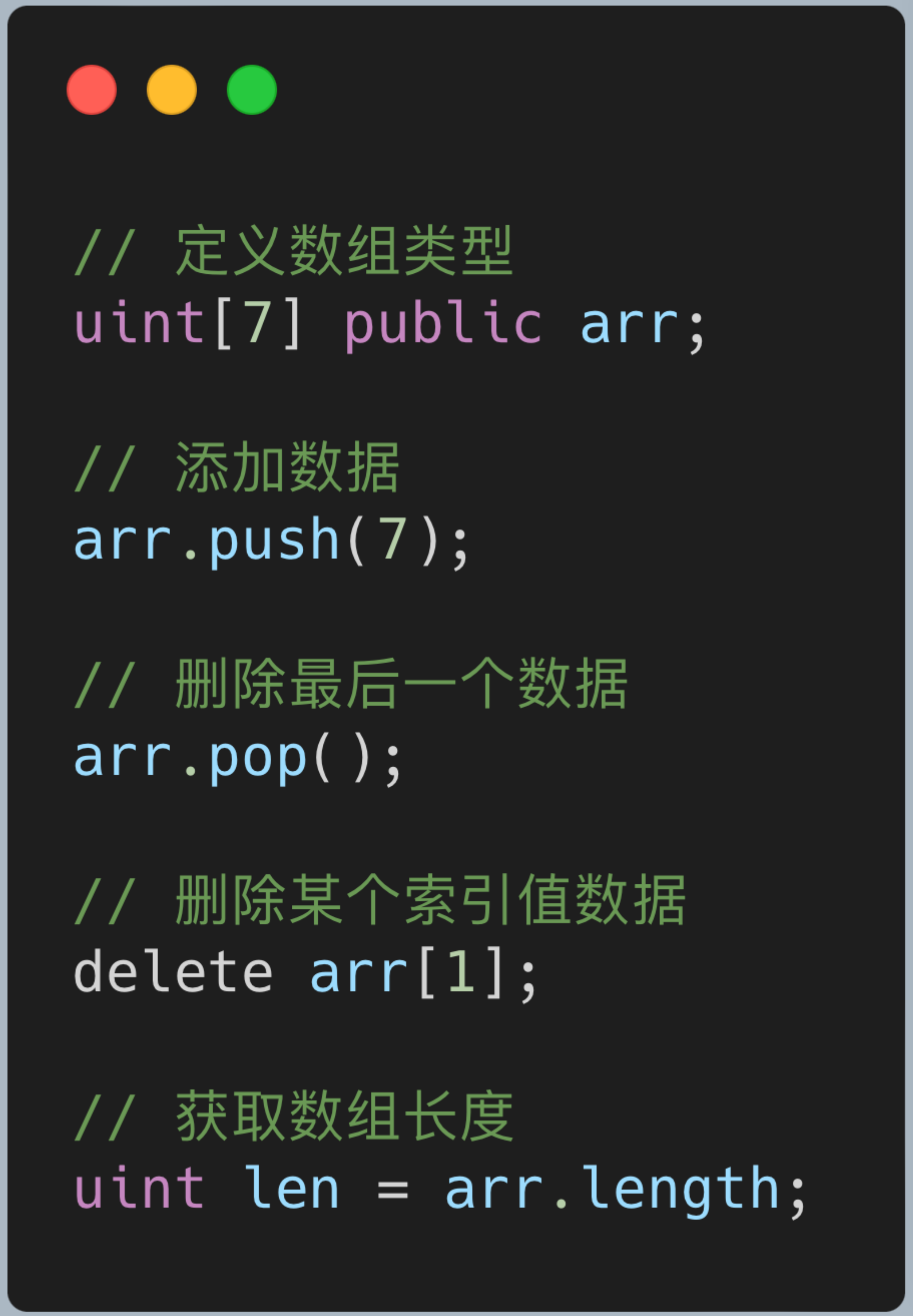
enum



```
enum Status {
    Unknown,
    Start,
    End,
    Pause
}


// 实例化枚举类型
Status public status;

// 更新枚举值
function pause() public {
    status = Status.Pause;
}

// 初始化枚举值
function reset() public {
    delete status;
}
```

数据类型

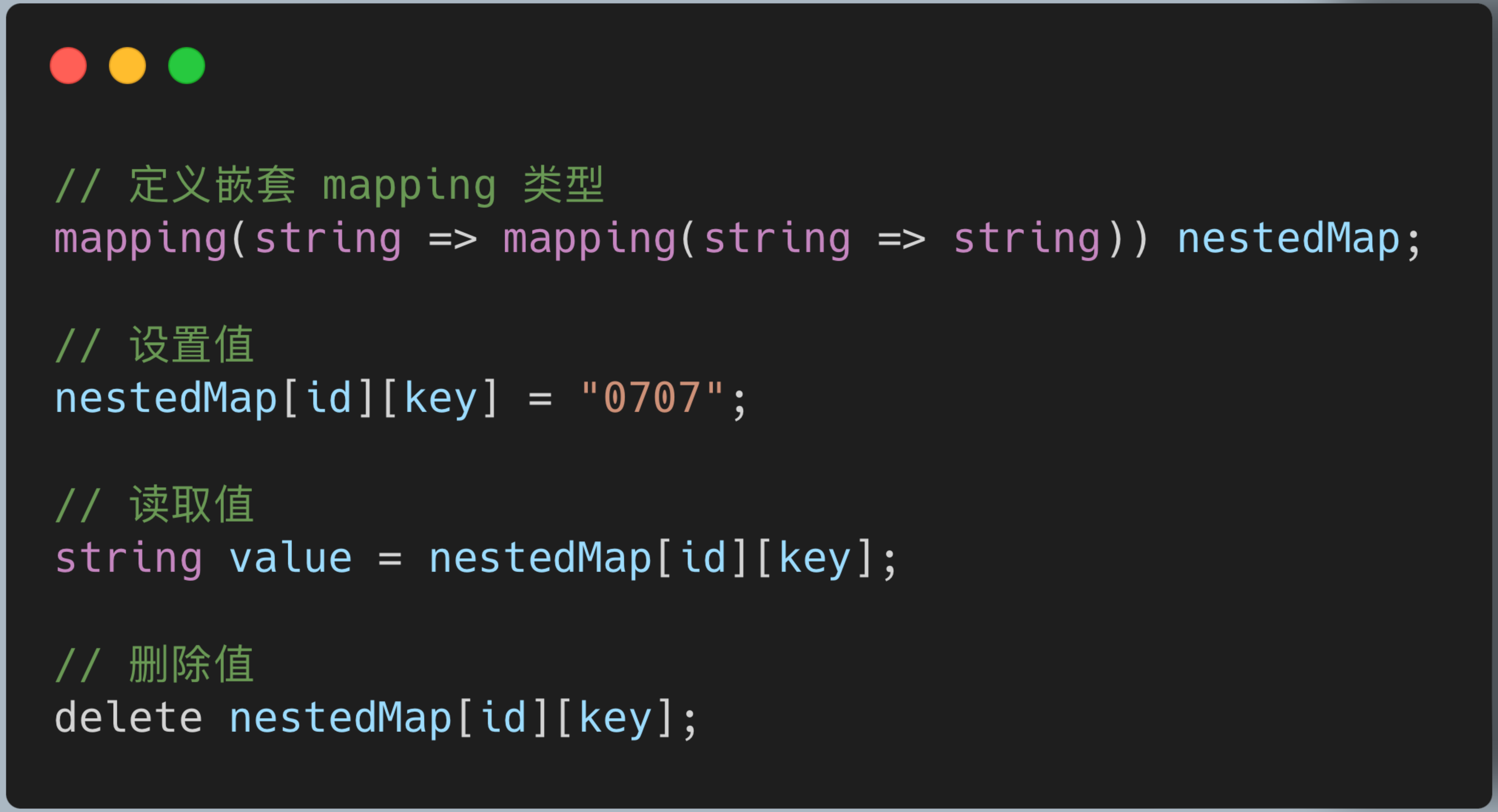
array



```
// 定义数组类型
uint[7] public arr;

// 添加数据
arr.push(7);

// 删除最后一个数据
arr.pop();

// 删除某个索引值数据
delete arr[1];

// 获取数组长度
uint len = arr.length;
```

