加密僵尸：搭建僵尸工厂

合约及版本号

```
pragma solidity ^0.4.19;//版本号

contract HelloWorld {//合约的基本框架

}
```

状态变量和整数

状态变量是被永久地保存在合约中。也就是说它们被写入以太币区块链中. 想象成写入一个数据库。

注: Solidity中， `uint` 实际上是 `uint256`代名词， 一个256位的无符号整数。你也可以定义位数少的uints — `uint8`， `uint16`， `uint32`， 等..... 但一般使用简单的 `uint`

数学运算

加减乘除和Java一样，不过Solidity 还支持 ***乘方操作*** (如：x 的 y次方） // 例如： 5 ** 2 = 25

```
uint x = 5 ** 2; // equal to 5^2 = 25
```

结构体

```
struct Person {
  uint age;
  string name;
}
```

结构体允许你生成一个更复杂的数据类型，它有多个属性

数组

```
// 固定长度为2的静态数组:
uint[2] fixedArray;
// 固定长度为5的string类型的静态数组:
string[5] stringArray;
// 动态数组，长度不固定，可以动态添加元素:
uint[] dynamicArray;
```

```
Person[] people; // 这是动态数组，我们可以不断添加元素
```

记住：状态变量被永久保存在区块链中。所以在你的合约中创建动态数组来保存成结构的数据是非常有意义的。

你可以定义 `public` 数组, Solidity 会自动创建 ***getter*** 方法. 语法如下:

```
Person[] public people;
```

其它的合约可以从这个数组读取数据（但不能写入数据），所以这在合约中是一个有用的保存公共数据的模式。

定义函数

```
function eatHamburgers(string _name, uint _amount) {

}
```

习惯上函数里的变量都是以(_)开头 (但不是硬性

使用结构体和数组

现在我们学习创建新的 Person 结构，然后把它加入到名为 people 的数组中.

```
// 创建一个新的Person:
Person satoshi = Person(172, "Satoshi");

// 将新创建的satoshi添加进people数组:
people.push(satoshi);
```

你也可以两步并一步，用一行代码更简洁:

```
people.push(Person(16, "Vitalik"));
```

私有/共有函数

如何定义一个私有的函数呢?

```
uint[] numbers;

function _addToArray(uint _number) private {
  numbers.push(_number);
}
```

这意味着只有我们合约中的其它函数才能够调用这个函数，给 numbers 数组添加新成员。

可以看到，在函数名字后面使用关键字 private 即可。和函数的参数类似，私有函数的名字用(_)起始。

函数的更多属性

返回值

要想函数返回一个数值，按如下定义:

```
string greeting = "What's up dog";

function sayHello() public returns (string) {
  return greeting;
}
```

函数的修饰符

这种情况下我们可以把函数定义为 ***view***, 意味着它只能读取数据不能更改数据:

```
function sayHello() public view returns (string) {
```

Solidity 还支持 ***pure*** 函数, 表明这个函数甚至都不访问应用里的数据，例如:

```
function _multiply(uint a, uint b) private pure returns (uint) {
  return a * b;
}
```

这个函数甚至都不读取应用里的状态 — 它的返回值完全取决于它的输入参数，在这种情况下我们把函数定义为 ***pure***.

Keccak256 和 类型转换

Ethereum 内部有一个散列函数keccak256，它用了SHA3版本。一个散列函数基本上就是把一个字符串转换为一个256位的16进制数字。字符串的一会引起散列数据极大变化。

类型转换

```
uint8 a = 5;
```

```
uint b = 6;
// 将会抛出错误，因为 a * b 返回 uint，而不是 uint8:
uint8 c = a * b;
// 我们需要将 b 转换为 uint8:
uint8 c = a * uint8(b);
```

事件

事件 是合约和区块链通讯的一种机制。你的前端应用“监听”某些事件，并做出反应。

```
// 这里建立事件
event IntegersAdded(uint x, uint y, uint result);

function add(uint _x, uint _y) public {
  uint result = _x + _y;
  //触发事件，通知app
  IntegersAdded(_x, _y, result);
  return result;
}
```

本章节全部代码

```
pragma solidity ^0.4.19;

contract ZombieFactory {

    event NewZombie(uint zombieId, string name, uint dna);

    uint dnaDigits = 16;
    uint dnaModulus = 10 ** dnaDigits;

    struct Zombie {
        string name;
        uint dna;
    }

    Zombie[] public zombies;

    function _createZombie(string _name, uint _dna) private {
        uint id = zombies.push(Zombie(_name, _dna)) - 1;
        NewZombie(id, _name, _dna);
    }

    function _generateRandomDna(string _str) private view returns (uint) {
        uint rand = uint(keccak256(_str));
        return rand % dnaModulus;
    }

    function createRandomZombie(string _name) public {
        uint randDna = _generateRandomDna(_name);
        _createZombie(_name, randDna);
    }

}
```

LEVI_104