【白话python连载（2）】python的程序结构--知其所以然

## 从小入手-python小应用

2017年已经步入尾声，听闻小伙伴们在畅想公司年会中的抽奖小惊喜，突然想到一个有意思的小应用--用Python实现抽奖流程，同时也借助该应用了解一下python程序的基本结构。

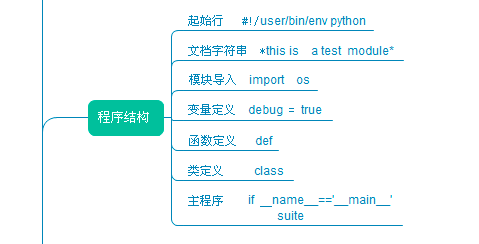
#问题描述：公司年终抽奖等级和人数的确定，并且从员工中进行随机抽选。

|  |
| --- |
| *#!/usr/bin/python #coding=utf-8* **import** random **import** sys **import** csv  print(**'年终抽奖--你会是下一个幸运者吗'**) *#定义抽奖的类* **class Lottery（object）:**  **’公司年终抽奖’** *#初始化所有抽奖人的名单地址* **def** \_\_init\_\_(self,*filepath*)**:** self.datasource **=** *filepath* **def Get\_people**(self)**:** *#获取待抽奖的用户的名单和用户个数  #获取方式，本次按照读取csv文件形式操作（也可以读取数据库等）* people\_list **=** []  **with** open(self.datasource) **as** source **:** all\_people **=** csv.reader(source)  **for** people **in** all\_people**:** people\_list.append(people)  people\_num **=** len(people\_list)  print(**'本次共有 {} 人参与抽奖'**.format(people\_num))   *#用户输入确认抽奖的等级（一等奖、二等奖、三等奖）,并且判断每个等级的奖品人数合理（不超过总人数）* level **=** int(input(**'请输入本次抽奖分几等：'**))  level\_dict **=** {}  sum\_level **=** 0  **for** i **in** range(level)**:** print(**'请输入第 {} 等级的抽奖'**.format(i**+**1)**+'数：'**)  one\_level **=** int(sys.stdin.readline())  sum\_level **=** sum\_level **+** one\_level  **if** sum\_level **<=** people\_num**:** level\_dict[i] **=** one\_level  **else:** sum\_origin **=** sum\_level **-** one\_level  cha **=** people\_num **-** sum\_origin  *# print('输入人数超额，请重新输入小于%s的数'%cha)* **while** one\_level **>** cha**:** print(**'人数超额，请重新输入不大于 %s 的数' %**cha)  one\_level **=** int(sys.stdin.readline())  sum\_level **=** sum\_origin **+** one\_level  level\_dict[i] **=** one\_level  print(**'每个等级及对应人数'**,level\_dict)   *# 抽取每个等级的获奖用户名单* **for** i **in** range(len(level\_dict))**:** level\_peoples **=** []  **for** j **in** range(int(level\_dict[i]))**:** level\_people **=** random.choice(people\_list)  level\_peoples.append(level\_people)  people\_list.remove(level\_people)  print(**'第 %s 等级下被抽中的人员为：' %** (i **+** 1 ))  print(level\_peoples)  *#类功能定义完毕，初始化并使用* **if** \_\_name\_\_ **== '\_\_main\_\_':** peoples **=** Lottery(**r'D:\vernacular\data\1.csv'**)  peoples.Get\_people()  else:  print(’未到年终，不抽奖呢’) |
| #运行结果  "D:\anaconda python\python3.6.exe" D:/vernacular/coding\_ziyi/lottery.py  年终抽奖--你会是下一个幸运者吗  本次共有 18 人参与抽奖  请输入本次抽奖分几等：3  请输入第 1 等级的抽奖数：  5  请输入第 2 等级的抽奖数：  7  请输入第 3 等级的抽奖数：  8  人数超额，请重新输入不大于 6 的数  7  人数超额，请重新输入不大于 6 的数  3  每个等级及对应人数 {0: 5, 1: 7, 2: 3}  第 1 等级下被抽中的人员为：  [['zjdsj\_014'], ['zjdsj\_010'], ['zjdsj\_007'], ['zjdsj\_018'], ['zjdsj\_013']]  第 2 等级下被抽中的人员为：  [['zjdsj\_001'], ['zjdsj\_009'], ['zjdsj\_004'], ['zjdsj\_016'], ['zjdsj\_002'], ['zjdsj\_008'], ['zjdsj\_005']]  第 3 等级下被抽中的人员为：  [['zjdsj\_011'], ['zjdsj\_006'], ['zjdsj\_015']]  Process finished with exit code 0 |

该应用实现的基本逻辑为：通过python读取员工信息表（核心：构建完整的抽奖员工名单列表）→设置本次抽奖的等级划分以及每个等级将获奖的人员数目（核心：要保证中奖的人数不能超过员工总人数）→依次抽出每个等级下的各个获奖人名单（核心：利用random函数保证抽奖的随机性）。