数据类型

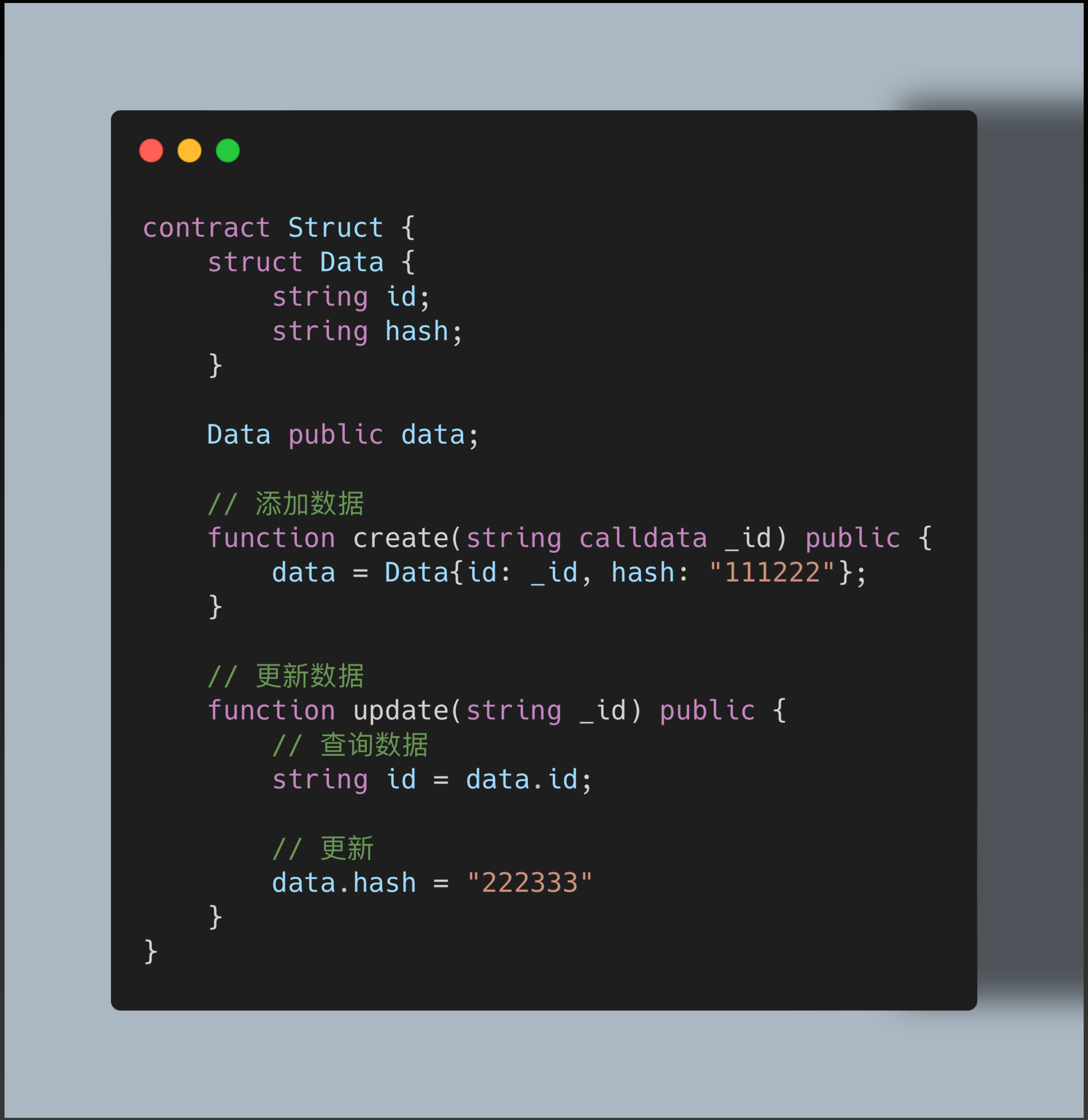
mapping



```
// 定义嵌套 mapping 类型
mapping(string => mapping(string => string)) nestedMap;

// 设置值
nestedMap[id][key] = "0707";

// 读取值
string value = nestedMap[id][key];

// 删除值
delete nestedMap[id][key];
```

数据类型

struct



```
contract Struct {
    struct Data {
        string id;
        string hash;
    }

    Data public data;

    // 添加数据
    function create(string calldata _id) public {
        data = Data{id: _id, hash: "111222"};
    }

    // 更新数据
    function update(string _id) public {
        // 查询数据
        string id = data.id;

        // 更新
        data.hash = "222333"
    }
}
```

