





硬件学Python 第五课


有了纠纷不要怕 我们一起玩猜拳

目录 Contents

第一部分  程序的选择结构与条件判断

第二部分  面向对象的基本概念

第三部分  猜拳游戏的基本原理

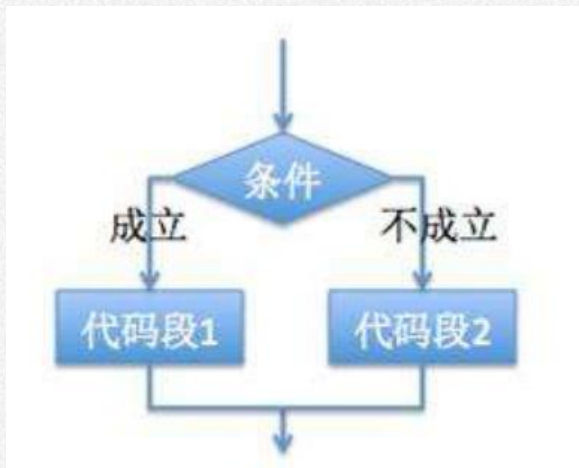
第四部分  猜拳游戏的代码讲解



第一部分 程序的选择结构和条件判断

程序的选择结构

- 选择结构程序的定义：根据条件表达式的值是**True/非零**还是**False/零**做出决策，控制代码块的执行；也就是当条件成立的时候执行某个代码段，条件不成立则执行另外的代码段



Python语言的条件判断

- 选择结构的核心就是用于条件判断的条件表达式
- Python语句提供了if语言，用于进行条件判断来控制程序的执行，基本形式为：

```
if 判断条件:  
    执行语句.....  
else:  
    执行语句.....
```

- 其中，else 为可选语句，当需要在条件不成立时执行内容则可以执行相关语句
- if 语句的判断条件可以用>（大于）、<（小于）、==（等于）、>=（大于等于）、<=（小于等于）来表示其关系。

选择结构的示例代码

```
#!/usr/bin/python

# -*- coding: UTF-8 -*-

# if 基本用法

flag = False

name = 'luren'

if name == 'python':    # 判断变量否为'python'

    flag = True        # 条件成立时设置标志为真

    print 'welcome boss' # 并输出欢迎信息

else:

    print name          # 条件不成立时输出变量名称
```

代码的输出结果为： luren

多分支选择结构

当程序存在多个分支，也就是说判断条件为多个值时，可以使用 `if ... elif ...` 或 `if ... elif ... else ...` 进行判断

```
if 判断条件1:  
    执行语句1.....  
elif 判断条件2:  
    执行语句2.....  
elif 判断条件3:  
    执行语句3.....  
else:  
    执行语句4.....
```

多分支选择结构的示例代码

```
# 多分支选择结构

num = 2

if num == 3:      # 判断num的值
    print 'boss'

elif num == 2:
    print 'user'

elif num == 1:
    print 'worker'

elif num < 0:      # 值小于零时输出
    print 'error'

else:
    print 'roadman' # 条件均不成立时输出
```

代码的输出结果为： user



第二部分 面向对象的基本概念

■ 什么是面向对象

- ✓ 面向对象 (Object Oriented, OO)，是一种软件开发方法，是一种对现实世界理解和抽象的方法。
- ✓ 面向对象是在结构化设计方法出现很多问题的情况下应运而生的，是计算机编程技术发展到现在一定阶段后的产物。
- ✓ 现阶段，面向对象的概念和应用已超越了程序设计和软件开发，扩展到如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD技术、人工智能等领域。

面向对象的主要概念

✓ 对象

- 人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件
- 对象具有自己的状态和行为

✓ 类

- 具有相同特性（数据元素）和行为（功能）的对象的抽象就是类。对象的抽象是类，类的具体化（也称实例）就是对象，类实际上就是一种特殊的数据类型
- 类具有自己的属性和操作

✓ 封装：封装防止了程序相互依赖性而带来的变动影响。类是封装良好的模块，类定义将其说明（用户可见的外部接口）与实现（用户不可见的内部实现）显式地分开，其内部实现按其具体定义的作用域提供保护

