

1、Solidity 智能合约结构

代码如下：

```
1.sol  X
1  contract SimpleStorage {
2      uint256 private _value; // 状态变量
3
4      // 设置值（写操作）
5      function setValue(uint256 newValue) public {
6          _value = newValue;
7      }
8
9      // 获取值（读操作）
10     function getValue() public view returns (uint256) {
11         return _value;
12     }
13 }
```

截图如下：

The screenshot displays a Solidity IDE interface. At the top, a file named '1.sol' is open, showing the 'SimpleStorage' contract code. Below the code editor, the 'SimpleStorage' contract is selected in the left sidebar. The main panel shows the contract's details, including its bytecode, ABI, and a list of functions. The 'getValue' function is selected, and its execution results are shown. The 'setValue' function is also listed, and its execution results are shown. The 'getValue' function's output is 'uint256: 100', indicating that the value stored in the contract is 100.

字节码 [部署合约](#)

0x608060405234801561001057600080fd5b506
1044a806100206000396000f300608060405260
0436106049576000357c010000000000000000

合约接口说明 (ABI)

[{ "constant": true, "inputs": [], "name":
"getValue", "outputs": [{ "name": "", "type":
"uint256" }], "payable": false, "stateMutability":

1

合约ID:0x6b86b273ff34fce19d6b804eff5a3f57...
TX Hash:0x4799bc80f244e5679aca835bdc14b...

☒ function getValue [调用合约](#)

tx hash
0x6b1dc53e88b1b6e196bab867733685...

input

output
uint256: 100

log

☒ function setValue [调用合约](#)

tx hash
0x934829ce163381581d7228852317a17...

input
newValue(uint256): 100

output

log

调用 setValue 输入 100 后，调用 getValue 返回值为先前设置的 100

2、整数

代码：

```
2.sol  X
1  contract IntegerOperations {
2      // 2.1 四则运算 (含异常处理)
3      function calculate(uint a, uint b) public pure returns (
4          uint sum,
5          uint diff,
6          uint prod,
7          uint quot
8      ) {
9          sum = a + b;           // 和 (溢出时会自动回滚)
10         diff = a > b ? a - b : 0; // 差 (处理负数情况)
11         prod = a * b;          // 积
12         quot = b > 0 ? a / b : 0; // 商 (除0保护)
13     }
14
15     // 2.2 取余
16     function mod(uint a, uint b) public pure returns (uint) {
17         require(b != 0, "Divisor cannot be zero");
18         return a % b;
19     }
20
21     // 2.3 移位运算
22     function shiftOps(uint x) public pure returns (uint left, uint right) {
23         left = x << 2; // 左移2位 (*4)
24         right = x >> 1; // 右移1位 (/2)
25     }
26 }
```

正常执行结果：

1.计算加减乘除：

function calculate

调用合约

tx hash

0xf9ea4a489cd2d32bd0c1fad2f5af7942...

input

a(uint256): 10

b(uint256): 3

output

uint256: 13

uint256: 7

uint256: 30

uint256: 3

log

2.取余:

function mod

调用合约

tx hash

0xf371e486604271d0061307bd367db6...

input

a(uint256): 10

b(uint256): 3

output

uint256: 1

log

3.: 左移右移

function shiftOps

调用合约

tx hash

0x7824e50ca5f008ea509b7ccddb4d57...

input

x(uint256): 8

output


uint256: 32

uint256: 4

log

异常执行结果：

输入超过 uint256 范围的值后运算发生溢出：

 function calculate

调用合约

tx hash

0x0ae60d7483d322822a86bed578af589...

