复盘总结

第一周：环境配置、变量运算、列表

## 1.本周学习到的东西

1. 环境配置

在windows中使用python之前，需要先配置环境。

安装python和sublime软件

打开sublime之后，输入windows build脚本

[windows build脚本]

{

"cmd": ["C:\\Python27/python","-u","$file"],

"file\_regex": "^[ ]\*File \"(...\*?)\", line ([0-9]\*)",

"selector": "source.python",

"encoding": "utf-8",

}

地址：<https://github.com/guoylyy/uband-python-s1/blob/master/week1/day1-note.md>

输入之后保存为sublime-build文件，取名为python2

接着在sublime里，选择tools-buildsystem-python2

以后每次打开一个新的sublime文件后，都记得保存为py格式，然后就可以运行了

保存为py格式之后，输入的命令行会自动显示为彩色

运行print “”，记得print后面要打空格

另外如果要输入中文的话，一开始一定要指定编码为utf8

# -\*- coding:utf-8 -\*-

然后ctrl+B就可以运行程序了

注意：运行程序之前一定要先保存为py格式，有中文内容的一定要先输入# -\*- coding:utf-8 -\*-

（2）变量

变量有三种类型：

数字变量：%d整数；%f浮点小数（默认显示小数点后6位，f前面加0.3，表示显示3位小数）；%g浮点数（不显示无意义的0，g前面加0.3，表示显示3位有效数字）

字符变量：%s字符，要加双引号或者单引号括起来

布尔变量：True或者False，注意首字母大写

（3）运算规则

加法：字符+字符，数字+数字

字符+数字会报错

乘法：字符\*数字=字符重复几遍

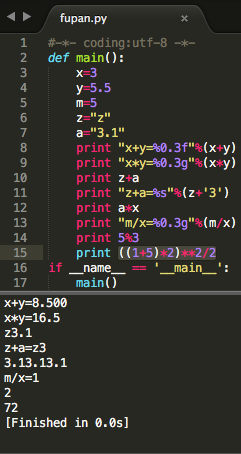
\*\*：同时打两个\*表示幂，如5\*\*2=25

除法：注意，如果是整数相除，结果也用整数表示，比如5/3=1;要想结果用小数表示，改为5.0/3

%：这不是除，而是显示余数，5%3=2，用这个运算符可以快速判断奇数还是偶数

（）：运算时，幂优先于乘除法，乘除法优先于加减法，括号里的先运算：1+5\*2\*\*2/2=11；((1+5)\*2)\*\*2/2=72

格式化print 的诸多方法 %d %f %g:注意格式化之后，括号前面不要忘了加%



## （4）单一职责原则

一个代码块只用来做一件事情

def main():

缩进之后写代码内容（注意缩进的时候用tab，千万不要tab和空格混用）

如果没有代码内容就写pass

if \_\_name\_\_==’\_\_main\_\_’

main()

后面这句话千万不要忘了

（5）利用return来把变量在不同的代码块之间传递

这部分虽然能够把作业做出来，但在第一周的时候其实还没明白为什么

return到底是什么意思？为什么传递变量的时候，变量后面要加上数字以示区别？加数字的规则是什么？为什么不同代码块之间调换位置并不影响运行？python是怎么识别出来的？

真正觉得有一点感觉是在跟着做完大作业之后，各种出现def XXX（YYY）：

之后再return，差不多比较理解用法了

简单来说def表示定义了一个函数XXX，如果XXX后面的括号里再加上（YYY），就表明这个函数中会使用YYY这个变量来进行运算，函数运算结束之后，生成的结果变量，假设用A来表示，需要return表示返回，这样一来以后在其他模块中，可以写A=XXX（YYY）来调用XXX函数，注意一点：这里写在括号里的YYY意思是定义时用的那个YYY变量实际的取值，比如：

def xxx(a1):