变量

类型

local

state

global

关键字声明

storage

memory

calldata

constant / immutable

constant

值不可变

节约 gas fee

immutable

可以在 constructor 中初始化

但不可以再次改变

函数

可见性与关键字

可见性

public

private

internal

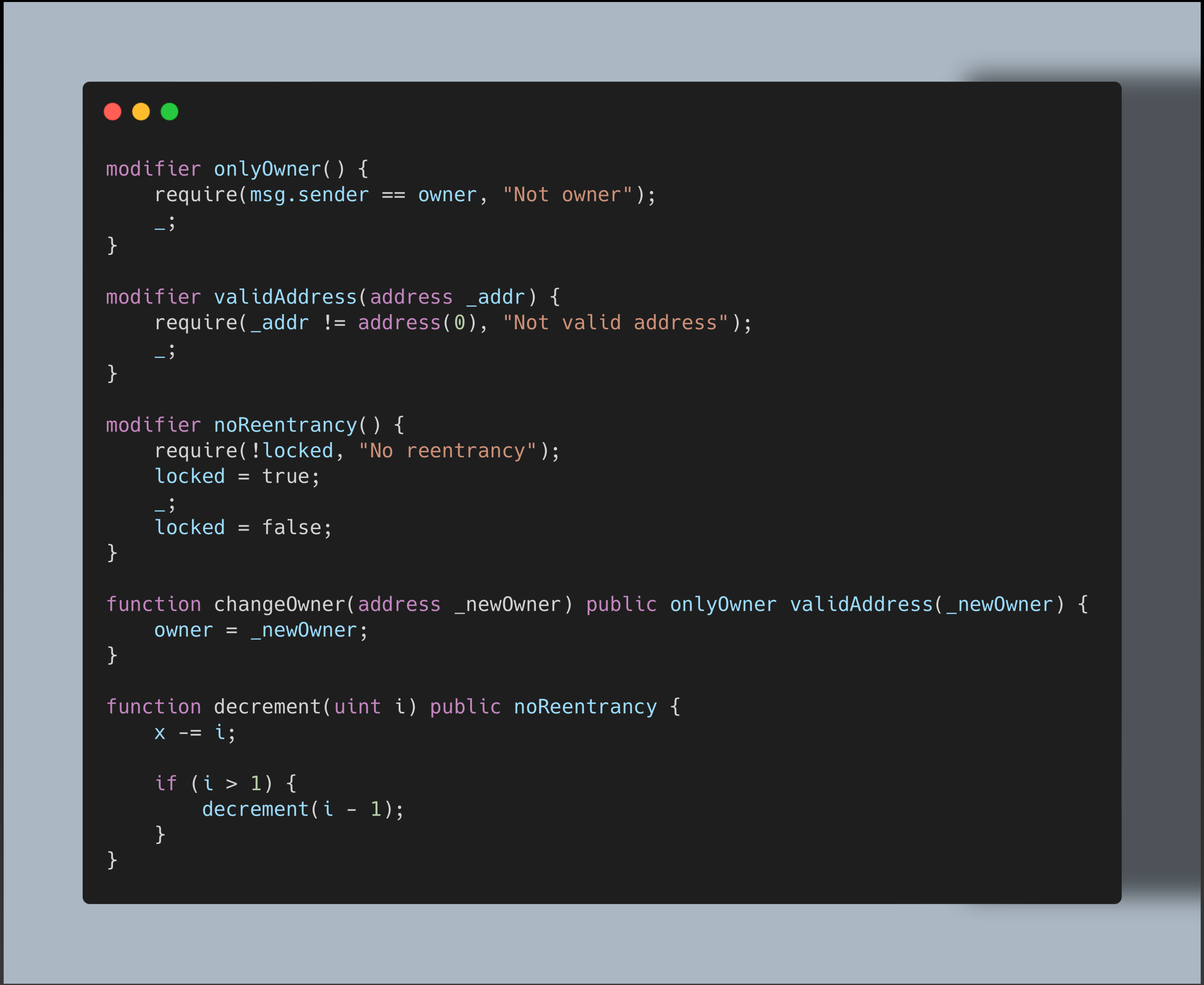
external

关键字

view

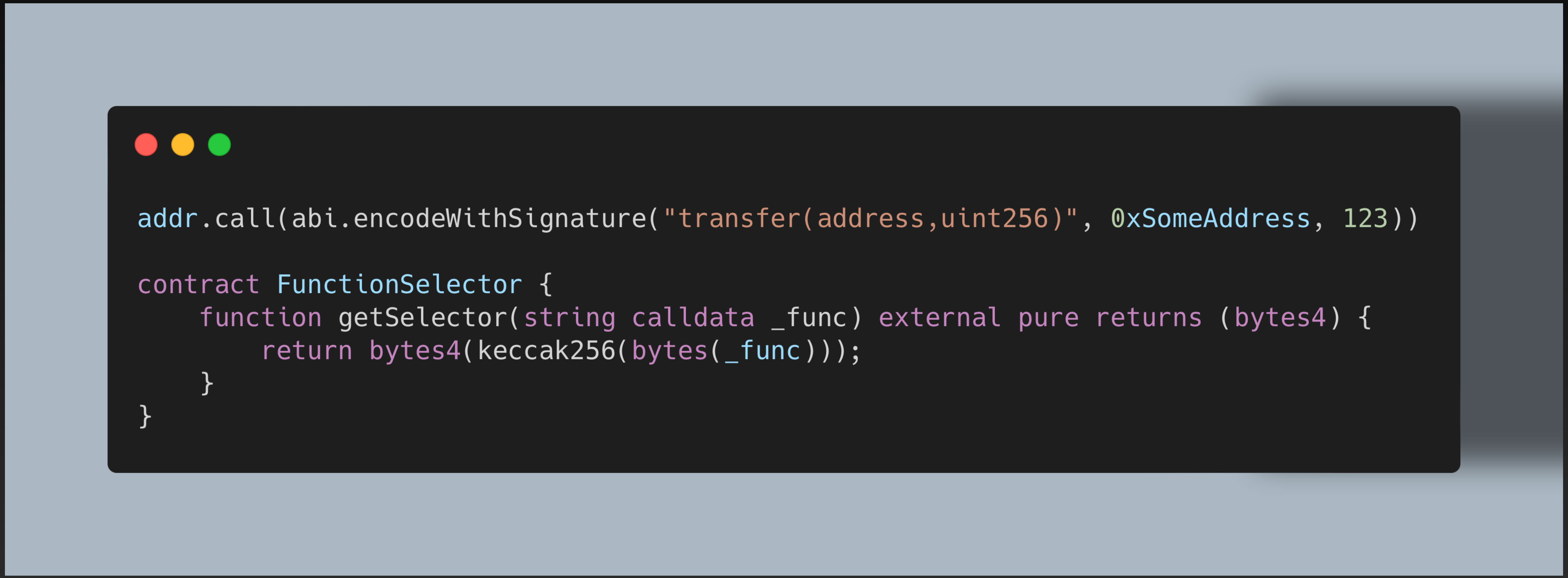
pure

函数
modifier



```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

modifier validAddress(address _addr) {
    require(_addr != address(0), "Not valid address");
    _;
}

modifier noReentrancy() {
    require(!locked, "No reentrancy");
    locked = true;
    _;
    locked = false;
}

function changeOwner(address _newOwner) public onlyOwner validAddress(_newOwner) {
    owner = _newOwner;
}

function decrement(uint i) public noReentrancy {
    x -= i;

    if (i > 1) {
        decrement(i - 1);
    }
}
```

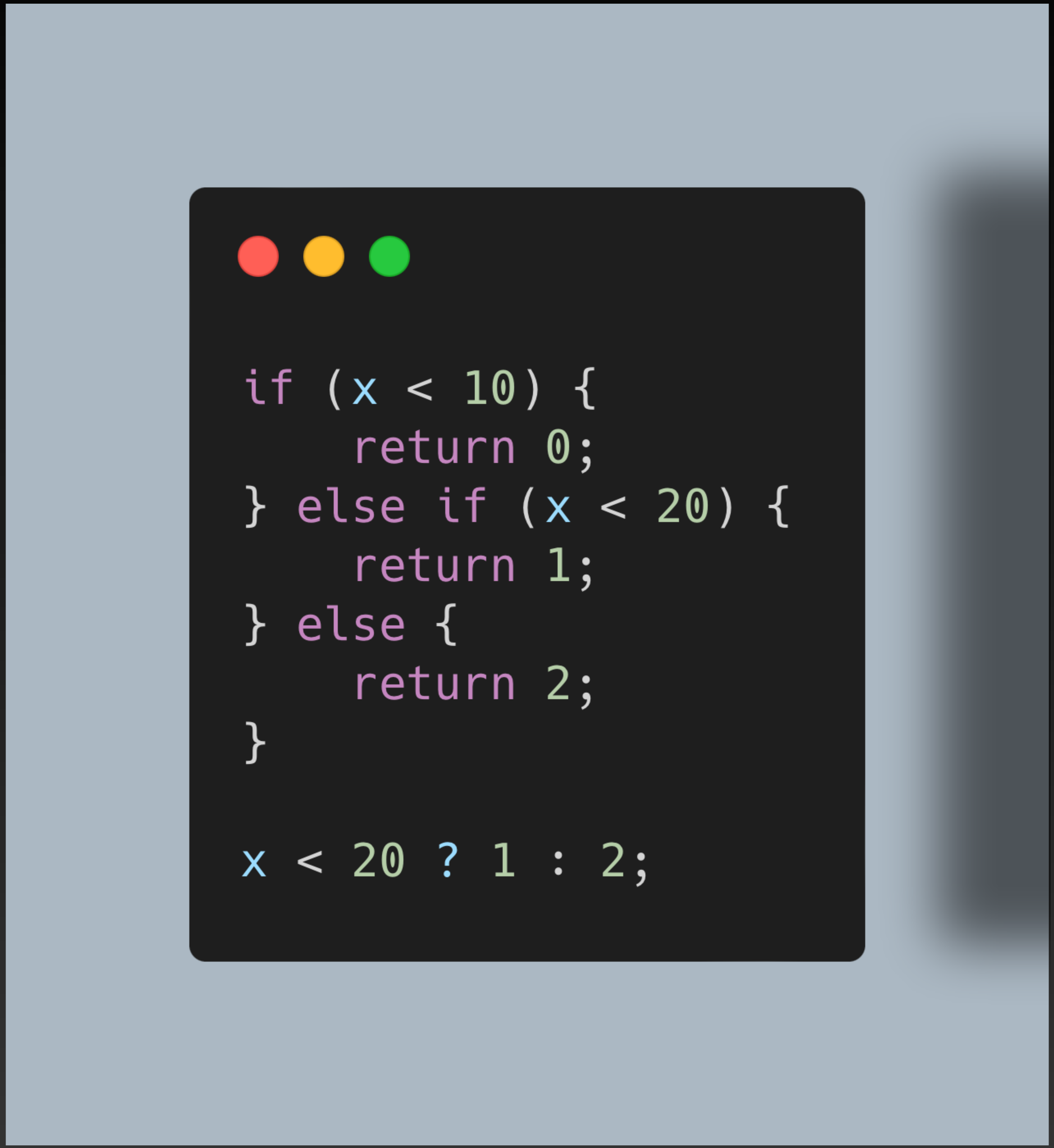
函数选择器



```
addr.call(abi.encodeWithSignature("transfer(address,uint256)", 0xSomeAddress, 123))

contract FunctionSelector {
    function getSelector(string calldata _func) external pure returns (bytes4) {
        return bytes4(keccak256(bytes(_func)));
    }
}
```

条件

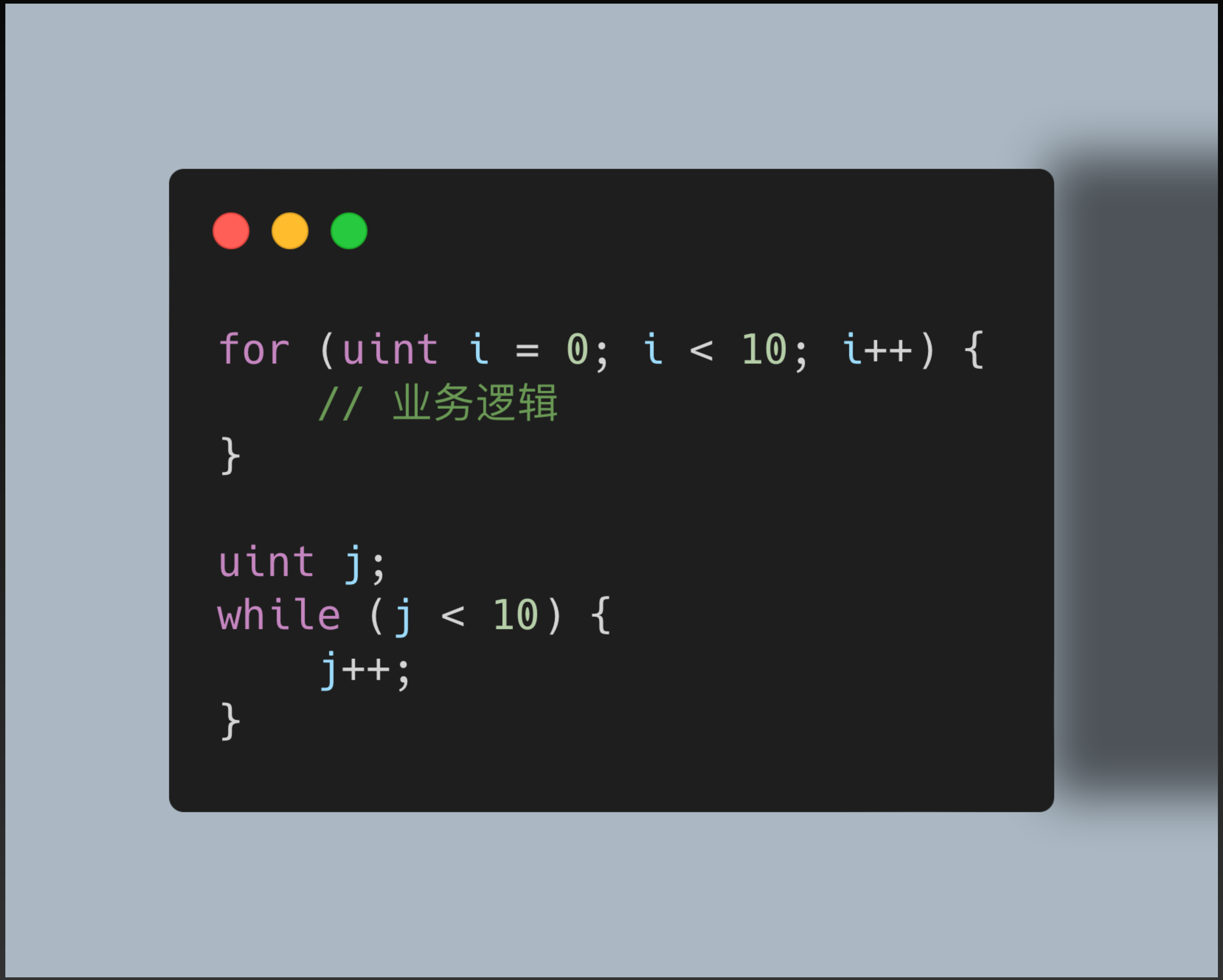
if / else if / else



```
if (x < 10) {
    return 0;
} else if (x < 20) {
    return 1;
} else {
    return 2;
}

x < 20 ? 1 : 2;
```