input

a(uint256): 1.157920892373162e+77

b(uint256): 1.157920892373162e+77

output

uint256: 7.689318425915529e+75

uint256: 6.784692728748996e+75

uint256: 9.624375634234677e+76

uint256: 16

log

3、布尔类型与枚举类型

代码：

```
3.sol  X
1  contract BoolEnum {
2      // 3.1 比较运算
3      function compare(int a, int b) public pure returns (
4          bool gt, bool lt, bool eq
5      ) {
6          gt = a > b;
7          lt = a < b;
8          eq = a == b;
9      }
10
11     // 3.2 枚举类型
12     enum Status { Pending, Approved, Rejected }
13     Status public status = Status.Pending;
14
15     function setStatus(uint _status) public {
16         status = Status(_status); // uint转枚举
17     }
18 }
```

运行：

1.输入状态为 1 后，可以看到发生了默认转换，输出也为 1

3

合约ID:0x4e07408562bedb8b60ce05c1decfe3...

TX Hash:0x941c3e01b44c2266a97fc92dec936f...

function

status

调用合约

tx hash

0x5c9976b1907d0b7945d15507a5b0e9...

input

output

uint8: 1

log

function

setStatus

调用合约

tx hash

0xb5f8113f294d528d65dd512d197e399...


input

_status(uint256): 1

output

log

2.比较 a、b 大小并返回比较结果

 function compare

调用合约

tx hash

0x4724e6c0ee08cd0732ebfdd3f30a96b...

input

a(int256): 1

b(int256): 2

output

bool: false

bool: true

bool: false

log

4、字符串和定长字节数组

代码：

```
1
2  contract BytesOperations {
3      // 不同长度的定长字节数组
4      bytes3 public fixedBytes3;    // 3字节长度
5      bytes5 public fixedBytes5;    // 5字节长度
6      bytes10 public fixedBytes10;  // 10字节长度
7
8      // 设置不同长度的字节数组值
9      function setBytesValues() public {
10         fixedBytes3 = "ABC";        // 3字节
11         fixedBytes5 = "Hello";      // 5字节
12         fixedBytes10 = "Solidity!"; // 9字节（自动填充到10字节）
13     }
14
15     // 获取字节数组值
16     function getBytes() public view returns (bytes3, bytes5, bytes10) {
17         return (fixedBytes3, fixedBytes5, fixedBytes10);
18     }
19
20     // 获取字节数组长度
21     function getBytesLengths() public pure returns (uint, uint, uint) {
22         bytes3 b3 = "123";
23         bytes5 b5 = "abcde";
24         bytes10 b10 = "0123456789";
25
26         return (b3.length, b5.length, b10.length);
27     }
28
29     // 直接转换（编译器自动处理）
30     function directConversion() public pure returns (bytes3) {
31         return bytes3("ETH"); // 转换为3字节数组
32     }
33 }
```


运行截图：

定长字节数组的创建、赋值、取值：



2.字符串直接转换为定长字节数组：



5、地址类型

代码如下：

```
1  contract AddressDemo {
2      // 使用标准地址类型
3      identity myAddress;
4
5      constructor() public { // 添加 public 可见性
6          myAddress = msg.sender;
7      }
8
9      function checkAddress() public view returns (identity) {
10         return myAddress;
11     }
12
13     function isMyAddress(identity addr) public view returns (bool) {
14         return addr == myAddress;
15     }
16 }
```

运行截图如下：

1. 获取地址并比较：

两者都是 0x1a51cbae01b5259ec81af40d933a694611c2a75fb66eaae417bf1646802e8101



链账户列表

账户名称	地址
takuyama	1a51cbae01b5259ec81af40d933a694611c2a75fb66eaae417bf1646802e8101

2.在区块链浏览器上查看交易信息：

交易详情

链ID:

a00e36c5 [开放联盟链](#)

交易 Hash:

7db1be45f561614e3456791098c16d5890dc3541f20f3262bd7d9a1bdbe837f7 [🔗](#)

交易类型:

TX_CALL_CONTRACT

时间戳:

2025-06-19 11:13:57

区块高度:

162318586

调用函数:

checkAddress

交易内容:

[hex查看](#)

Nonce:

404004717

发起地址:

1a51cbae01b5259ec81af40d933a694611c2a75fb66eae417bf1646802e8101 [🔗](#)