✓ 继承：子类自动共享父类数据结构和方法的机制，这是类之间的一种关系。继承性是面向对象程序设计语言的一个主要特点

■ 面向对象的简单理解

- ✓ 传统的面向过程：注重过程。当解决一个问题的时候，面向过程会把事情拆分成：一个个函数和数据（用于方法的参数）。然后按照一定的顺序，执行完这些方法（每个方法看作一个个过程），等方法执行完了，事情就搞定了
- ✓ 面向对象的解决思路：当解决一个问题的时候，面向对象会把事物抽象成对象的概念，就是说这个问题里面有哪些对象，然后给对象赋一些属性和方法，然后让每个对象去执行自己的方法，问题得到解决

问题： 冰箱里面放有脏衣服，怎么洗干净？

面向过程：

- 1 执行加洗衣粉方法
- 2 执行加水方法
- 3 执行洗衣服方法
- 4 执行清洗方法
- 5 执行烘干方法

总结：拆成一个个方法，通过一个个方法的执行解决问题

面向对象：

- 1 设计洗衣机的类和对象
 - 2 为洗衣机类加入属性和方法（加水方法、洗衣方法、清洗方法等）
 - 3 初始化洗衣机对象
 - 4 通过对象依次调用所需方法
- 总结：先抽象出类和对象，然后用对象执行方法的方式解决问题。



Python的面向对象

- ✓ Python从设计之初就是一门面向对象的语言，支持面向对象的程序开发
- ✓ 在Python中使用 `class` 语句来创建一个新类，`class` 之后为类的名称并以冒号结尾，之后是类的主体
- ✓ 类的主体由类的方法和属性组成。

```
# 定义一个类
# className为类的名字
class className:
    # 类的主体
    class_suite
```

Python类定义的代码示例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name, ", Salary: ", self.salary
```


Python类定义的代码示例

- ✓ `empCount` 变量是一个类变量，它的值将在这个类的所有实例之间共享。
你可以在类的内部访问或在外使用 `Employee.empCount` 访问
- ✓ 第一种方法 `__init__()` 方法是一种特殊的方法，被称为类的构造函数或初始化方法，当创建了这个类的实例时就会调用该方法
- ✓ `self` 代表类的实例，而不是类本身；`self` 在定义类的方法是必须有的（与普通函数的主要差别），虽然在调用时不必传入相应的参数

Python类的使用

- ✓ 实例化类其他编程语言中一般用关键字 `new`，但是在 `Python` 中并没有这个关键字，`Python`类的实例化类似函数调用方式。
- ✓ 基于使用类的名称`Employee`来实例化对象，并通过 `__init__` 方法初始化
- ✓ 对于具体的对象可以使用点号 `.` 来访问对象的属性

```
#创建 Employee 类的第一个对象
emp1 = Employee("Zara", 2000)
#创建 Employee 类的第二个对象
emp2 = Employee("Manni", 5000)
```

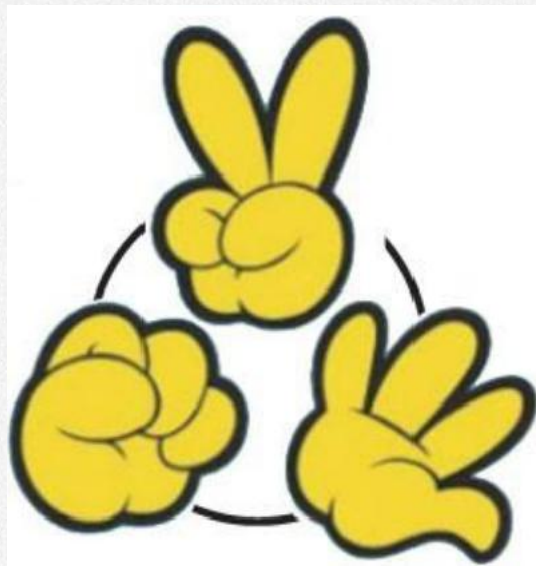
```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```