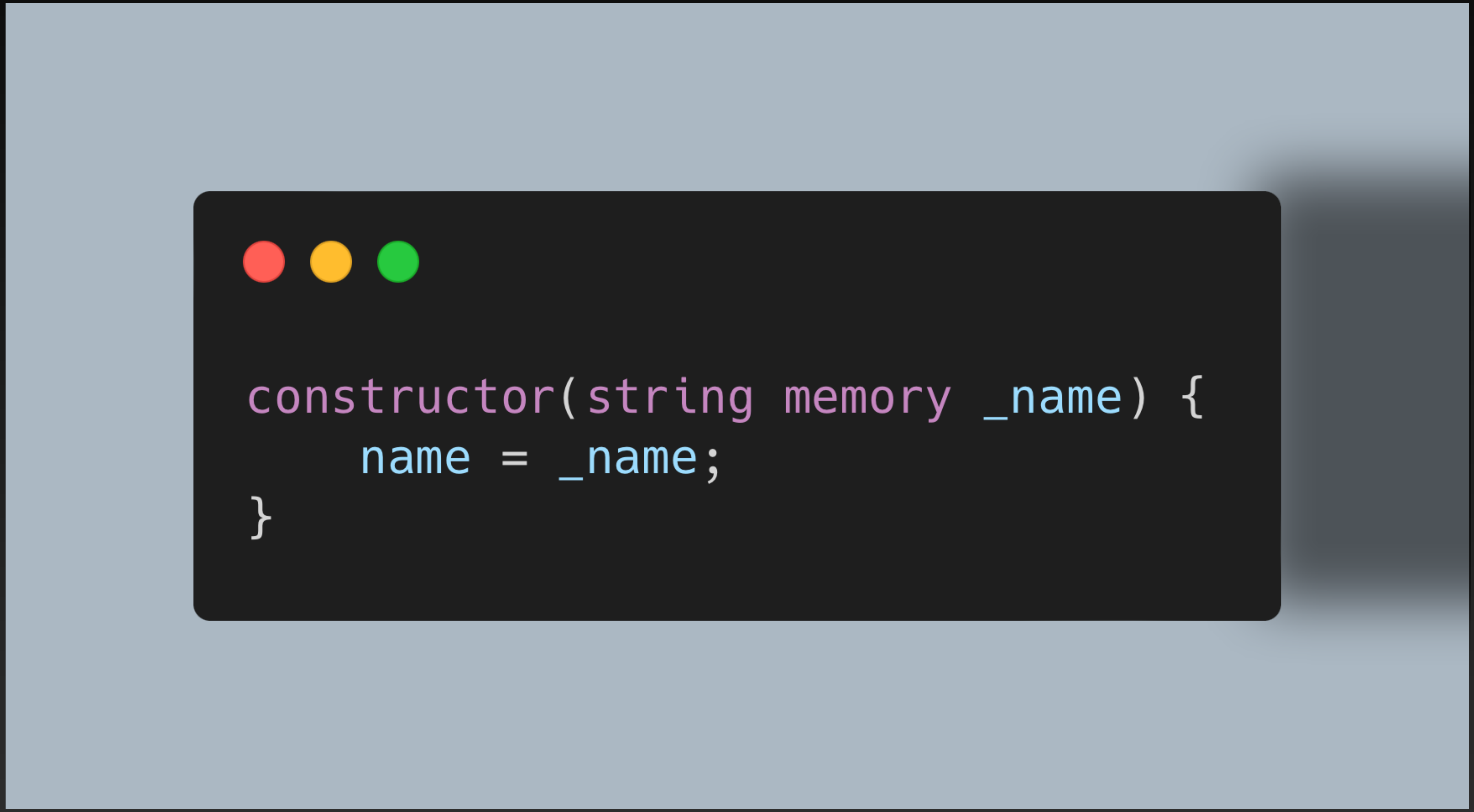
循环

for / while / do while



```
for (uint i = 0; i < 10; i++) {
    // 业务逻辑
}

uint j;
while (j < 10) {
    j++;
}
```

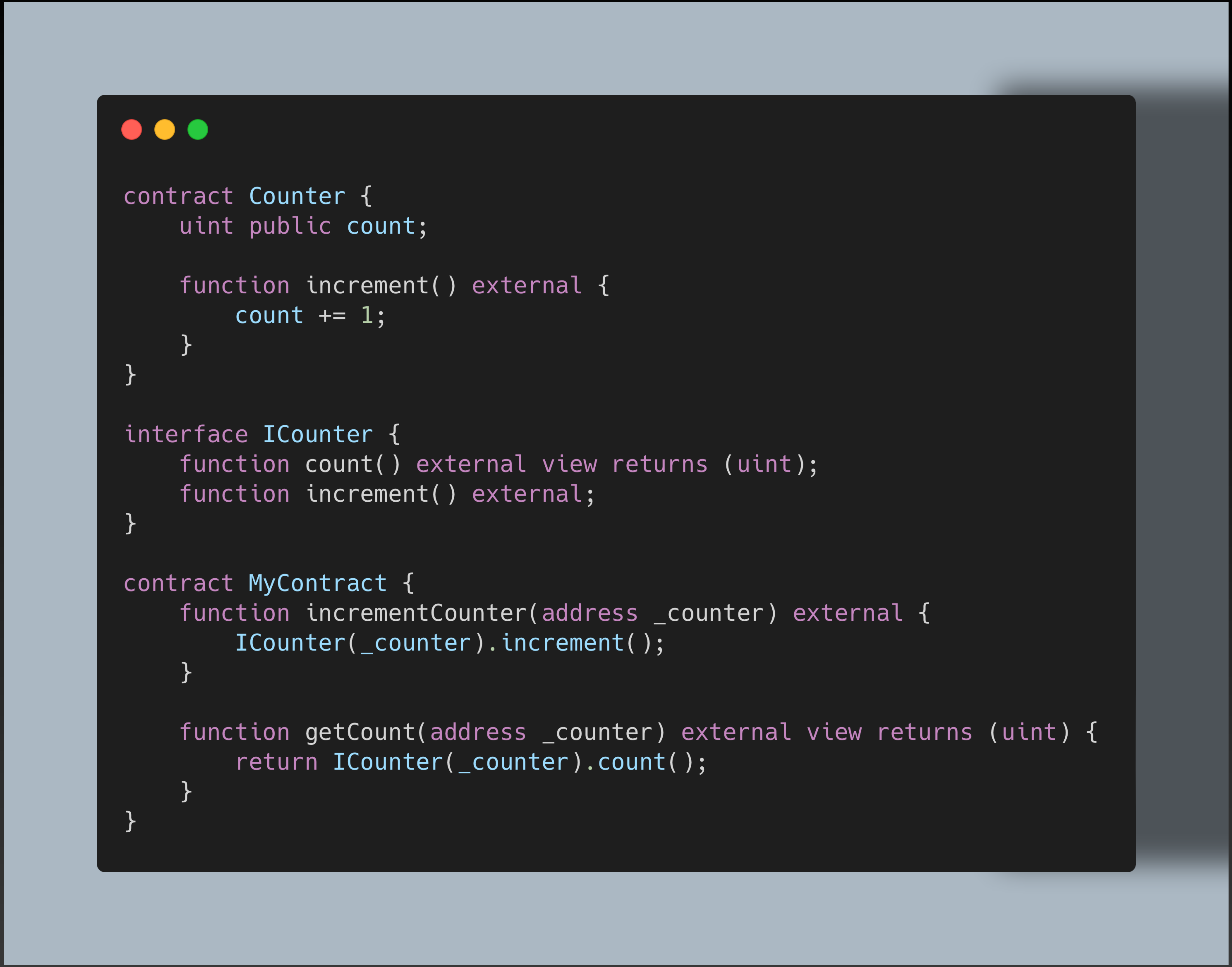
合约
constructor



```
constructor(string memory _name) {
    name = _name;
}
```

