本课程分为两个部分：第一部分主要学习 Cadence 语言；在学完 Cadence 语言之后，第二部分我们来开发一个完整的去中心化应用(decentralized application)，也就是 Dapp。文字讲义中涉及到知识点和概念的使用参见视频教程，您可通过以下方式联系并加入开发者引擎社区获取：

第一课 初始 Cadence

本节课主要分为三个部分：

1、Flow 生态开发以及 Cadence 简介

2、HelloWorld 示例讲解

3、Cadence 语言学习（一）

一、Flow 生态开发以及 Cadence 简介

1.1、Flow 生态概览

Flow 的创建者 Roham Gharegozlou(Dapper Labs 的 CEO)，从四个方面对 Flow 及其发展做了相关的介绍，原视频参见 https://youtu.be/XgBLCLHdHCQ。

1、为什么开发 Flow？他们第一次开发的 CryptoKitty 项目是在以太坊上发布的，由于太过火爆了，然后就暴露了以太坊的很多的问题：比如说用户体验问题，它的可扩展性能问题以及部署这成本或交易成本等，所以为了解决这些问题，Dapper Labs 决定开发一个全新的公链，于是 flow 就应运而生了。

2、由于 Flow 的创建者自己也是链上应用的开发者，所以在设计之初，他们除了要考虑到 Flow 公链本身要有更高的扩展性，更好的性能等，他们也会考虑到怎么样给开发者提供封装性更好以及更易用的公共接口。

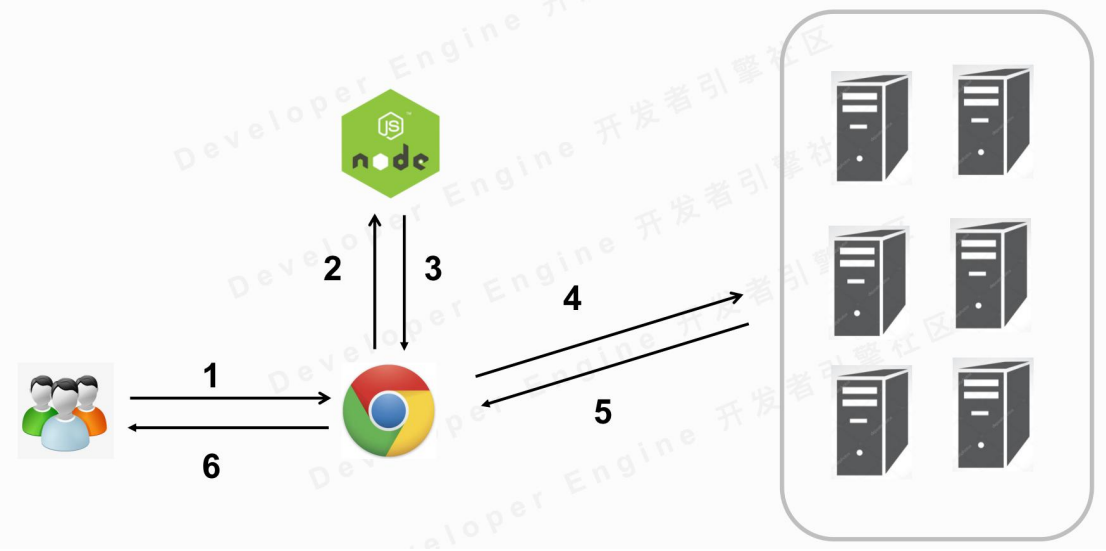
3、Flow 链的一个底层的架构，它是基于多角色的矿工与验证者分离的共识算法，不同于比特币和以太坊使用的工作量证明。一共有三个白皮书，如果想详细了解的，可以查阅官方文档