## b1=a1+5

## return b1

## def main():

## a=9

## c=xxx(a) #意思是把9代入到函数XXX（9）中，XXX表示让代入进来的每一个数字+5，返回结果

## print c

## if \_\_name\_\_ == '\_\_main\_\_':

最后print出来的结果就是14，但如果不写return b1，运行出来的结果是none

此外，还可以同时return多个变量，如加上以下语句：

c1=b1+6

return b1,c1 #返回一个元祖（）

return (b1,c1)#返回一个元祖（）

return [b1,c1] #返回一个列表[]

return {b1,c1} #返回一个set([])

return {b1:c1}#返回一个字典

注意return之后代码就停止运行了，上面这些return同时写，只运行第一个

## main()

## （6）主函数 main()

是函数文件运行代码的入口

## （7）列表

list：创建一个列表

命令为 lst（列表名称，可以自己命名）=[‘a’,’b’,’c’] #注意，要创建的列表项目用方括号括起来，里面的项目如果是字符变量，要加引号；如果是数字变量，不用加引号

在list中增加一个新变量：lst.append(),注意：这个命令只能增加一个新变量

在list中增加一组新变量：lst.extend([])

在list中的某个位置插入一个新变量：lst.insert(位置，变量)，注意：位置从0开始编号，第一个是0

在list中删除一个新变量：lst.remove

在list中删除某个位置的变量：del lst[]方括号中填位置

（如何一次性删除多个变量？如何一次性删除多个不同地方的变量？）

list长度:len()

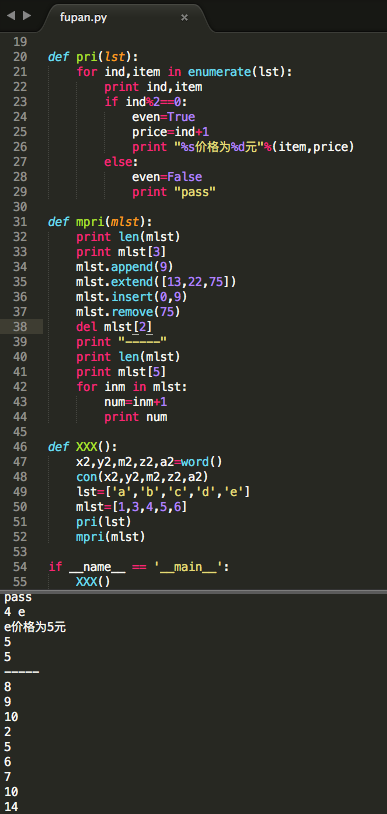
list具体某个位置的某个项 lst[],方括号里填下标

for：循环，注意加冒号

enumerate：迭代，一次性报告lst中的所有项和下标

在写大作业的时候发现自己对列表的理解还不够深入，经常会习惯性使用lst=lst.append()这种表达，结果就会报错

要记住：增加、插入、删除变量，通通是在lst后面打点，然后接命令，不要加等号



## 2.本周编程中所犯的错误以及思维过程

1）安装sublime软件时，半天找不到官网，后来意识到sublime和python不是同一家的，sublime有自己的官网，而我一直在python的官网搜

2）经常忘了在for，if，def后面加冒号

## 3.本周学到的对于之后几周学习指导意义的方法

1）多试代码，有疑惑就按自己的理解写了试着跑跑看，根据报错的结果来判断

2）不要忘了冒号！

第二周：学习了列表、字典、元祖

1. 列表：list
   1. lst=[‘a’,’b’,’c’]
   2. 列表中的不同项之间用逗号隔开，列表里的项可以是字符、数字或者布尔值，注意如果是字符，要加引号
   3. 列表可以添加、插入、修改、删除内容，分别是lst.append(待添加项)，lst.extend([待添加列表]，lst.insert(下标，待添加内容)，lst[下标]=修改为哪一项，del lst[下标])
   4. 列表可以切片，lst1=lst[起始下标：终止下标]，注意左边为开区间，右边为闭区间，即lst[0:2]取第0，1两位值
   5. 列表可以利用for循环遍历，注意enumerate函数，可以同时遍历列表中的下标和每一项内容
2. 字典：dictionary
   1. 字典用花括号{}来表示，其中包括键key和值value，dictionary={key1:value1,key2:value2,key3:value3}