合约

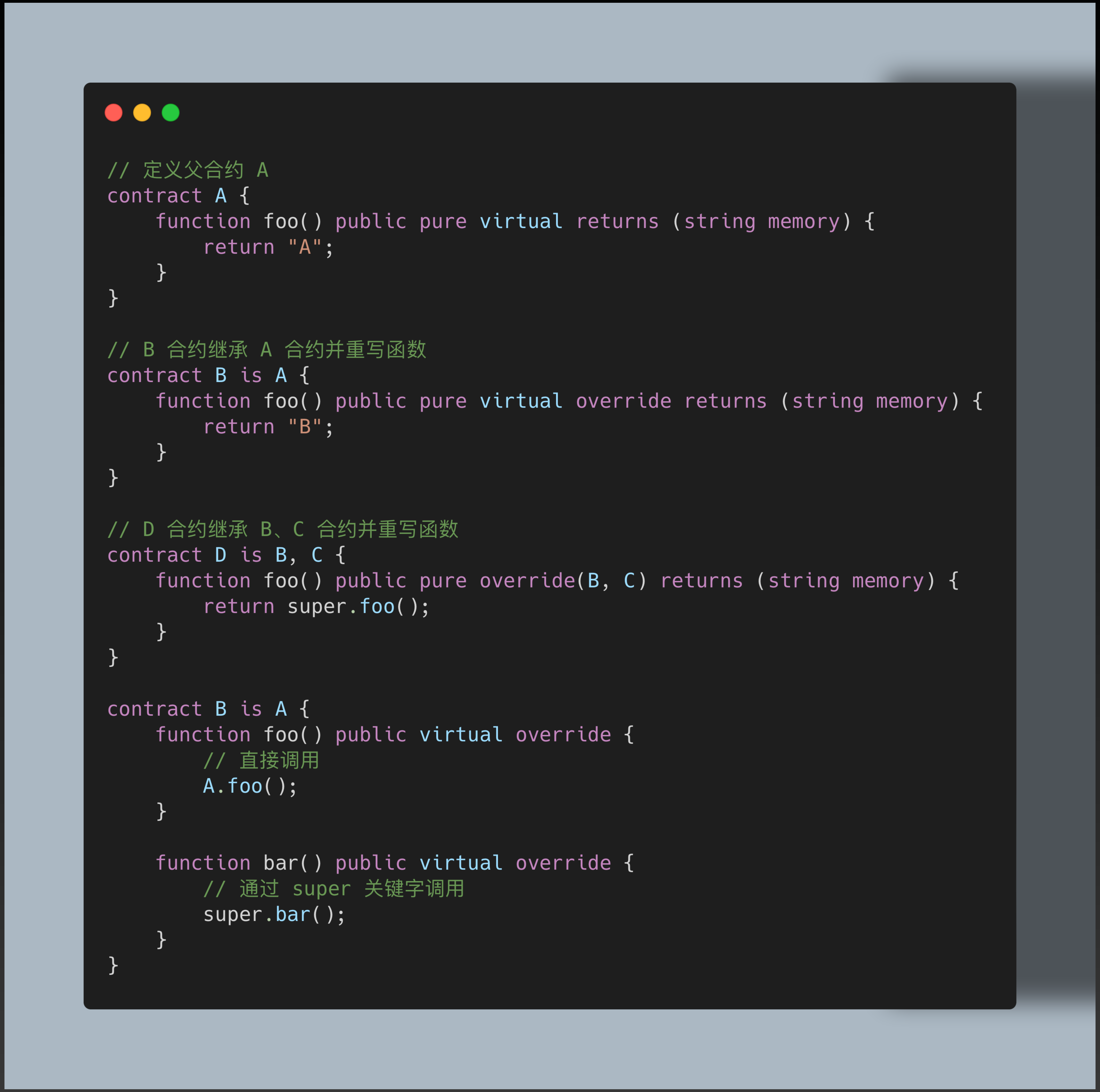
interface



```
contract Counter {
    uint public count;

    function increment() external {
        count += 1;
    }
}

interface ICounter {
    function count() external view returns (uint);
    function increment() external;
}

contract MyContract {
    function incrementCounter(address _counter) external {
        ICounter(_counter).increment();
    }

    function getCount(address _counter) external view returns (uint) {
        return ICounter(_counter).count();
    }
}
```

合约

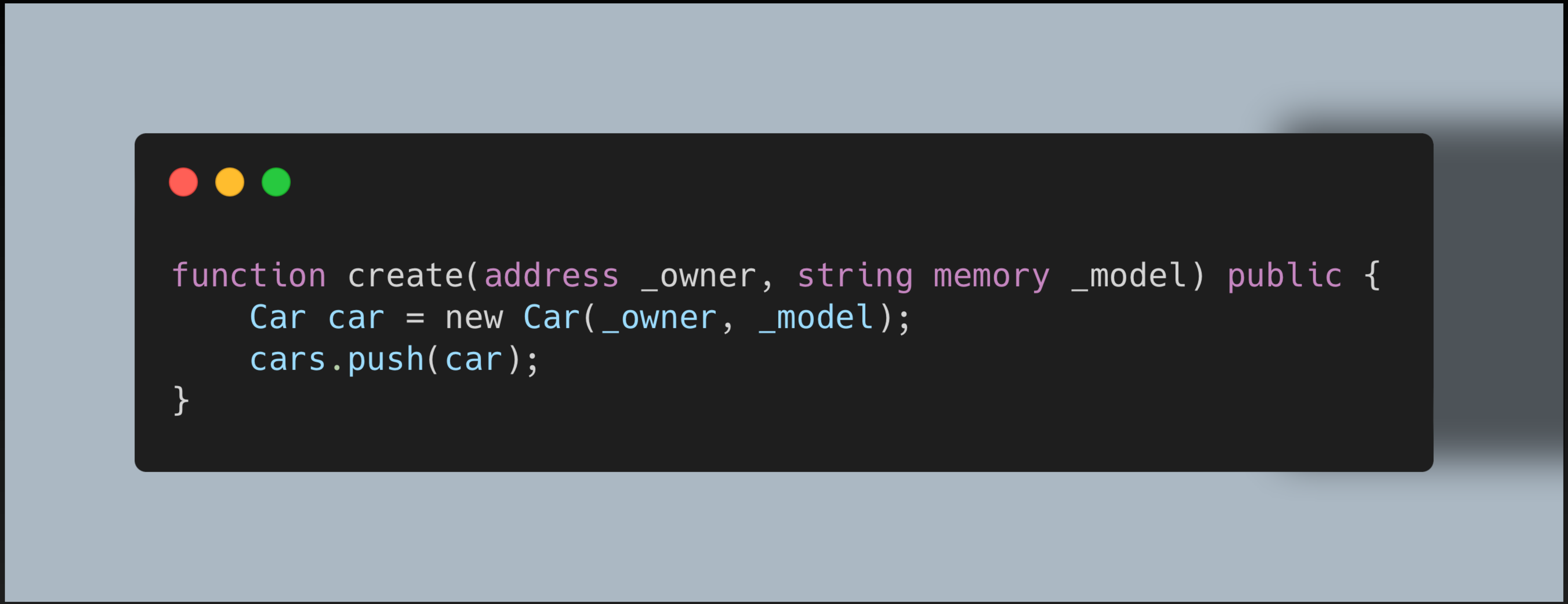
继承



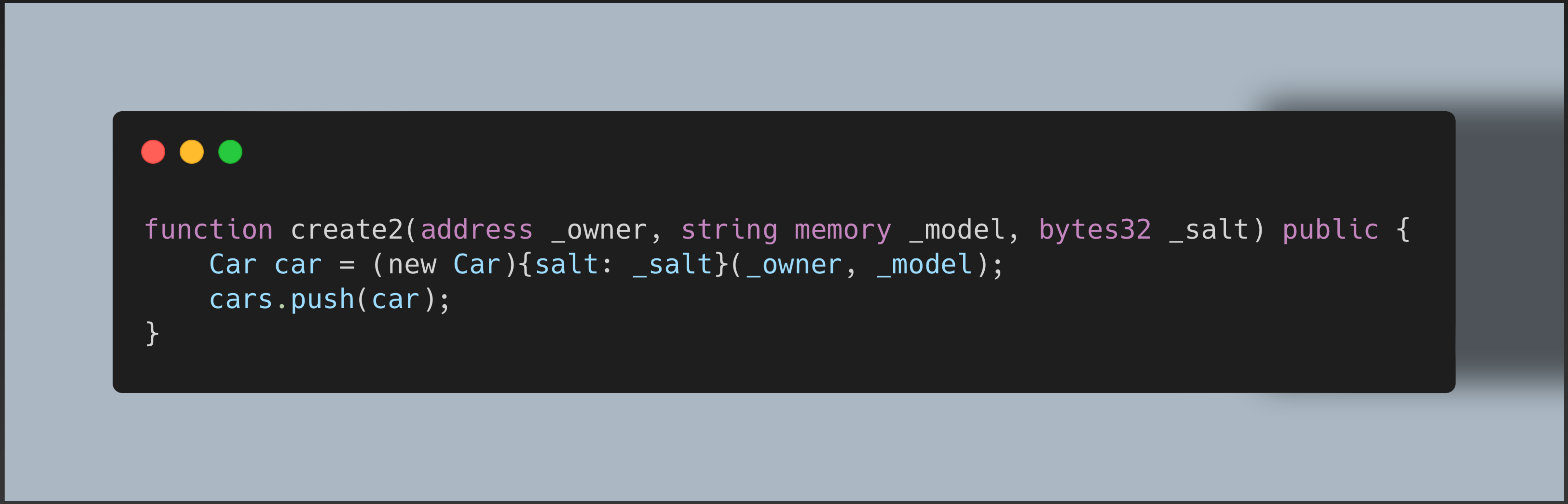
```
// 定义父合约 A
contract A {
    function foo() public pure virtual returns (string memory) {
        return "A";
    }
}

// B 合约继承 A 合约并重写函数
contract B is A {
    function foo() public pure virtual override returns (string memory) {
        return "B";
    }
}

// D 合约继承 B、C 合约并重写函数
contract D is B, C {
    function foo() public pure override(B, C) returns (string memory) {
        return super.foo();
    }
}

contract B is A {
    function foo() public virtual override {
        // 直接调用
        A.foo();
    }

    function bar() public virtual override {
        // 通过 super 关键字调用
        super.bar();
    }
}
```