[交易激增 +100.00%](#) [...](#)

目标地址:

3647c2e1006011a1c7a85790a3cb8ea9c88b5e5ff579cdb58b1f0cf4a96de7e3 [🔗](#)

[活跃合约](#) [...](#)

交易金额:

0

gasUsed / gasLimit ☺:

21042/200000 (10.52%)

签名:

c19a201851be66bdd05e1200f1296e04becd76f6f8cf047f10c93186e4ce301c31580d8e8ace7141bb9a5a8b90758b4ff461f6279d96a9e3aa5393a499a2283700

结果:

● 交易成功

6、数据位置

代码如下：

```
3 contract DataLocation {
4     uint[] public storageArray; // 默认storage
5
6     function memoryDemo(uint[] memory input) public pure returns (uint) {
7         uint[] memory localArray = new uint[](3); // memory数组
8         return input[0] + localArray[0];
9     }
10 }
```

Storage 与 Memory 的区别为，storage 永久存储在区块链状态数据库中，修改会消耗大量 gas；而 memory 存储在临时内存中，函数结束后数据消失，gas 开销低

7、数组

代码如下：

```
2  contract ArrayOperations {
3      // 定长数组
4      uint[3] public fixedArray = [1, 2, 3];
5
6      function takeArray (uint index) public returns (uint) {
7          return fixedArray[index];
8      }
9
10     // 动态字符串数组
11     string[] public dynamicArray;
12
13     constructor () public {
14         dynamicArray.push("Hello");
15         dynamicArray.push("World");
16     }
17
18     // 字节数组与字符串转换
19     function bytesToString(bytes memory data) public pure returns (string memory) {
20         return string(data);
21     }
22 }
```

运行截图如下：

1. 创建定长数组并获得元素：



2. 创建动态字符串数组，并获得元素“World”

function dynamicArray

调用合约

tx hash

0x5aabf57f09a005ef5b1df139ba0870b5b...

input

Key / Index(uint256): 1

output

string: World

log

3.变长字节数组与字符串的转换:

function bytesToString

调用合约

tx hash

0x1147c56d54d3ded69d4c5dcc02c431e...

input

data(bytes): "1"

output

string: 1

log

8、映射

代码如下：

```
2  contract MappingDemo {
3      mapping(identity => uint) public balances;
4
5      function updateBalance(uint newBalance) public returns (identity, uint) {
6          balances[msg.sender] = newBalance;
7          return (msg.sender, balances[msg.sender]);
8      }
9
10 }
```

运行截图如下：

可以看到当前地址下成功添加了 100 的整数



9、结构体

代码如下：

```
2  contract ParkingSystem {
3      struct Car {
4          string licensePlate;
5          uint entryTime;
6      }
7
8      Car[] public parkedCars;
9
10     // 结构体映射
11     mapping(identity => Car) public carOwners;
12
13     // 停车操作
14     function parkCar(string memory plate) public {
15         parkedCars.push(Car(plate, block.timestamp));
16         carOwners[msg.sender] = Car(plate, block.timestamp);
17     }
18
19     // 查询车辆
20     function getCarByOwner(identity owner) public view returns (string memory) {
21         return carOwners[owner].licensePlate;
22     }
23 }
```

运行截图如下：

先调用 parkCar 函数输入号牌“沪 AH1234”，再调用 parkedCars 查询到对应信息

The screenshot displays two transaction details from a blockchain explorer. The top transaction is for the `function parkCar`, showing a transaction hash, an input parameter `plate(string): "沪AH1234"`, and an empty output field. The bottom transaction is for the `function parkedCars`, showing a transaction hash, an input parameter `Key / Index(uint256): 0`, and an output containing `string: 沪AH1234` and `uint256: 1750300962`. Both transactions have a "调用合约" (Call Contract) button next to their function names.