dic={"书":"book","笔":"pen","纸":"paper"}

dic2={'1+2':3,

'2+3':5,

'5\*6':30,

'7-3':4,

'8/4':2

} #字典里的词条可以写在一排，也可以换行，但一定要记得用逗号隔开

dic3={'list1':[1,3,5,7,9],9:[8,7,6,5,4,3,2,1]} #字典的值也可以是一个列表

dic5={'1':{'1+2':3,'3+5':8},'2':{'1\*2':2,"3\*5":15}} #字典的值还可以是一个字典

* 1. 字典和列表一样，可以添加、删除、修改和替换

dic["爱"]='love' #在字典中加入一个新的键

dic['书']='books' #替换一个已经存在的键值

del dic['爱'] #删除一个键值

dic.clear() #清空字典里所有的值

* 1. 字典也可以遍历

for i in dic2.keys(): #遍历dic2中的每一个键

if dic2[i]%2==0:

print i

for n in dic.values(): #遍历dic中的每一个值

print n

* 1. 字典的键和值可以形成单独的列表

print dic2.keys() #获得一个列表，列举出字典里的key，注意不是按字典里写的顺序列举的

print dic.values() #获得一个列表，列举出字典里所有的值

print dic3['list1'][1] #当字典里的值是一个列表时，可以print列表中的某一项

print dic5['1']['3+5'] #当字典里的值是字典时，可以获得字典中某个key的某项取值

* 1. 字典可以查找

print len(dic) #字典的长度，即字典中有多少个key（value）

print dic.has\_key("书") #当字典里存在某个key值时，返回True

print dic2.has\_key(1) #当字典里不存在这个key值时，返回false

1. 元祖：tuple
   1. 元祖用小括号来表示tup=（1,2,3）
   2. 元祖的取值和切片

print tup[2]#记得这里要打中括号[]，和列表一样下标从0开始，不能同时取两个0，1，想要同时取多个值只能用切片

# 切片

print tup[0:1]#切片左边是闭区间，右边是开区间，即右边的实际取值会比写出来的下标少一位

print tup[2:]#右边什么都不写，默认取到最后一位

print tup[:3]#左边什么都不写，默认从第一位开始数

* 1. 查找元祖中是否存在某个值

print (1 in tup) #存在，则打印ture

print (5 in tup) #不存在，则打印false

* 1. 元祖的遍历

for item in tup:

print item

print "enumerate-------"

for index,item in enumerate(tup):

print index+item

* 1. 元祖与列表的差别：元祖不可以编辑（添加、删除、插入、修改），而列表可以
  2. 元祖可以用来给变量赋值

tup = (1,2,3,4)

（a,b,c,d）=tup

或者

（a,b,c,d）=(1,2,3,4)

print a,b,c,d

得到的结果是1，2，3，4

思考题：

[1,2,”3”,”aaa”]

{“aa”:”bb”, “cc”:”dd”}

(1,2,3)

以上这三个数据结构有何异同？什么场景时候使用，说出自己的理解

以上三个数据结构分别是列表、字典和元祖

列表有下标来说明顺序，可以编辑、修改、删除或添加元素，可以利用下标来取值、可以切片、可以遍历

字典没有下标，键值的排列没有顺序，可以编辑、修改、删除或添加元素，可以利用键值来取值、可以遍历，还可以查找某个键值是否存在于字典中

对比列表和字典，可以认为字典是给列表中的每一个值取了唯一的“名字”(key)