合约

创建



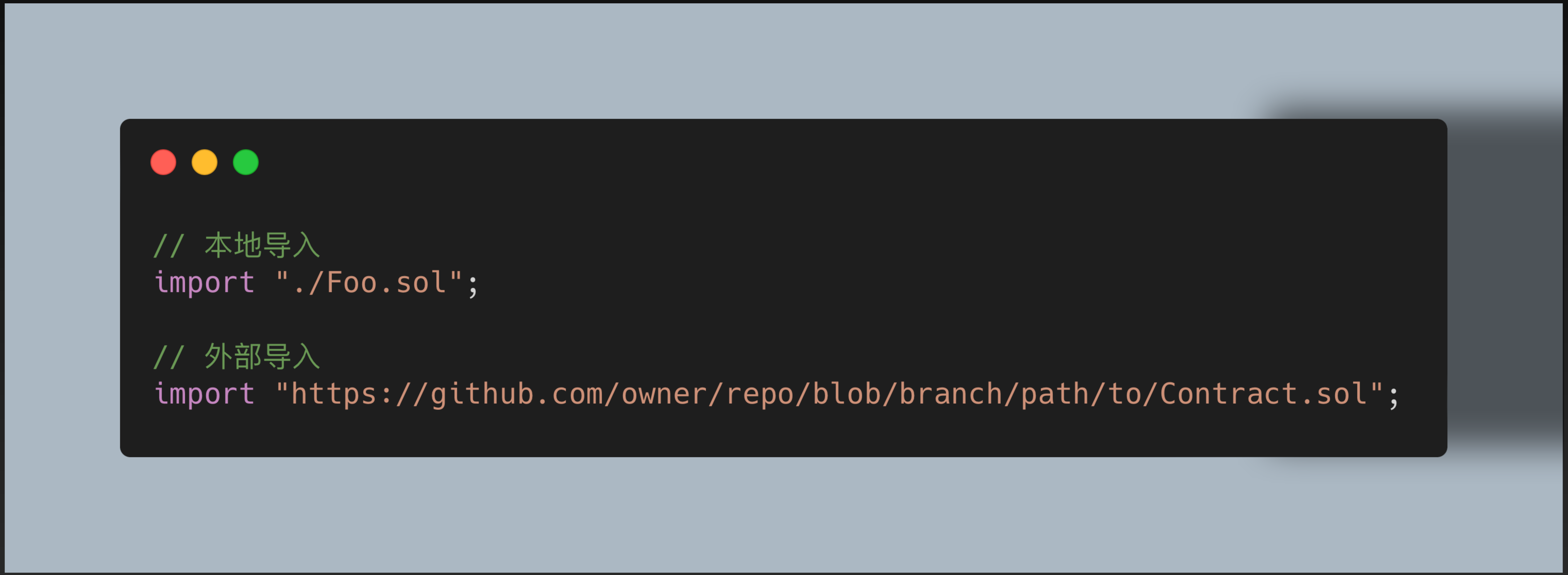
```
function create(address _owner, string memory _model) public {
    Car car = new Car(_owner, _model);
    cars.push(car);
}
```



```
function create2(address _owner, string memory _model, bytes32 _salt) public {
    Car car = (new Car){salt: _salt}(_owner, _model);
    cars.push(car);
}
```

合约

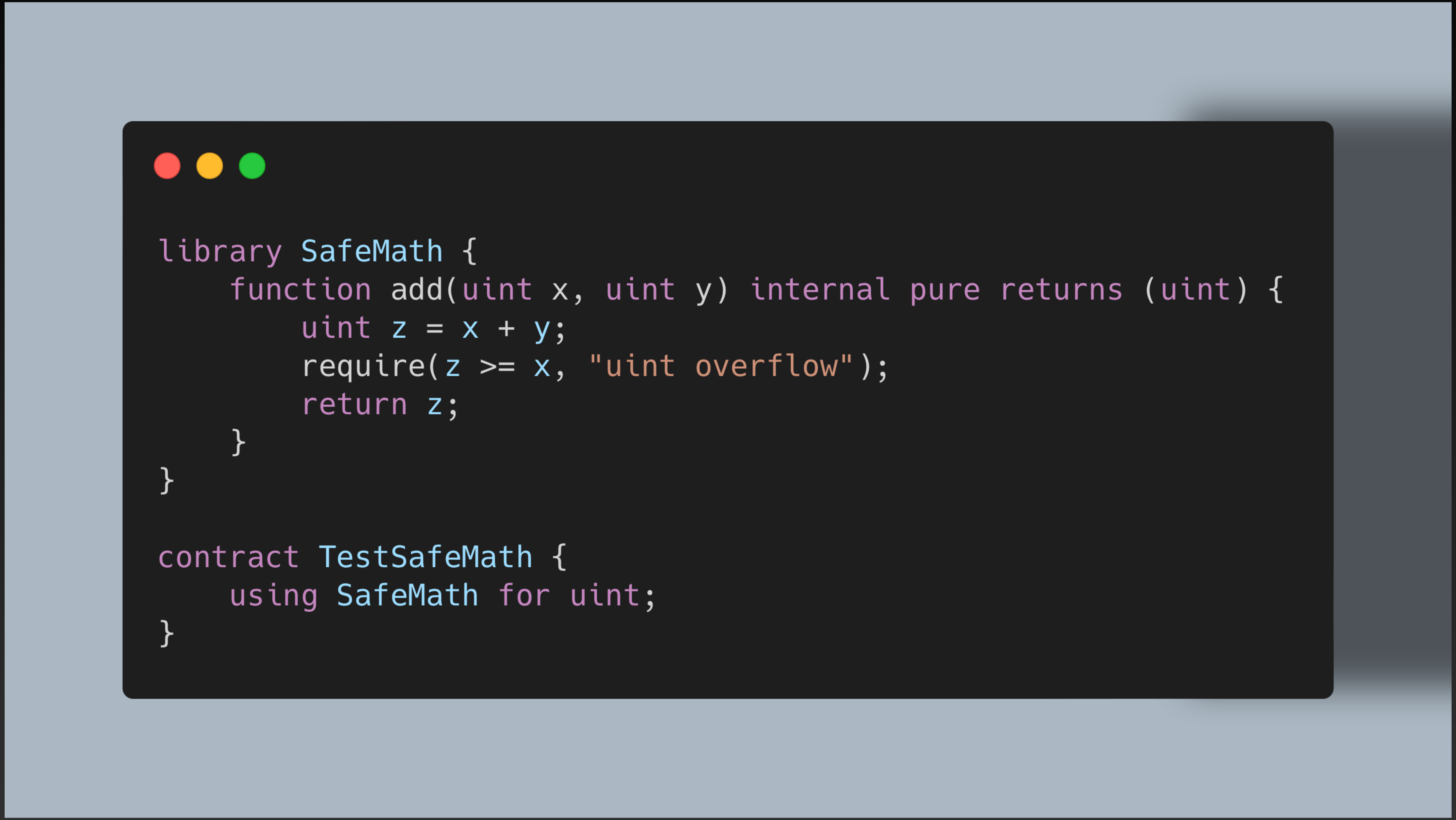
导入合约



```
// 本地导入
import "./Foo.sol";

// 外部导入
import "https://github.com/owner/repo/blob/branch/path/to/Contract.sol";
```

合约

导入库



```
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint) {
        uint z = x + y;
        require(z >= x, "uint overflow");
        return z;
    }
}

contract TestSafeMath {
    using SafeMath for uint;
}
```