Function	tx hash	input	output	log
function parkCar	0x0e040d54513a135b865583073c827b...	plate(string): "沪AH1234"		
function parkedCars	0x3ff3d95c44b64453e4baedb6e59e50cf...	Key / Index(uint256): 0	string: 沪AH1234 uint256: 1750300962	

10、时间单位

代码如下：

```
contract TimeOperations {  
    uint public timestamp = block.timestamp; // 当前时间戳  
  
    // 时间运算  
    function addOneHour() public view returns (uint) {  
        return block.timestamp + 1 hours;  
    }  
}
```

运行截图如下：

1. 获得当前时间戳：



2. 时间运算



11、Solidity 函数

代码如下：

```
2 contract FunctionTypes {
3     // 11.1 返回值
4     function multiply(uint a, uint b) public pure returns (uint) {
5         return a * b;
6     }
7
8     // 11.2 命名调用
9     function namedCall(uint x, uint y) public pure returns (uint) {
10        return multiply({a: x, b: y});
11    }
12
13    // 11.3 可见性测试
14    function internalFunc() internal pure returns (string memory) {
15        return "Internal";
16    }
17
18    function callInternal() public pure returns (string memory) {
19        return internalFunc(); // 内部可调用
20    }
21 }
```

运行截图如下：

1. 分别用函数与命名函数计算 2*3:

The screenshot displays two function call details in a Solidity IDE. The first function, `multiply`, is called with a transaction hash `0x376525ebe64478c4c3053c64c8f8d96...`. Its input parameters are `a(uint256): 2` and `b(uint256): 3`, and it returns an output of `uint256: 6`. The second function, `namedCall`, is called with a transaction hash `0xfa6edd8e4d1c1d583bf60f95b32f7930...`. Its input parameters are `x(uint256): 2` and `y(uint256): 3`, and it also returns an output of `uint256: 6`. Both functions are shown with their respective '调用合约' (Call Contract) buttons.

2.测试 internal 函数:

function

callInternal

调用合约

tx hash

0x1e5d040e72f384378fc32b21e996e62...

input

output

string: Internal

log

12、Solidity 条件语句

代码如下：

```
2
3  contract GradingSystem {
4      function grade(uint score) public pure returns (string memory) {
5          if (score >= 90) {return "优秀";}
6          else if (score >= 60) return "良好";
7          else if (score >= 30) return "一般";
8          else return "差";
9      }
10 }
```

运行截图如下：

4

合约ID:0x4b227777d4dd1fc61c6f884f48641d...

TX Hash:0x9f88d0ae5ce55831d7694e2d33d47...

 function grade

调用合约

tx hash

0xd5fa3c97f0c0a874cc66a78a46905a57...

input

score(uint256): 20

output

string: 差

log

13、Solidity 循环语句

代码如下：

```
1
2  contract LoopOperations {
3      // 计算100的阶乘
4      function factorial1() public pure returns (uint) {
5          uint result = 1;
6          for (uint i = 1; i <= 100; i++) {
7              result *= i;
8          }
9          return result;
10     }
11
12     function factorial2() public pure returns (uint) {
13         uint result = 1;
14         uint x = 1;
15         while (x <= 100){
16             result = result * x;
17             x += 1;
18         }
19         return result;
20     }
21
22     function factorial3() public pure returns (uint) {
23         uint result = 1;
24         uint x = 1;
25         do{
26             result = result * x;
27             x += 1;
28         }while(x <= 100);
29         return result;
30     }
31
32     // break/continue
33     function sumEven() public pure returns (uint) {
34         uint sum;
35         for (uint i=0; i<10; i++) {
36             if (i % 2 != 0) continue; // 跳过奇数
37             if (i > 8) break;         // 提前终止
38             sum += i;
39         }
40         return sum; // 0+2+4+6+8=20
41     }
42 }
```

运行截图如下：

1.三种循环计算 100 的阶乘：

function factorial1

调用合约

tx hash

0x6cf27422fab0a23bfd7d0c4a21a61d5e...