(https://zh.onflow.org/technical-paper)。

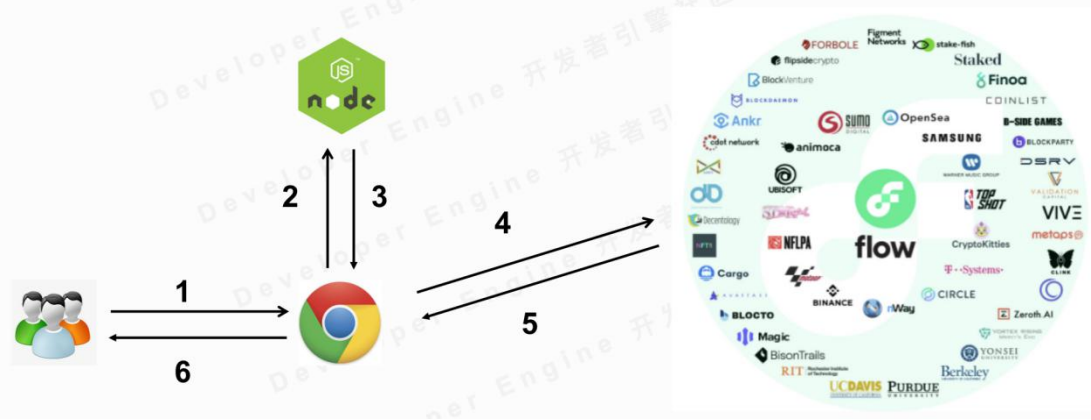
4、现在已经有很多优秀的项目在 Flow 链上了，未来还会有越来越多的开发者和团体加入，比如说正在学习的我和你们。

1.2、Flow 开发生态总览

下图展示 Web2 中比较典型的前后端分离架构。用户在浏览器上输入一个 URL，它的第一步并不是说去后端直接请求数据或者拉页面，而是会先到一个前端的 node 服务器，拉取到 html+js 页面，然后再向后端的服务器来请求数据，进而渲染整个页面呈现给用户。



那么在区块链 Web3 中，我们会有多大不同呢？实际上我们只需要把后端一换就可以了，如下图，这是第一点。第二点对于我们前端开发的伙伴来说，相关前端知识基本不变，只需要把原来封装的那些 Http 请求库或者叫 Rest API 的那些库，换成 Flow 提供的与链进行交互的库就可以了。



1.3、Cadence 语言简介

在《*深入了解* Flow Cadence，*为数字资产而专门打造的智能合约编程语言*》文章中，作者比较全面详细的介绍了 Cadence 语言，这里提取出如下三点内容阐述。

原文链接：https://mp.weixin.qq.com/s/9NrkQ_pJrCq8jCViwtL7-g

1、Cadence 是一种强类型的静态语言，也就是说它跟 golang, Java 等语言一样，你必须要先经过编译才能上线，这样就避免了很多语法层面的问题，不小心推到线上。

2、Cadence 最大的特点就是面向资源，我们从两个维度来理解：

第一个从资源的存储来这方面。在 solidity 语言开发的合约中有一个中央账本来的结构保存用户拥有的资源。比如 ERC20 合约中有一个 map 的映射结构，它的 key 是用户的地址 address，value 就是你现在有多少余额，这个 map 结构就是一种中央账本的形式。而对于 Cadence，你拥有的资产就直接存储在你账户下的私人空间内，而不是集中的中央账本。如果说合约有安全问题，比如说它有一些漏洞等等，那么在受到攻击，将会影响所有人的财产。