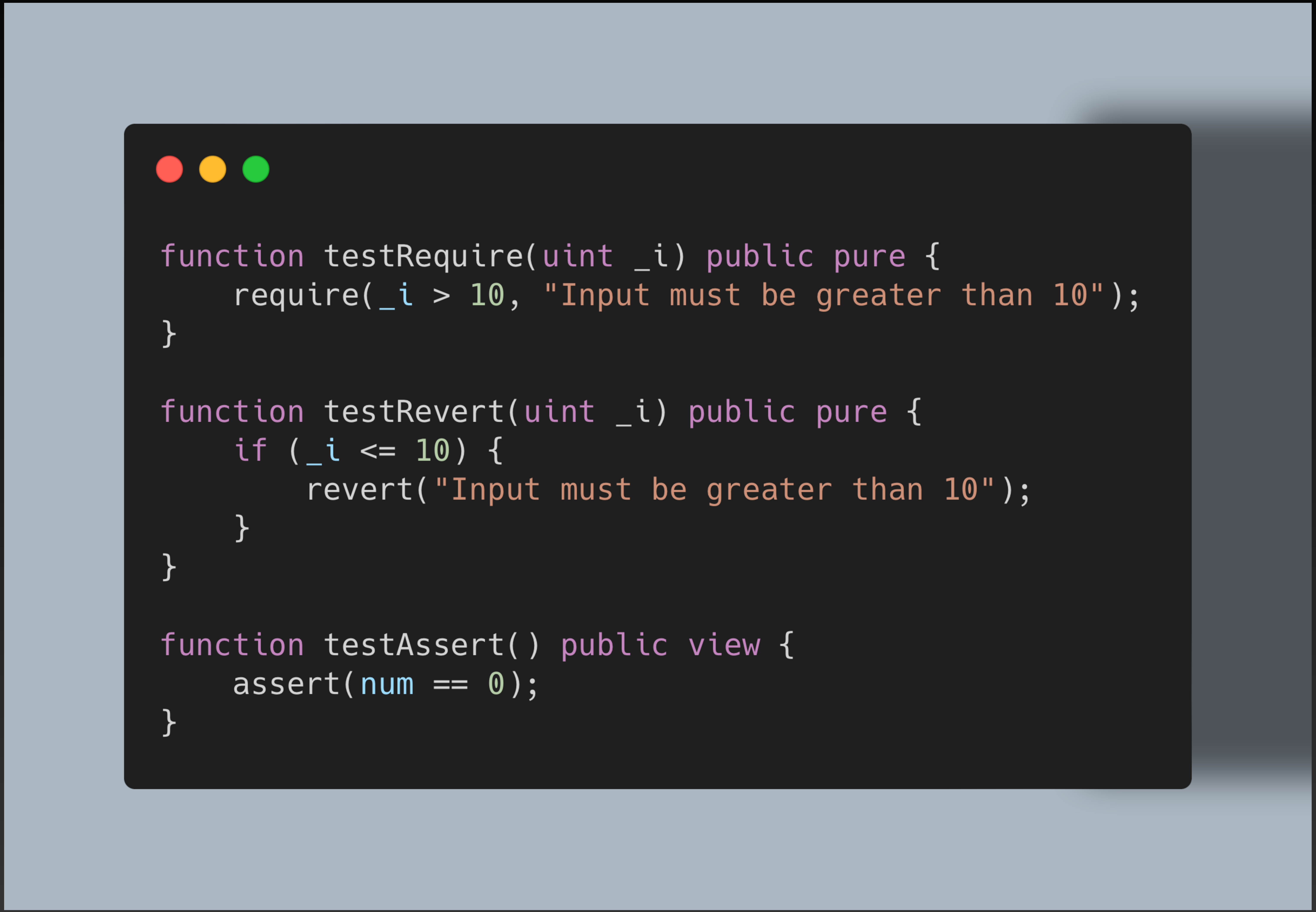
事件

event

```
// 定义事件
event Log(address indexed sender, string message);
event AnotherLog();

// 抛出事件
emit Log(msg.sender, "Hello World!");
emit Log(msg.sender, "Hello EVM!");
emit AnotherLog();
```

错误处理

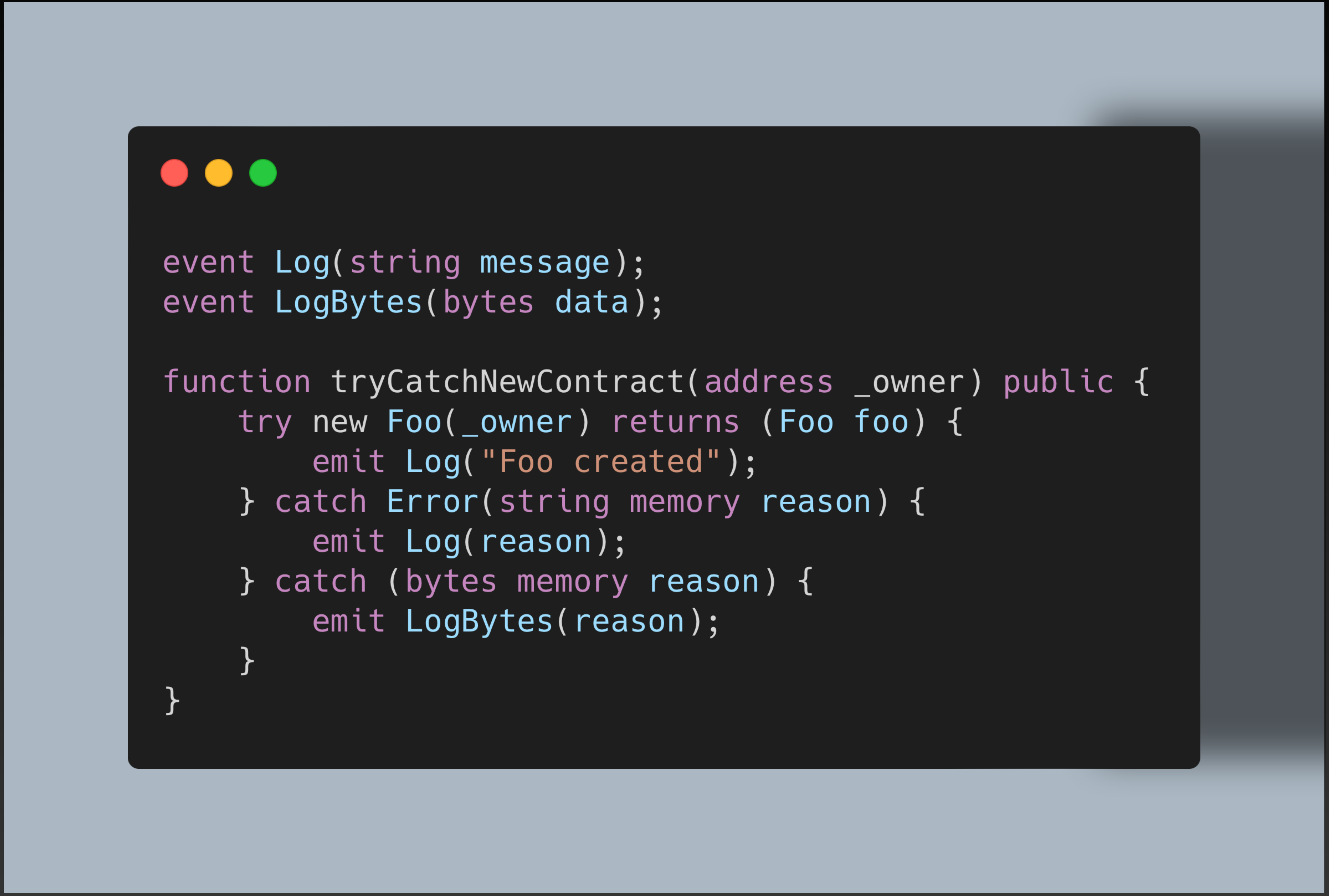
require / revert / assert



```
function testRequire(uint _i) public pure {
    require(_i > 10, "Input must be greater than 10");
}

function testRevert(uint _i) public pure {
    if (_i <= 10) {
        revert("Input must be greater than 10");
    }
}

function testAssert() public view {
    assert(num == 0);
}
```

错误处理

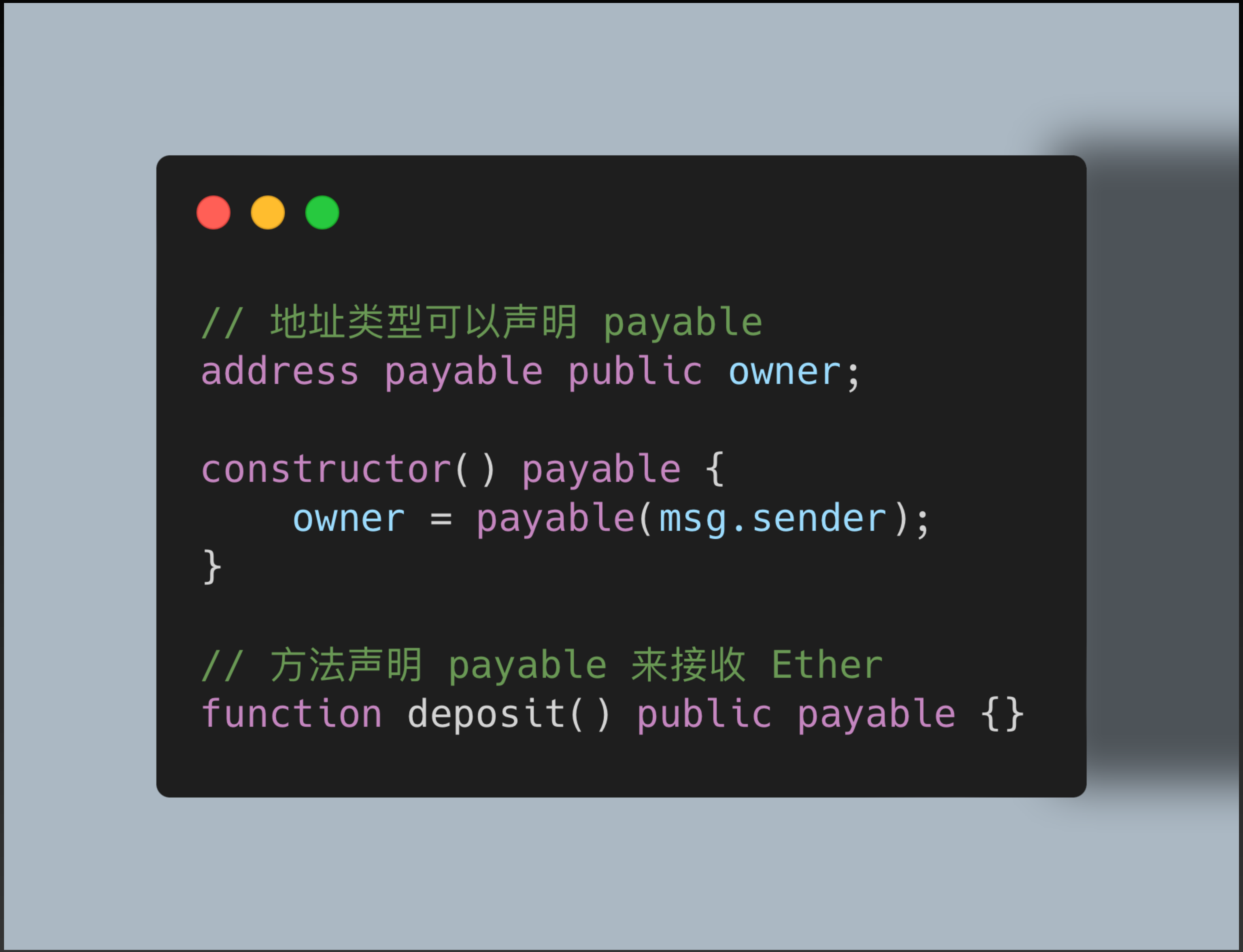
try / catch



```
event Log(string message);
event LogBytes(bytes data);

function tryCatchNewContract(address _owner) public {
    try new Foo(_owner) returns (Foo foo) {
        emit Log("Foo created");
    } catch Error(string memory reason) {
        emit Log(reason);
    } catch (bytes memory reason) {
        emit LogBytes(reason);
    }
}
```