input

output

uint256: 3.136339052934619e+76

log

function factorial2

调用合约

tx hash

0xb73bad3dac3e6b410ff04622648890e...

input

output

uint256: 3.136339052934619e+76

log

function factorial3

调用合约

tx hash

0x47f2ddd02a5b030e35bc2641325cc78...

input

output

uint256: 3.136339052934619e+76

log

2.break 与 continue 语句:



14、蚂蚁链常用平台接口函数

代码如下:


```
2 contract PlatformAPIs {  
3     // 获取当前区块Gas上限  
4     function getGasLimit() public view returns (uint) {  
5         return block.gaslimit;  
6     }  
7  
8     // Keccak256加密  
9     function hashData(string memory input) public pure returns (bytes32) {  
10        return keccak256(abi.encodePacked(input));  
11    }  
12 }
```

运行截图如下:

1.获取 gas 上限并输出



2.调用加密函数：

 function hashData

调用合约

tx hash

0xbb967e4de0b475d100789717be2e9c...

input

input(string): "test"

output

bytes32: 0x9c22ff5f21f0b81b113e63f7d...

log

15、CloudIDE 使用调试技巧

1.本地模拟器部署并查看输出结果：

合约模拟执行与单步调试

编译合约之后，可以部署合约到本地模拟器中执行，部署成功后，即可对合约方法调用测试。对于执行过的交易可以进一步使用单步调试功能，进行深入的命令级别调试，查看合约状态存储、变量、内存、堆栈等详细信息。

▼ 1.sol

SimpleStorage

字节码

部署合约

```
0x608060405234801561001057600080fd5b506
1044a806100206000396000f300608060405260
0436106049576000357c01000000000000000000
```

合约接口说明 (ABI)

```
[ { "constant": true, "inputs": [], "name":
"getValue", "outputs": [ { "name": "", "type":
"uint256" } ], "payable": false, "stateMutability":
```

1

合约ID:0x6b86b273ff34fce19d6b804eff5a3f57...

TX Hash:0x208733da623f804415c5b75275819...

function getValue

调用合约

tx hash

0xeef1d720e42c605a6d94daad3eb151a...

input

output

uint256: 1

log

2.单步调试:

22) {
23sum = a + b; // 和 (溢出时会自动回滚)
24diff = a > b ? a - b : 0; // 差 (处理负数情况)
25prod = a * b; // 积
26quot = b > 0 ? a / b : 0; // 商 (除0保护)
27}
28

编译

编译详情

合约分析

调试详情

交易详情

txhash: 0x2ee254c3854b064a33bd6203190bb7c5479b7005f54078b6bb63fc4164ff3475

<⏮⏵>

0354 PUSH1 00
0356 PUSH2 016c
0359 JUMP
0360 JUMPDEST
0361 DUP5
0362 DUP7
0363 DUP

VM Info:

vm trace step:75 execution step:75 gas:3 remaining gas:299978090

Solidity Locals:

sum uint256:"3"
a uint256:"1"
b uint256:"2"
diff uint256:"0"

3.合约检测任务与查看

合约名称	安全问题	合约接口	遵循标准
SimpleStorage	3	2	
TimeOperations	2	1	
FunctionTypes	4	3	
GradingSystem	1	1	
LoopOperations	10	4	
PlatformAPIs	3	2	
IntegerOperations	9	3	
BoolEnum	5	2	
BytesOperations	5	4	
AddressDemo	2	3	
DataLocation	3	1	
ArrayOperations	2	3	
MappingDemo	1	1	
ParkingSystem	2	2	