第三部分 猜拳游戏的基本原理

■ 游戏规则

- 猜拳游戏，即“石头、剪子、布”，是一种广泛流传的手技游戏，通过不同的手势分别表示石头、剪刀或布
- 游戏规则：石头胜剪刀，剪刀胜布，布胜石头



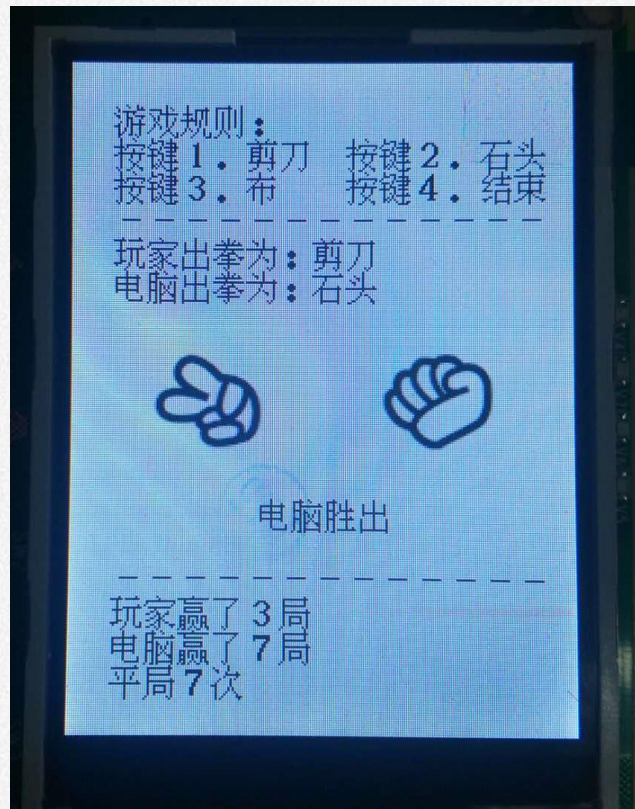
- 猜拳游戏跟“掷硬币”、“掷骰子”的原理类似，就是用产生的随机结果来作决策
- 在游戏中，用户通过按下不同的按键来表示不同的手势，分别代表石头、剪刀或布；电脑从“石头、剪刀、布”三者中随机选择一个手势，和用户的手势进行对比



第四部分 猜拳游戏的代码讲解

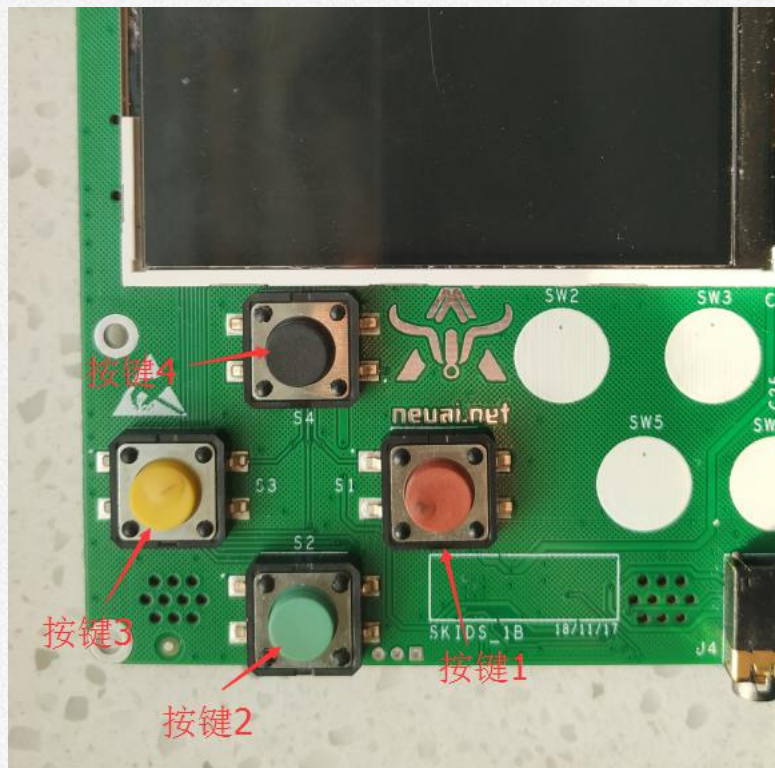
游戏界面

- 游戏界面分为三个区域
- 最顶部的区域显示游戏规则和操作说明
- 中间区域显示每次猜拳的情况，包括玩家手势、电脑手势和胜负结果
- 玩家手势通过不同的按键来表示
- 最下面的区域显示游戏胜负情况的汇总结果



