Cha3-认识 Unity3d 的好基友 C# 06

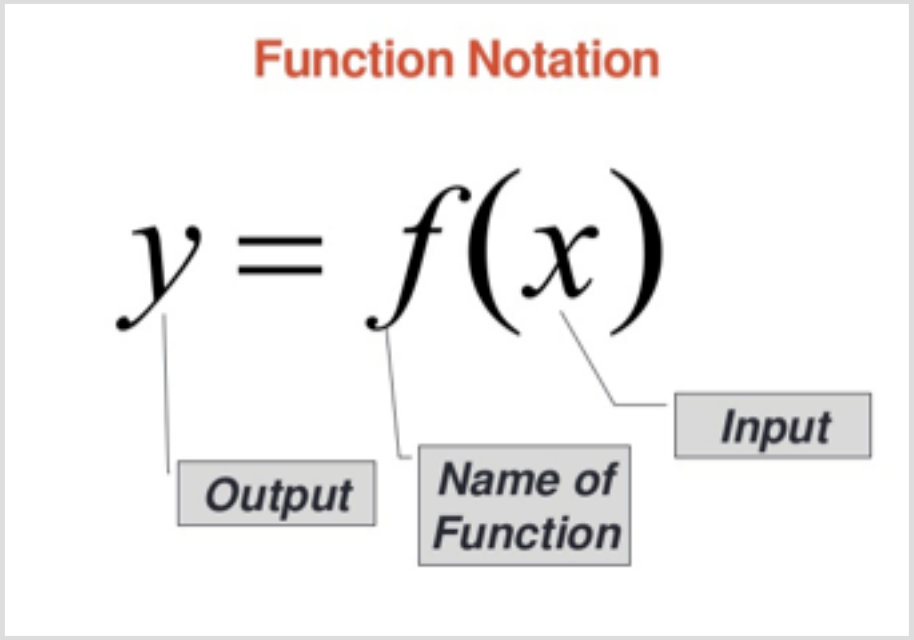
在上一章的内容中，我们一起学习了 C#中的流程控制。

而在这一课的内容中，我们将学习 C#的另外一个重要语法基础，也就是函数。

让我们开始吧~

个人微信号：iseedo

微信公众号：vrlife



04 认识 Unity3d 的好基友 C#-C#中的函数

在数学里面，函数的作用是让输入值根据特定的规则计算出某个结果并输出。

而在编程领域，函数的作用与之类似，不过不仅仅局限于计算数值，而是可以实现任何所需要的功能。简单来说，函数就是可以完成特定功能，可以重复执行的代码块。

以王者荣耀为例，在无限大乱斗模式里面，每两分钟会随机变换一次 BUFF，比如所有英雄的移速

增加 30%、所有英雄攻击增加 30%，所有英雄额外获得钟馗的飞钩技能……要让游戏里面 10 个英雄同时获取这些随机变换的 BUFF，就可以用函数来实现了，因为我们在实现这些功能的时候，就是输入了一些可以重复执行的代码块。

函数除了可以方便重复调用特定的代码段，还可以让整个代码文件看起来更加有序，也更方便 1 年后的自己或者另外一个童鞋来阅读代码~

为了说明函数的作用，这里我们把**上一课**中的所有代码重构，使用函数的方式来实现所需要的功能。

【示例】使用函数重构上一课中的所有代码

打开 Unity Hub，新建一个名为 FunctionScript 的项目。在默认场景中添加一个名为 FunctionScriptObject 的游戏对象，然后通过 Add Component 的方式在 Inspector 视图中 Create and Add 一个新的游戏脚本，命名为 FunctionScript。当然别忘了在 Project 视图中创建一个名为 _Scripts 的文件夹，然后把 FunctionScript 脚本文件拖到文件夹里。以上的具体操作这里就不再详细描述了。

如果到现在还不太熟悉怎么创建游戏脚本的童鞋请回头看看前几课的内容，直到闭着眼睛都能完成操作为止~

最后双击 FunctionScript，在 Visual Studio 中打开脚本文件，并更改其中的代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FunctionScript : MonoBehaviour
{
    //1.把定义变量和常量的语句放到最前面
    //定义一个布尔变量
    bool willTommorrowRain = false;


    //定义两个整数变量，保存支付宝账户余额以及本月订单数
    int myAlipayBalance = 100000;
    int amountOforders = 0;


    //定义一个数组，用来保存英雄的战力值
    int[] heroPowers = new int[] { 358, 762, 1903, 10, 998, 12,
13 };



    //定义一个变量，用来保存臣子的姓名
    string heroName = "Jon Snow";



    //2.Start 是 Unity 的内置函数，在游戏的开始会被调用


    // Start is called before the first frame update
    void Start()
    {
        //调用查看天气状况的函数
        CheckWeatherStatus();


        //调用查看支付宝余额的函数
        CheckAlipayBalance();