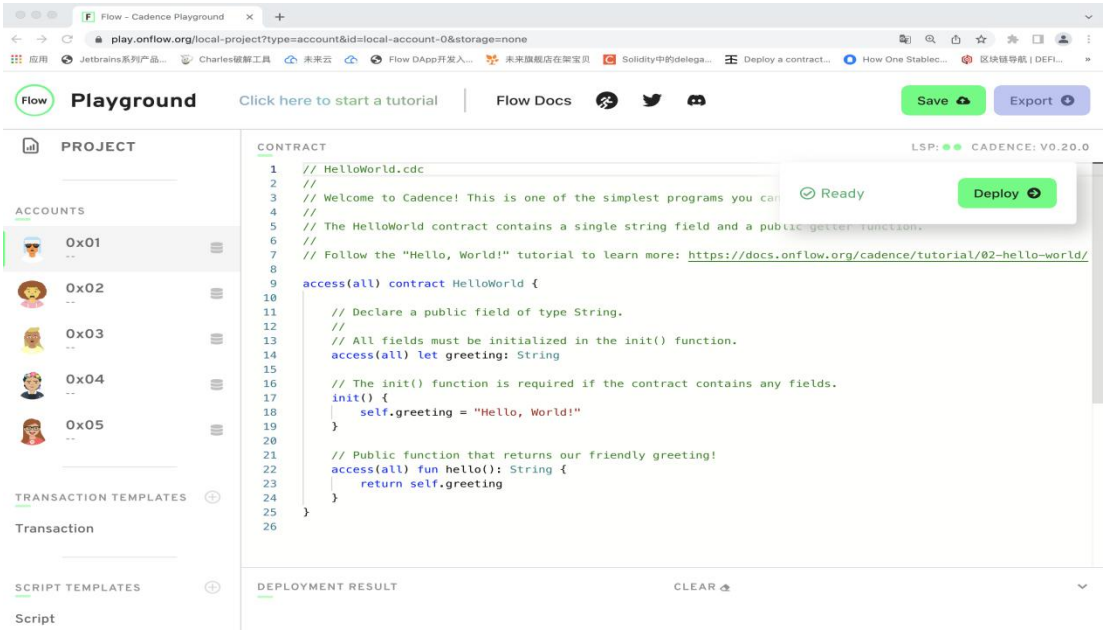
第二个方面是 Cadence 对资源（或者资产）的使用是有严格限制的，防止它由于开发者不小心或者疏忽而造成这些资产的丢失。

3、前已述及过，对开发者非常的友好。

这就是 Cadence 语言比较明显的特点，随着后续深入的学习，相信大家会有更好的理解。

二、HelloWorld 示例简介

对于初学者在学习 Cadence 时，无需急于在本地去构建一套开发环境，可以先使用官方提供的一个非常简单易用的平台叫做 playground(https://play.onflow.org/local-project)，如下图。



首先声明一点，flow 上的合约是部署到某一个账户下或者说 account 下，所以 playground 模拟了一些账户，他们地址分别是 0x1，0x2，0x3，0x4，0x5。模拟账户区域下边是 transaction，这就是跟合约进行交互的交易，再往下是 script，也是跟合约进行交互的。这两者的区别稍后再说。

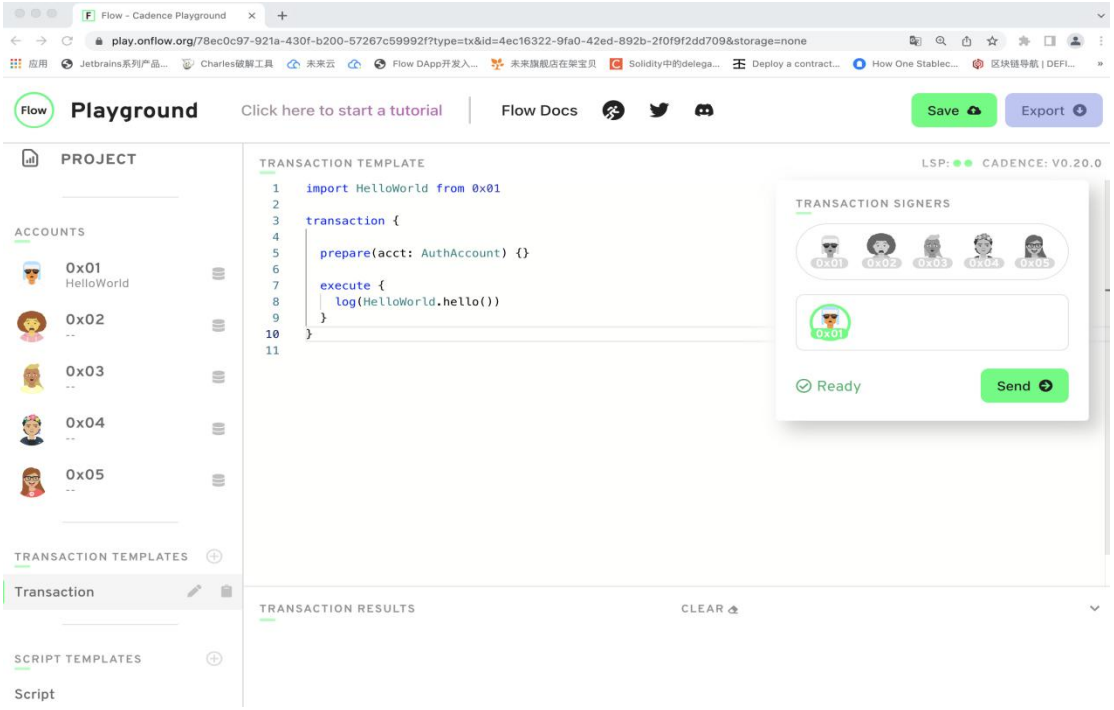
点击 0x1 账户，看到有一个 HelloWorld 最简单的合约，主要功能是定义个字符串，然后再输出。首先看第 9 行，第一个关键字 access(all)，相当于其他语言当中的 public，一种访问控制的描述，但是在 Cadence 中没有 public，它为我们简化成了 pub。接下来也是个关键字 contract 合约。然后第三个是定义的合约的名字，紧接着就是一对花括弧来表示这个合约体。关于访问控制，建议写成 access(all)的形式，至于其他的访问控制描述，比如 access(self)，access(account)等，稍后到第三部分学习 Cadence 语言的时候，我们再详细来看一下它们之间的区别。