## python程序结构详细介绍

依次来看该应用背后python的具体程序结构。一个规范的python程序，由起始行、文档字符串、模块导入部分、变量定义、函数定义、类定义、主程序等7大部分构成。



**起始行**

为了保证良好的编程习惯，在每个python脚本文件起始行需要附上特殊标识和编码标识。

**特殊标识**

在Windows上虽然不能实现像.exe文件那样直接运行.py文件，但是，在Mac和Linux上是可以的，前提是需要在.py文件的第一行加上一个特殊的注释，然后，通过命令给\*\*\*.py以执行权限，就可以直接运行.py文件了。

|  |  |  |
| --- | --- | --- |
| *#!/usr/bin/python*  特殊注释 | ’’’  多行注释  ’’’ | ”””  多行注释  ””” |
| *$ chmod a+x \*\*\*.py 执行权限* |

其中‘#’表示对python的单行代码进行注释（快捷键为：shift+3或Ctrl+/），而“’’’\*\*\*\*’’’”表示同时对多行代码或整段代码进行注释。在python中单引号和双引号作用类似，但是必须成对出现，可以互相嵌入使用。

**编码标识**

Python3的字符串在内存中默认以Unicode进行编码，一个字符对应若干个字节。Unicode作为万国码，可以看做是一个符号集，它规定了如何编码，但没有规定如何传输、保存这个编码，而UTF-8作为Unicode的实现方式之一，最大的一个特点，根据不同的符号而变化字节长度，按照实际需求使用1~4个字节表示一个符号。因此在编写python程序时常采用utf-8的编码方式进行存储（可以确保代码的通用性）。

为方便代码的通用性，需要提前指定程序的编码方式，申明的方式主要有如下三种，这样可以保证python编译器在读取代码文件时按utf-8方式读取，但这个声明不能将py文件保存成utf-8格式的！而为了确保\*\*\*.py文件是utf-8格式，需要在高级文档编辑器中设置。如果是使用pycharm的集成环境默认的保存的py文档为utf-8的。如果有需要可以通过查找‘file - settings – editor - file encoding’进行修改。

|  |  |  |
| --- | --- | --- |
| *# -\*- coding: utf-8 -\*-* | #coding=utf-8 | # vim:set fileencoding=<utf-8>: |

如果想将Unicode的字符以utf-8形式保存，用encode(‘utf-8’)转化为utf-8编码的bytes，写入文件。对于可能采用gbk或其他编码方式进行存储的其他文件或者网页内容，需先明确源文件的编码格式，然后通过decode（‘gbk’）进行解码实现与其他数据进行交互。

|  |  |
| --- | --- |
| *‘我’.encode(’utf-8’)→b'\xe6\x88\x91'* | b'\xce\xd2'.decode(’gbk’)→ 我 |

**统一设置**

起始行的两条说明，需要在每个py文件中进行设置，因此pycharm也提供了统一设置的快捷方式，通过查找‘file - settings – editor - file and code templates – python scripte’进行设置

文档字符串

文档字符串是辅助理解python程序的重要工具，而且属于程序某个对象的一个属性（\_\_doc\_\_），在运行程序时，可以从一个函数中返回文档字符串，提高交互性。

|  |  |
| --- | --- |
| *print(Lottery.\_\_doc\_\_)* | 公司年终抽奖 |

相关定义

|  |  |  |  |
| --- | --- | --- | --- |
| 模块导入 | 变量定义 | 函数定义 | 类的定义 |
| **import** random **import** sys **import** csv | level\_dict = {}  people\_list = []  sum\_level = 0 | def \_\_init\_\_(self,filepath)：  #构造函数（函数体）  def Get\_people(self): #普通函数 | class Lottery(object): |
| 导入模块后，可以直接使用对应模块中的相关方法和功能。需提前将所需包在开发环境中安装好 | Python中的变量不需要提前使用，而且变量的类型是在定义变量时直接确定 | def：表示函数定义的关键字  \_\_init\_\_/ Get\_people ：为函数的名称，可以根据函数功能设定  （self，filepath）：表示该函数对应的参数  函数体：表明函数可实现的功能 | 类的方法下需要额外的self参数。self代表当前对象的地址，能避免非限定调用造成的全局变量。 |

主程序

Python程序在运行时其实并不需要主程序（main()），而是文件自上而下的依次执行，但是在很多python脚本中在最后的部分会执行一个判断语句if \_\_name\_\_ == "\_\_main\_\_:"，之后还可能会有一些执行语句。这样做的目的是可以拓展python文件的多用性，既可以独立运行，也可以被当做模块导入（import）到其他文件。当导入到其他文件时，\_\_name\_\_一直将为导入程序的文件名，而不是\_\_main\_\_。这样也可以进一步控制可以执行的功能。只要将它们放到if \_\_name\_\_ == "\_\_main\_\_:"判断语句之后就可以了。

## 面向对象的程序设计思想