        //调用欢迎女王驾到的函数
```

```
        WelcomeMyQueen();



        //调用查看英雄战力的函数
        CheckHeroPowers();

        //调用查看对英雄评价的函数
        CheckHeroStatus();


    }

    //3.Update 也是 Unity 的内置函数，在游戏的每一帧都会被调用
    // Update is called once per frame
    void Update()
    {

    }

    //4.将之前 Start()函数的代码块组织成不同的函数，以方便调用
    //根据明天的天气来决定要做什么事情
    void CheckWeatherStatus()
    {

        if (willTommorrowRain)
        {
            Debug.Log("在家看权力的游戏第 8 季~");
        }
        else
        {
            Debug.Log("去深圳湾公园捡贝壳~");
        }
    }

    //根据支付宝的余额决定是否买买买~
    void CheckAlipayBalance()
    {
```

```
        while (myAlipayBalance > 99)
        {
            //如果余额足够，当然是买买买了。
            Debug.Log("买买买！");

            //从支付宝余额中扣款~
            myAlipayBalance -= 100;

            //本月订单数量增加 1
            amountOforders++;
        }

        Debug.Log("本月一共买买买了 " + amountOforders + "件东西。
");
        Debug.Log("余额宝中的余额还有" + myAlipayBalance + "月光不是
梦，还敢这么剁手吗？~");

        //不管支付宝里面的余额够不够，先买了再说，当然得用花呗了。
        do
        {
            //如果余额足够，当然是买买买了。
            Debug.Log("买买买！");

            //从支付宝余额中扣款~
            myAlipayBalance -= 100;

            //本月订单数量增加 1
            amountOforders++;
        } while (myAlipayBalance > 99);

        Debug.Log("本月一共买买买了 " + amountOforders + "件东西。
");
        Debug.Log("余额宝中的余额还有" + myAlipayBalance + " ...都成
首负了，还在买买买！！！");
```

```
}

//热烈欢迎女王驾到
 void WelcomeMyQueen() {
        //for 循环通常用于执行固定次数的循环指令
    for (int i = 0; i <= 10; i++)
    {
        Debug.Log("热烈欢迎北境之王和龙母莅临临冬城考察指导工作");
    }
}

//输出每个英雄的战力
void CheckHeroPowers()
{
    //遍历数组中的每个元素，输出每个英雄的战力值
    foreach (var power in heroPowers)
    {
        Debug.Log(power);
    }
}

//给出对每个英雄的评价
void CheckHeroStatus()
{
    switch (heroName)
    {

        case "Jon Snow":
            Debug.Log("He is my love，可惜know nothing");
            break;
        case "Sansa Stark":
            Debug.Log("小姑子，很厉害的北方女王，不好惹~");
            break;
        case "Jamie Lannister":
            Debug.Log("曾经的弑君者杀父仇人，眼不下这口气啊");
            break;
        case "Tyrion Lannister":
```

```
                Debug.Log("我的国王之手，但是最近智商全面下线，还敢相信他吗？");
                break;
            case "Cersei Lannister":
                Debug.Log("必须除掉的敌人，还有什么好说的");
                break;
            default:
                break;
        }
    }