合约名称	漏洞名称	严重等级	工具类型	触发位置	跟踪信息	漏洞详情	修复建议
全局	编码规范 - 未标注solc编译器版本号	警告	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
SimpleStorage	安全漏洞 - 未授权写入	中危	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
SimpleStorage	编码规范 - 未被调用的函数应被声明为external	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
SimpleStorage	编码规范 - 未被调用的函数应被声明为external	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
全局	编码规范 - 未标注solc编译器版本号	警告	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 未被调用的函数应被声明为external	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 未被调用的函数应被声明为external	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 未被调用的函数应被声明为external	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 推荐使用SafeMath库	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 推荐使用SafeMath库	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	编码规范 - 推荐使用SafeMath库	提示	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	安全漏洞 - 整数溢出	高危	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	安全漏洞 - 整数溢出	高危	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看
IntegerOperations	安全漏洞 - 整数溢出	高危	蚂蚁Solidity合约静态分析工具	查看	查看	查看	查看

下面是各部分的代码：

1.

```
contract SimpleStorage {  
    uint256 private _value; // 状态变量  
  
    // 设置值（写操作）  
    function setValue(uint256 newValue) public {  
        _value = newValue;  
    }  
  
    // 获取值（读操作）  
    function getValue() public view returns (uint256) {  
        return _value;  
    }  
}
```

2.

```
contract IntegerOperations {
    // 2.1 四则运算 (含异常处理)
    function calculate(uint a, uint b) public pure returns (
        uint sum,
        uint diff,
        uint prod,
        uint quot
    ){
        sum = a + b;           // 和 (溢出时会自动回滚)
        diff = a > b ? a - b : 0; // 差 (处理负数情况)
        prod = a * b;          // 积
        quot = b > 0 ? a / b : 0; // 商 (除 0 保护)
    }

    // 2.2 取余
    function mod(uint a, uint b) public pure returns (uint) {
        require(b != 0, "Divisor cannot be zero");
        return a % b;
    }

    // 2.3 移位运算
    function shiftOps(uint x) public pure returns (uint left, uint right) {
        left = x << 2; // 左移 2 位 (*4)
        right = x >> 1; // 右移 1 位 (/2)
    }
}
```

3.

```
contract BoolEnum {
    // 3.1 比较运算
    function compare(int a, int b) public pure returns (
        bool gt, bool lt, bool eq
    ){
        gt = a > b;
        lt = a < b;
        eq = a == b;
    }

    // 3.2 枚举类型
    enum Status { Pending, Approved, Rejected }
    Status public status = Status.Pending;

    function setStatus(uint _status) public {
        status = Status(_status); // uint 转枚举
    }
}
```

4.

```
contract BytesOperations {
    // 不同长度的定长字节数组
    bytes3 public fixedBytes3;    // 3 字节长度
    bytes5 public fixedBytes5;    // 5 字节长度
    bytes10 public fixedBytes10; // 10 字节长度

    // 设置不同长度的字节数组值
    function setBytesValues() public {
        fixedBytes3 = "ABC";      // 3 字节
        fixedBytes5 = "Hello";    // 5 字节
        fixedBytes10 = "Solidity!"; // 9 字节（自动填充到 10 字节）
    }

    // 获取字节数组值
    function getBytes() public view returns (bytes3, bytes5, bytes10) {
        return (fixedBytes3, fixedBytes5, fixedBytes10);
    }

    // 获取字节数组长度
    function getBytesLengths() public pure returns (uint, uint, uint) {
        bytes3 b3 = "123";
        bytes5 b5 = "abcde";
        bytes10 b10 = "0123456789";

        return (b3.length, b5.length, b10.length);
    }

    // 直接转换（编译器自动处理）
    function directConversion() public pure returns (bytes3) {
        return bytes3("ETH"); // 转换为 3 字节数组
    }
}
```

5.

```
contract AddressDemo {  
    // 使用标准地址类型  
    identity myAddress;  
  
    constructor() public { // 添加 public 可见性  
        myAddress = msg.sender;  
    }  
  
    function checkAddress() public view returns (identity) {  
        return myAddress;  
    }  
  
    function isMyAddress(identity addr) public view returns (bool) {  
        return addr == myAddress;  
    }  
}
```