第 14 行定义了一个成员字符串常量，使用访问控制 access(all)/public 修饰。let 相当于其他语言的 const，常量在这里面只能被赋值一次。需要注意的一点是在类型前边注意要有一个冒号。

第 17 行定义了一个 init()函数，此函数相当于其他面向对象语言中类的构造函数，同样的这个函数也可以接受参数的。

然后第 22 行定义了一个普通的成员函数 hello()，功能就是返回定义的成员常量。

这就是一个最简单的合约，接下来我们就部署一下，点击右上角的 Deploy 按钮。部署成功了之后，在 0x01 账户地址下面出现了 HelloWorld 合约的名称。接下来使用 transaction 与合约进行一下交互，如下图。



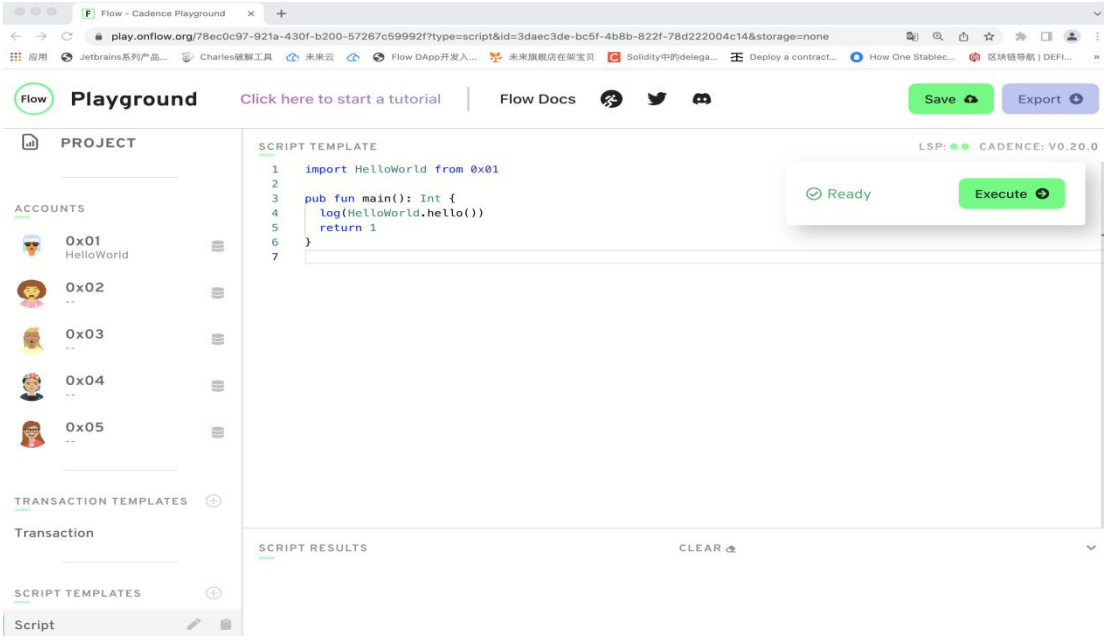
第 1 行，使用 import 关键字把要使用的合约引进来，从账户 x01 下引入 HelloWorld。