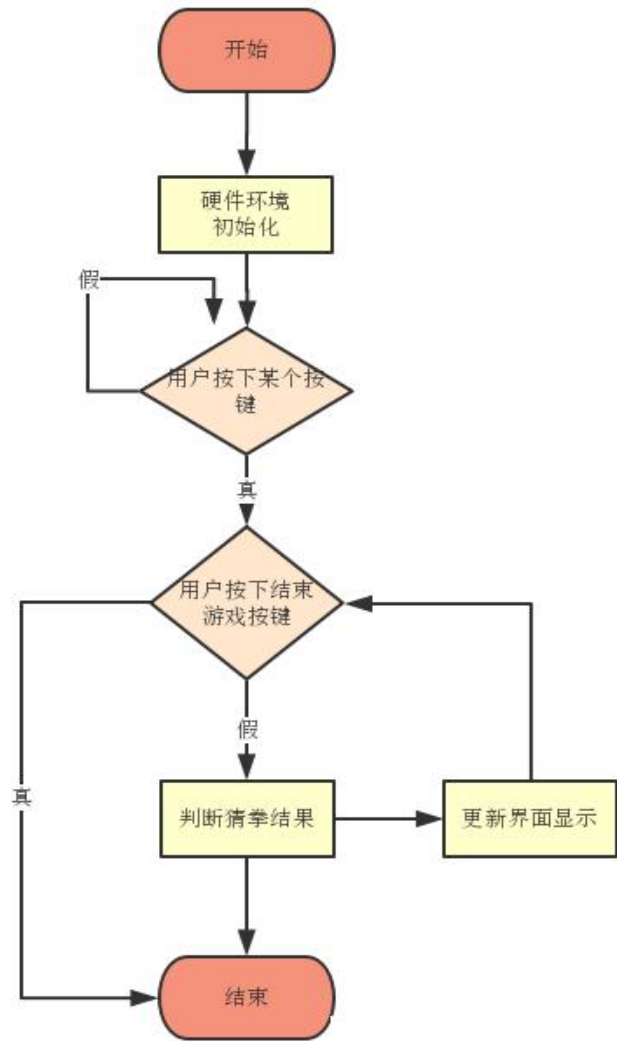
Skids按键排列

- Skids的4个按键的排列顺序如下图所示：



程序的流程图

- 程序启动后，首先进行硬件初始化，主要是对显示屏和按键进行设置
- 完成硬件初始化后，进行一个无限循环中，等待用户按键
- 当用户按下按键后，判断是否为结束按键；如果是，则结束游戏；如果不是，则获取用户输入的手势信息，同时为计算机随时生成一个手势，和用户输入进行对比，确定胜负关系
- 更新界面显示
- 等待用户的下一次按键



程序的类设计

- 创建了一个名为**Game**的类，封装了游戏的主体功能

```
class Game():
```

- 类的构造函数，负责对硬件（屏幕显示和按键设置）进行初始化，同时将游戏的一些统计数据进行清零

```
def __init__(self, playerName, computerName):  
    .....  
    #设置按键数组  
    for p in pins:  
        keys.append(Pin(p,Pin.IN))  
    #初始化屏幕  
    self.displayInit()
```

displayInit函数

- 在构造函数__init__()中，调用了displayInit()函数来进行屏幕初始化

```
def displayInit(self, x=10, y=10, w=222, h=303):  
    #显示游戏规则信息  
    mentionStr1 = "游戏规则: "  
    mentionStr2 = "按键1.剪刀 按键2.石头"  
    mentionStr3 = "按键3.布 按键4.结束"  
    text.draw(mentionStr1, 20, 20, 0x000000, 0xffffffff)  
    text.draw(mentionStr2, 20, 36, 0x000000, 0xffffffff)  
    text.draw(mentionStr3, 20, 52, 0x000000, 0xffffffff)  
    text.draw("-----", 20, 68, 0x000000, 0xffffffff)  
    self.updateTotolArea()  
    #设置游戏运行状态  
    self.gameStart = True
```

startGame函数

- 类的成员函数startGame()负责启动游戏的主流程

```
def startGame(self):  
    print("-----猜拳游戏开始-----")  
    while True:  
        i = 0  
        j = -1  
        for k in keys:  
            if(k.value() == 0):  
                if i!=j:  
                    j = i  
                    self.pressKeyboardEvent(i)  
                i = i+1;  
            if(i > 3):  
                i = 0  
        time.sleep_ms(100) #按键防抖
```