6.

```
contract DataLocation {  
    uint[] public storageArray; // 默认 storage  
  
    function memoryDemo(uint[] memory input) public pure returns (uint) {  
        uint[] memory localArray = new uint[](3); // memory 数组  
        return input[0] + localArray[0];  
    }  
}
```


7.

```
contract ArrayOperations {  
    // 定长数组  
    uint[3] public fixedArray = [1, 2, 3];  
  
    function takeArray() public view returns (uint) {  
        return fixedArray[0];  
    }  
  
    // 动态字符串数组  
    string[] public dynamicArray;  
  
    constructor () public {  
        dynamicArray.push("Hello");  
        dynamicArray.push("World");  
    }  
  
    // 字节数组与字符串转换  
    function bytesToString(bytes memory data) public pure returns (string memory) {  
        return string(data);  
    }  
}
```

8.

```
contract MappingDemo {  
    mapping(identity => uint) public balances;  
  
    function updateBalance(uint newBalance) public {  
        balances[msg.sender] = newBalance;  
    }  
}
```

9.

// 9.4 停车场管理系统

```
contract ParkingSystem {  
    struct Car {  
        string licensePlate;  
        uint entryTime;  
    }
```

// 9.2 结构体数组

```
Car[] public parkedCars;
```

// 9.3 结构体映射

```
mapping(identity => Car) public carOwners;
```

// 停车操作

```
function parkCar(string memory plate) public {  
    parkedCars.push(Car(plate, block.timestamp));  
    carOwners[msg.sender] = Car(plate, block.timestamp);  
}
```

// 查询车辆

```
function getCarByOwner(identity owner) public view returns (string memory) {  
    return carOwners[owner].licensePlate;  
}  
}
```

10.

```
contract TimeOperations {  
    uint public timestamp = block.timestamp; // 10.1 当前时间戳  
  
    // 10.2 时间运算  
    function addOneHour() public view returns (uint) {  
        return timestamp + 1 hours;  
    }  
}
```

11.

```
contract FunctionTypes {
    // 11.1 返回值
    function multiply(uint a, uint b) public pure returns (uint) {
        return a * b;
    }

    // 11.2 命名调用
    function namedCall(uint x, uint y) public pure returns (uint) {
        return multiply({a: x, b: y});
    }

    // 11.3 可见性测试
    function internalFunc() internal pure returns (string memory) {
        return "Internal";
    }

    function callInternal() public pure returns (string memory) {
        return internalFunc(); // 内部可调用
    }
}
```

12.

```
contract GradingSystem {
  function grade(uint score) public pure returns (string memory) {
    if (score >= 90) {return "优秀";}
    else if (score >= 60) return "良好";
    else if (score >= 30) return "一般";
    else return "差";
  }
}
```

13.

```
contract LoopOperations {
    // 计算 100 的阶乘
    function factorial1() public pure returns (uint) {
        uint result = 1;
        for (uint i = 1; i <= 100; i++) {
            result *= i;
        }
        return result;
    }

    function factorial2() public pure returns (uint) {
        uint result = 1;
        uint x = 1;
        while (x <= 100){
            result = result * x;
            x += 1;
        }
        return result;
    }

    function factorial3() public pure returns (uint) {
        uint result = 1;
        uint x = 1;
        do{
            result = result * x;
            x += 1;
        }while(x <= 100);
        return result;
    }

    // break/continue
    function sumEven() public pure returns (uint) {
        uint sum;
        for (uint i=0; i<10; i++) {
            if (i % 2 != 0) continue; // 跳过奇数
            if (i > 8) break;         // 提前终止
            sum += i;
        }
        return sum; // 0+2+4+6+8=20
    }
}
```

14.

```
contract PlatformAPIs {  
    // 14.1 获取当前区块 Gas 上限  
    function getGasLimit() public view returns (uint) {  
        return block.gaslimit;  
    }  
  
    // 14.2 Keccak256 加密  
    function hashData(string memory input) public pure returns (bytes32) {  
        return keccak256(abi.encodePacked(input));  
    }  
}
```