第 3 行关键字 transaction，说明接下来我要声明一段 transaction 的代码。

第 5 行的 prepare()函数也有些可以类似类的构造函数，因为它是执行 transaction 时候首先被执行的一段代码体，不同的是 prepare()的入餐比较特殊，表示谁发起交易的账户信息，或者称为给交易进行签名的账户信息。

第 7 行是一个 executive，在 execute 体当中的语句，是此交易真正要做的事情。在这里就简单的调用了一个 HelloWorld 合约中定义的 hello()方面。

这就是一个简单的 transaction 交易，在右上角的面版中，选择一个账户进行签名，点击 Send 按钮，就可以看到底部输出 hello world。接下来我们使用 Script 与合约进行交互，同样输出 hello world，如下图。



如上图，第 1 行依然引入要交互的合约。

第 3 行定义了一个 main()函数，相信大家都不陌生。脚本入口函数就是 main()函数，同样简单的调用了一下 HelloWorld 合约的 hello()方法。

这就是一个最简单的脚本 script，点击右上角的 Execute 按钮，输出 hello world 字符串。

然后我在这里直接输出了 word。

那么 transaction 和 script 有什么区别呢？实际上对于简单的功能用两个都可以。但严格来说，当访问的合约函数是只读的，即只是从合约当中读取数据，那么理论上应该用 script，所以在这个示例中应该用 script。而 transaction 用于访问那些对链上的数据状态发生变更的方法，比如说转账，从 a 账户转一个 NFT 给 B 账户等，这时候必须要用 transaction 与合约进行交互。

三、Cadence 语言学习（一）

3.1、常量、变量和注释

1、定义声明形式：

变量定义：var a: UInt = 100

常量定义：let a: String = “abc”

注意名称后类型前的冒号。

2、子作用域中可以重复定义，见如下代码示例

```
let a = 1

if true {
    let a = 2
}

// `a` is `1`
```

3、变(常)量命名规范：

1）名称由下划线，英文，数字构成，且不可以数字开头；

2）通常情况下，合约，接口,资源和结构体使用大驼峰，其余使用小驼峰

4、单行注释使用“//”，多行注释“/*xxxx*/”，文档注释“///”

3.2、值与类型

这一部分内容涉及的比较多，但都非常基础。本课程并不列举支持的类型，具体参考官方文档，这里总结一些需要注意的内容。

1、布尔（Bool）类型： true 或者 false

2、整数支持

1）十进制，二进制(0b1010)，十六进制(0xa)，八进制(0o12)

2）有符号(Int64)，无符号(UInt64)，不检查溢出的字类型(Word64)

对于 var i: UInt8, i = 250 + 20 // 溢出报错

对于 var w: Word8, w = 250 + 20 = 14 // 不会发生溢出报错

3）每种类型都有个内置的最大值和最小值标识，如最大值 UInt64.max，最小值 UInt64.min

3、小数支持

1）只支持十进制 64 位定点数（精度为 8 位小数），且不支持浮点数

2）分为有符号 Fix64，无符号 UFix64

3）最大值 UFix64.max，最小值 UFix64.min

4、整数和小数内置功能

1）fun toString(): String 将数转化成字符串输出；

fun toBigEndianBytes():[UInt8] 将数的底层表示转化成大端的字节序列。

2）Cadence 支持整饱和运算，即两数相加的结果超过该类型数值表示的最大值，则结果为最大值，如下所示，result 结果为 255

```
let a: UInt8 = 200
let b: UInt8 = 100
let result = a.saturatingAdd(b)
```