}
```

天啊，这么一大堆代码，是不是看起来直接懵逼了~

不会的，这里我们按照注释行的数字编号来逐一简单解释下：

1.为了让整个代码看起来清晰易于理解，同时让变量的作用区域在整个方法体中，我们把所有定义变量和常量的语句都放到了最前面。

至于什么是方法，我们在下一课中会讲到的~

接下来的语句（一直到注释行 2 之前）都是用来定义不同变量的，大家看看对应的注释应该就可以理解了。

其中稍微奇怪一点的就是定义数组的那一行语句，不过这就是 C#语法的规定，就跟学英语学其它外语的习惯表达方式一样，不这么写就不行，所以习惯了就好~

2.Start 函数是 Unity 中的内置函数，在游戏的开始会被调用。

这里我们调用了 5 个不同的函数，而每个函数各自完成不同的任务。

大家可以对比一下上一课的 FlowControlScript.cs，就会发现使用函数来组织代码的妙处了。

3.Update 函数同样是 Unity 的内置函数，在游戏的每一帧都会被调用。

4.接下来的代码其实是上一课中相同的代码，但是我们根据其功能把代码块组织成了不同的函数，从而方便调用。

注意到，我们在谈到函数的时候，总是说调用调用，其实“调用”这个词很简单，你可以暂且理解为开始执行的意思。至于更深层次的含义，我们在之后的学习中再慢慢讲。

希望大家在学习的时候，一定要手动输入所有的代码，copy &paste 的做法是要不得的~

另外大家可能注意到了，到目前为止，我们接触到的函数都是

void FunctionName(){}的形式

这个 void 代表什么呢？代表返回值为空，也就是不返回任何数值，FunctionName 是函数名，()圆括号内是参数，而花括号内就是函数的具体内容。

实际上完整的函数定义可能是下面这样的：

modifiers return-type FunctionName(parameter list){}

看起来有点抽象，我们举个实际的例子：

public int AddThreeNumbers(int number1, int number2, int number3){}

在这个例子中，

public 就是 modifiers（修饰符），代表这个函数在方法体之外也可以访问（关于方法下一课会讲到~），int 是 return-type（返回值类型），代表这个函数会返回一个 int 类型的数值，AddThreeNumbers 是函数名 FunctionName，而()圆括号内就是 parameter list（参数列表），这里有三个 int 类型的参数。

假定某个英雄的物理攻击力= 等级对应的基础攻击力+武器攻击加成+技能加成，那么就可以列个简单的公式了。

heroAttackForce = levelBasicAttack + weaponAttack +skillAttack

好了，带着这个公式，我们创建一个新的脚本来测试下函数的完整用法~

返回 Unity 编辑器，暂时禁用 FuncitonScriptObject，新创建一个 FullFunctionObject，然后创建一个新的脚本 FullFunctionScript 并关联这个游戏对象。在 Visual Studio 中打开 FullFunctionScript，并输入以下代码：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FullFunctionScript : MonoBehaviour
{
    //1.定义一个变量，用来保存英雄的攻击力
    int myHeroAttack;

    // Start is called before the first frame update
    void Start()
    {
        //2.调用英雄攻击力计算函数
        myHeroAttack = HeroAttackForce(300, 15, 12);
```

```
        //3.输出信息
        Debug.Log("英雄的当前攻击力是: " + myHeroAttack);
    }

    // Update is called once per frame
    void Update()
    {

    }

    //4.定义一个函数，计算并返回英雄的最终攻击力
    public int HeroAttackForce(int basicAttack, int
weaponAttack, int skillAttack)
    {
        //5.定义一个临时变量
        int finalAttack;
        //6.使用公式计算英雄的攻击力
        finalAttack = basicAttack + weaponAttack + skillAttack;
        //7.返回计算结果
        return finalAttack;
    }
}
```

这里还是按照注释行的数字编号来解释一下：

1. 定义了一个变量，用来保存英雄的攻击力

2. 在 Start 函数中调用计算英雄攻击力的函数

3. 使用 Debug.Log 内置函数输出英雄攻击力的信息

4. 定义了一个函数，用来计算并返回英雄的最终攻击力

5. 定义了一个本地变量，保存计算结果

6. 使用公式计算英雄的攻击力

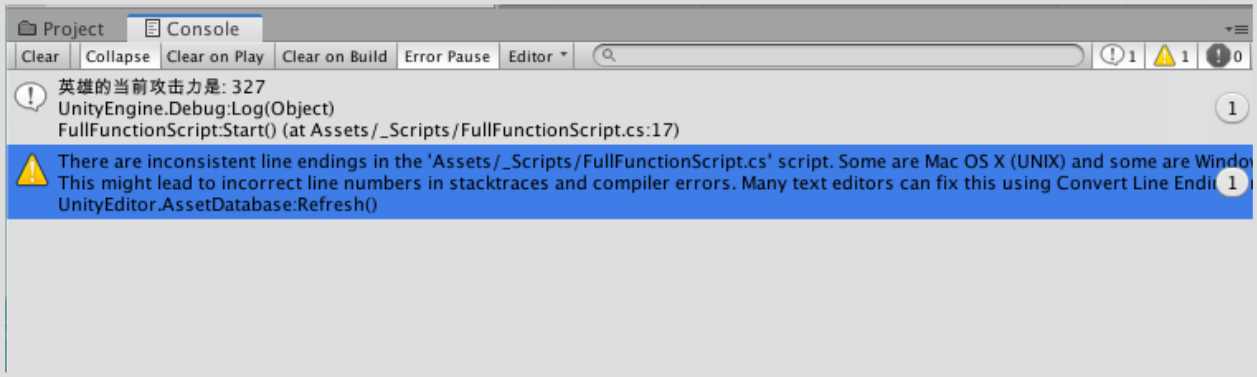
实际上，不管你信不信，这个公式就是一个最简单的算法~

7. 返回计算结果

点击 Visual Sutido 工具栏上的 Attach to Unity and Play（或者在 Unity 编辑器中点击工具栏上的 Play 按钮），切换到 Console 视图，可以查看相关的结果。

注意到除了我们期待的输出结果外，可能还会看到一行警告信息，但是 Don’t panice，这个是无关紧要的~

好了，关于 C#中的函数，就先简单介绍到这里。和其它语言类似，函数的概念并不复杂，只是我们需要注意区分内置函数和自定义函数的区别。



在下一课的内容中，我们将学习 C#语言中另外一个非常重要的概念，类和方法。

让我们下一课再见~