■ pressKeyboardEvent函数

- 当用户按下按键后，类的成员函数pressKeyboardEvent()负责进行具体的处理
- 该函数是整个程序中最重要函数，复杂完成具体的游戏过程处理和胜负逻辑判断
- 在函数中，首先判断游戏是否已经开始；如果游戏未开始，则不必处理键盘输入，函数直接返回

```
def pressKeyboardEvent(self, key):  
    keymatch=["Key1","Key2","Key3","Key4"]  
    #游戏还未开始，不必处理键盘输入  
    if(self.gameStart == False):  
        return
```

pressKeyboardEvent函数 -- 处理用户输入

- 对用户按下的按键进行判断，按键1代表剪刀、按键2代表石头、按键3代表布，按键4代表游戏结束；用数字1、2、3分别代表剪刀、石头和布

```
if(keymatch[key] == "Key1"):
    self.playerStatus = 1
    self.playerMessage = "%s出拳为：剪刀"%self.playerName
    bmp_jiandao.draw(40, 140)
elif(keymatch[key] == "Key2"):
    self.playerStatus = 2
    self.playerMessage = "%s出拳为：石头"%self.playerName
    bmp_shitou.draw(40, 140)
elif(keymatch[key] == "Key3"):
    self.playerStatus = 3
    self.playerMessage = "%s出拳为：布 "%self.playerName
    bmp_bu.draw(40, 140)
else:
    text.draw("游戏结束", 90, 210, 0x000000, 0xffffffff)
    #设置游戏运行状态
    self.gameStart = False
    return
```

pressKeyboardEvent函数 -- 为计算机选择随机数

- 确定用户的出拳情况后，为计算机选择一个随机数（1~3），作为计算机的出拳

```
#电脑的出拳为一个随机值
self.computerStatus = random.randint(1,3)
print(self.computerStatus)
if(self.computerStatus == 1):
    self.computerMessage = "%s出拳为：剪刀"%self.computerName
    bmp_jiandao.draw(150, 140)
if(self.computerStatus == 2):
    self.computerMessage = "%s出拳为：石头"%self.computerName
    bmp_shitou.draw(150, 140)
if(self.computerStatus == 3):
    self.computerMessage = "%s出拳为：布 "%self.computerName
    bmp_bu.draw(150, 140)
```


pressKeyboardEvent函数 -- 判断胜负情况

- 确定了用户和计算机的出拳后，对胜负结果进行判断，并记录结果

#判断胜负并显示结果

```
resultMessage = "平局！"
```

```
if(self.playerStatus == self.computerStatus): #出拳相同，为平局
```

```
    self.equalNum+=1 #平局次数加1
```

```
elif(self.playerStatus==1 and self.computerStatus==3): #用户剪刀、计算机布，用户胜
```

```
    resultMessage = "%s胜！"%self.playerName
```

```
    self.playerScore+=1 #用户获取次数加1
```

```
elif(self.playerStatus==2 and self.computerStatus==1): #用户石头、计算机剪刀，用户胜
```

```
    resultMessage = "%s胜！"%self.playerName
```

```
    self.playerScore+=1
```

```
elif(self.playerStatus==3 and self.computerStatus==2): #用户布、计算机拳头，用户胜
```

```
    resultMessage = "%s胜！"%self.playerName
```

```
    self.playerScore+=1
```

```
else: #用其它情况，计算机胜
```

```
    resultMessage = "%s胜！"%self.computerName
```

```
    self.computerScore+=1 #计算机获取次数加1
```

■ 程序的主函数

- 程序的主函数比较简单，主要进行两步操作：
 - 创建一个Game类的对象实例并初始化
 - 通过Game对象，调用Game类的startGame()函数

```
if __name__ == '__main__': #程序的主函数  
    newGame = Game("玩家", "电脑") #创建Game类的实例并初始化  
    newGame.startGame() #调用startGame()函数启动游戏流程
```

■ 实践练习

- 修改按键的处理规则，将Key4、Key3和Key2分别对应剪刀、石头和布，Key1对应结束游戏
- 调整游戏流程：当出现平局的时候，提示让用户重新按下某个按键，并为计算机重新选择一个随机数，将两者进行比较判断胜负情况；如果平局则再次提示用户重新输入，直到分出胜负



THANK YOU

牛艾科技