5、地址（Address）

1）其形式为 8 字节 64 位无符号整数，通常以十六进制表示

2）fun toString(): String 和 fun toBytes():[UInt8]分别表示以字符串和字节序列形式输出。

6、字符串 let str: String = “Cadence”

1) 两个属性：str.length 表示字符串长度, str.utf8 为字符序列的形式

2) 支持的函数，如下：

```
fun concat(_ other: String): String    // 两个字符串连接
fun slice(from: Int, upTo: Int): String  // 截取部分字符串
fun toLower(): String  // 将字符都转换成小写

fun decodeHex(): [UInt8]   // 将十六进制字符串解析成字节数组
fun String.encodeHex(_ data: [UInt8]): String // 与decodeHex相反
```

7、Optional 类型

如果写过 rust，可能对此类型理解起来更容易。引入此类型的意义主要有两个方面：第一个是在赋值的时候，我们可以把一个无效的空值付给 optional 类型，而之前的具体类型是无法接收 nil 类型空值的。第二个意义在于当读取一个变量的时候或者资源的时候，有可能这个资源已经被销毁了或者说转移走了，那么这个时候就要有一种形式来呈现这种无效的资源不存在的状态，在 Cadence 中使用 nil 空值，而具体的类型不接受 nil 空值，于是通过引入一种 optional 类型来满足需求。

1) 定义形式上由以上具体的类型加问号？构成，此时这个具体的类型称为此 Optional 类型的子类型，如下所示：

let a: Int? = 4; // 不是 int

let str1:String? = nil // 赋值空类型

let str2:String? = “ABCD”

2）由上可见被赋值时可接受空值 nil；读取值时可能读出无效的空值 nil。

3) Optional 类型无法像具体类型那样直接被使用，包括运算和调用成员(或内置)函数，必须先强制解构 ！(Force Unwrap)。如下所示：

let b = a! + 6; // 此时 b = 10

str2!.toLower() // 输出 abcd

4）读取判空时，除了使用常规的 if 语句，还可使用更简洁的空合运算符 ??，如：let d = a ?? 100 表示如果 a 不为 nil 则赋值给 d，如果 a 为 nil，则 d = 100。

8、数组

定义形式：let nums:[UInt8] = [1, 2, 3, 4]

Cadence 中的数组与其他语言相比，没有太大的不同。

1) 支持二维数组：let arrays = [[1, 2], [3, 4]]
2) 一个属性：nums.length 表示数组长度
3) 支持的函数如下：

```
fun concat(_ array: T): T     // 合并连接两个数组
fun slice(from: Int, upTo: Int): [T]   // 截取部分数组
fun contains(_ element: T): Bool // 判断是否包含某个元素
fun firstIndex(of: T): Int?     // 查找某元素第一次出现的位置索引

fun append(_ element: T): Void    // 向数组末尾添加元素
fun appendAll(_ array: T): Void   // 向数组末尾追加另一个数组
fun insert(at index: Int, _ element: T): Void   //向某个位置插入元素
fun remove(at index: Int): T     // 移除某个位置的元素
fun removeFirst(): T      // 移除数组头的一个元素
fun removeLast(): T      // 移除数组尾的一个元素
```

9、字典

定义形式：let dicts:{UInt8:String} = {1: "a", 2: "b"}

在其他的语言中，有的称为 Map。需要注意的一点是

1) 访问一个 key 返回一个 Optional 类型的数据
2) 三个属性：nums.length , nums.keys, nums.values
3) 支持的函数：

```
fun insert(key: K, _ value: V): V?  // 插入key-value对，返回旧value或nil
fun remove(key: K): V?     // 移除key-value对，返回value或nil
fun containsKey(key: K): Bool // 判断字典中是否包含某个key
```