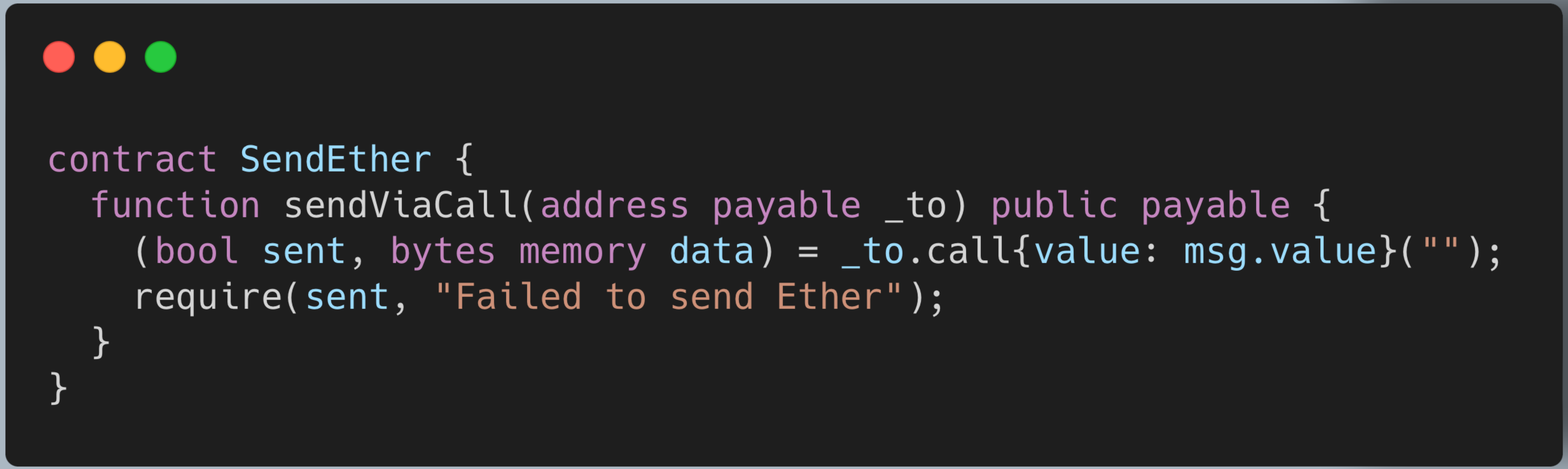
资产

payable



```
// 地址类型可以声明 payable
address payable public owner;

constructor() payable {
    owner = payable(msg.sender);
}

// 方法声明 payable 来接收 Ether
function deposit() public payable {}
```

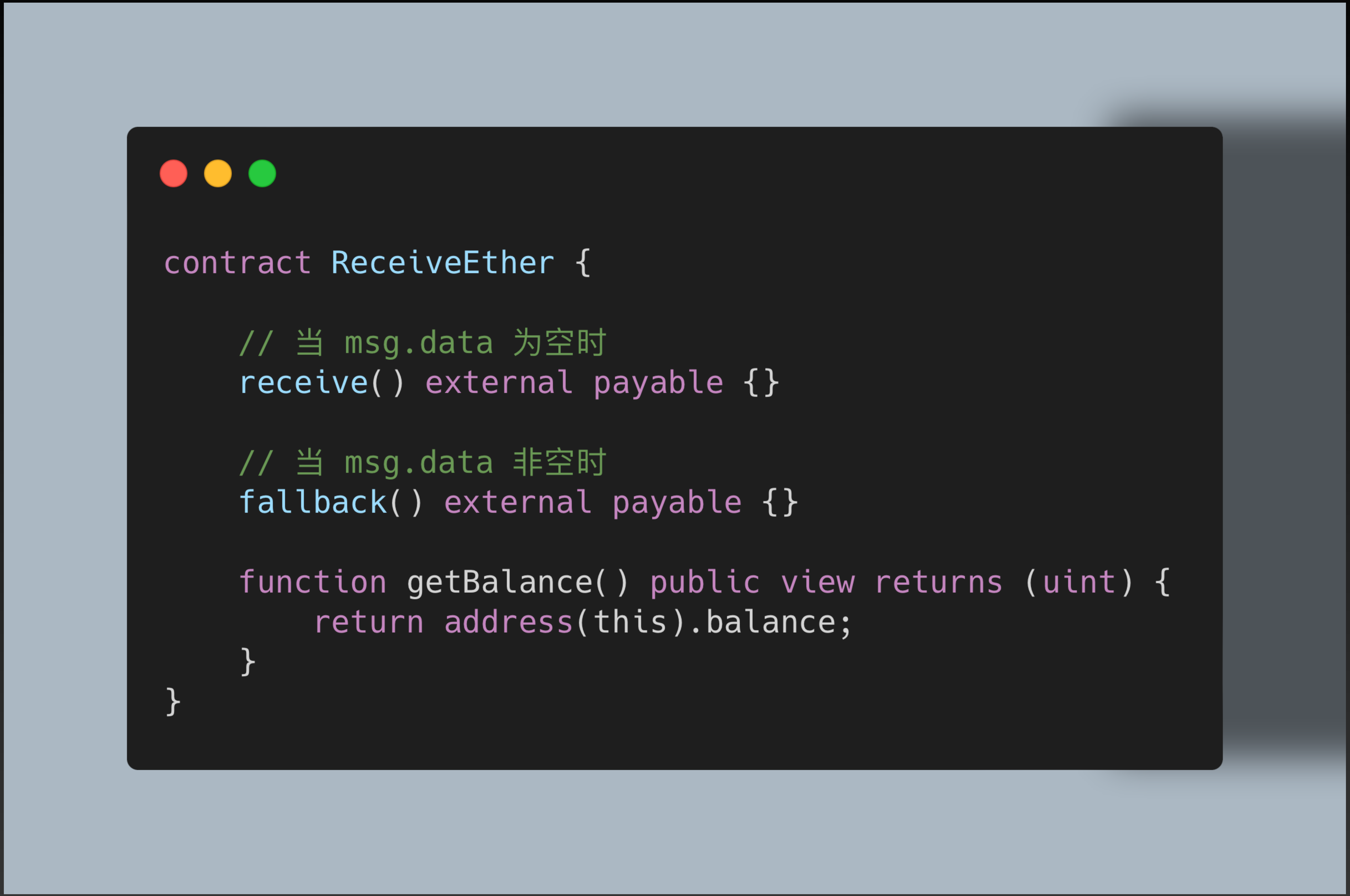
资产
发送



```
contract SendEther {
  function sendViaCall(address payable _to) public payable {
    (bool sent, bytes memory data) = _to.call{value: msg.value}("");
    require(sent, "Failed to send Ether");
  }
}
```

资产

接收



```solidity
contract ReceiveEther {

    // 当 msg.data 为空时
    receive() external payable {}

    // 当 msg.data 非空时
    fallback() external payable {}

    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

资产

Gas Fee

参数

gas spent

gas price

gas limit

block gas limit

技巧

使用 calldata 替换 memory

将状态变量载入内存

使用 i++ 而不是 ++i

缓存数组元素

...

References

Demo code repository

https://github.com/pseudoyu/social-dApp-demo-contracts

Solidity Smart Contract Development - Basics

https://www.pseudoyu.com/zh/2022/05/25/learn_solidity_from_scratch_basic/

foundry-starter-kit

https://github.com/pseudoyu/foundry-starter-kit

Other resources

Solidity by Example

Learn Solidity, Blockchain Development, & Smart Contracts

Thank you!

emit Log(msg.sender, “Thank You”);

pseudoyu | November 26, 2022