3.3、访问控制

Cadence 中支持 5 种访问控制及其权限如下表所示，具体演示参见视频教程。

访问描述符	访问作用域	允许操作
priv/access(self)	当前及其内部	读
access(contract)	当前，内部及其合约内	读
access(account)	当前，内部，同账户的所有合约	读
pub/access(all)	全部	读

pub(set)	全部	读，写

表中有两个概念，当前和内部。当前指的就是某个变量的定义所处的定义域；内部是指当前作用域中的子作用域，通常由一对花括号{}括起来的代码块。其余的“合约内，同账户下所有合约，全部”就是见名知意的描述。此外还有一点需要注意：前 4 种限制的操作都是读操作，都不允许写，即使是 public 也不允许。只有用最后的 pub(set)修饰的变量才能修改。于是，只在定义时赋值一次的常量只能接受前 4 种访问控制的修饰。

3.4 流程控制

Cadence 中流程控制主要分为两大类，判断和循环。具体的形式本文就不再罗列了，参见官方文档，但有以下几点需要注意：

1、关于 if 判断语句，条件表达式无需使用括号括起来

2、关于 switch-case 多分支条件判断：

1）每个分支不会隐式陷落(fallthrough)，默认 break；

2) 建议添加 default 分支，当所有分支 case 都不满足时默认执行。

3）当有重复的分支 case 时，只有第一个匹配的分支被执行。

3、关于循环，官方文档中是支持 for index, element in array {} 这种形式的循环，但在制作本课程实践时却报错，需要注意一下。

4、两个关键字：break 和 continue 的用法与其他语言类似。

3.5 函数的基础用法

1、函数的定义，以下为一简单函数的定义及其解释：

```
fun double(num: Int): Int {
    return num * 2
}
```

定义关键字：fun;
函数名称：自定义 double
入参：num: Int
返回值类型：Int,可无返回值，注意冒号 ：
{...}： 函数体语句

2、关于入参，需要注意如下几点：

1）num 不仅仅是形参名，还是参数标签，这是一个有别于其他语言的特点。在调用此函数时需标明对应参数标签，如 double(num：10)，函数返回 20；如果想调用时省略某个参数标签，则需在定义时，在该参数标签前加'_ '，则上面定义需改为： fun double(_ num: Int): Int{...}，此时调用时可省略函数标签，使用常见熟悉的形式 double(10)，当然也支持带上参数标签；

2）入参被等同为常量，在函数体内只能读取。

3）入参支不支持可变参数，不支持默认值。此外，据官方文档说不支持 Optional 类型，但在制作本课程录制时却是支持的，可参见视频教程的演示。

4）当入参的类型是复合类型时，则允许传入任意子类型。

3、对于返回值，Cadence 中不支持多返回值，即或者没有，或者只有一个。

3.6、复合类型

1、Cadence 中复合类型主要分为两类：结构体 struct 和资源 resource。定义与简单使用如下所示：

```
pub struct Token {
    pub let id: Int
    pub var balance: Int

    init(id: Int, balance: Int) {
        self.id = id
        self.balance = balance
    }
}

let token = Token(id: 1, balance: 100)
```

```
pub resource NFT {
    pub var value: Int
    init(value: Int) {
        self.value = value
    }
    destroy() {
        //called when resource is destroyed
    }
}

let nft: @NFT <- create NFT(value: 666)
destroy nft
```

2、结构体的使用需要注意：

1）使用 struct 关键字定义, init()类似类构造函数，初始化各成员变量；

2）实例化时，不使用 new 关键字；

3) 在 Cadence 中，结构体之间相互赋值时为值拷贝。

3、资源的基本使用需要注意，下节课会进一步介绍资源的使用：

1) 声明资源类型的变(常)量时需要加@符号;

2）使用 resource 关键字，init()类似类构造函数，destroy()类似类的析构函数；

3）Cadence 对资源的使用有着严格的限制：一旦创建，只能有三种操作：存储起来，移动和销毁；

4）创建使用 create 关键字，移动使用<-符号，存储使用账户的 save()方法，销毁使用 destroy 关键字。

结构体和资源的基础使用参见视频教程的演示。

本节课结束~