元祖有下标来说明顺序，可以取值、可以切片、可以遍历，但不能编辑

元祖可以用来给多个变量一次性赋值

对比列表和元祖，可以认为元祖是固定顺序、不能编辑的列表

try/except

这个编程语法存在的价值？说出你的理解

在编程过程中，有时候程序运行会出错，一般情况下，出错之后，程序就自动终止了，然而有时候，我们希望程序出错后还能够继续运行下去，只是在出错的地方做一下标记

这个时候，就可以使用try/except命令，在可能出错的命令行前面输入try，在except之后输入如果出错后希望执行的操作，就能在出错时不影响整个程序的运行

“Don’t repeat yourself” 原则是什么

一些代码如果多次重复出现（三次及以上），就要看看这种重复是否能够简化

简化的方法，一是利用变量，二是利用函数

变量的作用是对于一些重复的元素，把它们变成变量，以后再输入时就能只输入变量，而非反复输入该元素，并能方便修改

函数的作用是对于一些重复的操作，把它们写成一个单独的函数，以后想要进行这个操作时，直接调用函数就可以了

## 2.本周编程中所犯的错误以及思维过程

1. 我在写day12的作业，即修改芦苇的代码时，发现无法运行粘贴字典的那行命令，检查后发现，根据代码的逻辑，“撕毁页面”和“粘贴页面”之间，是一个嵌套语句，查询的单词应该和前面一开始查询的单词不同，于是重新修改了代码，使查询的单词下标+1，然而这里又报错了，原因在于列表里遍历到最后一项时，再+1就超出了列表的范围
2. 一开始采取的修正方法是加上一个条件，index<len(lst)
3. 后来学习了try之后，使用try命令来报错，也能使程序继续运行下去

第三周

1. 本周学习到的东西

（1）命令行：在shell或者iTerm里打开命令行

六个基本命令：who am i（我是谁）cd（改路径）pwd（当前路径）ls（列出所在文件夹下的所有文件）man（了解某个命令）python（直接通过命令行运行python程序）

（2）库：import 可以用来引入库，可以通过devdocs.io网站查询各种不同库的用法

（3）类：class，通过类来定义一系列参数，即把各种def打包到一个class里，类反映了一种“面向对象”的思想，即写程序的时候，考虑的不是线性的过程，而是通过定义某个对象的一系列参数，方便后来调用，等于是把某个具体对象的行为抽象为一类行为，以后但凡涉及到这类对象中的某一个具体要求，都可以通过对类的调用来完成

类的写法：

class Bike():

"""docstring for bike"""

def \_\_init\_\_(self, ride\_class,ride\_time):#注意这里，定义的时候初始的固定用法是def \_\_init\_\_,后面的括号里第一个self是必须的，其他根据所需要的数据决定，第一步是把和这个类相关的数据传递进来初始化

什么是初始数据呢，就是在后面调用的时候，如果直接调用bike1=Bike(“ride\_class”,”ride\_time”)，调用的就是初始化时设定的数据，也即将类Bike赋值给了变量bike1

self.ride\_class=ride\_class

self.ride\_time = ride\_time

def be\_ride(self,weekday,distance):

这里定义类的具体行为，比如be\_ride

后面调用的时候，要在变量bike1后面打一个点，再直接写行为就好，类似列表里的lst.append()，这里写bike1.be\_ride(weekday,distance)，表示把括号里的值赋给前面定义的行动方式

speed=float(distance)/self.ride\_time

print "%s的时候骑%s,平均时速为%0.2fkm/h"%(weekday,self.ride\_class,speed)

1. 思考题

1.1 你认为commnad or shell这种命令行软件有什么用处？（可搜索）

是一个命令解释器，找到在操作系统的哪个位置有一个什么样的文件，并说明要对这个文件进行什么样的操作

1.2 你尝试了哪几个python 的library？有感受关于库的使用的好处？有哪些？（自己描述）

turtle，有非常详细的解释，说明到底要怎么使用这个命令，有实例来帮助理解，但有时候也会看不太明白

1.3 面向对象的原理是什么？对比了面向对象和面向过程，你觉得面向对象（或者说类）带来了哪些好处？ （自己描述）

建立一个“类”，以后的很多操作就可以直接调用这个“类”里面的各种命令，和面向过程的方法相比，面向对象能够减少较多重复劳动

2.本周编程中所犯的错误以及思维过程

1）经常忘记打冒号，系统报错后才反应过来

2）还是没有想明白要怎么同时循环调用几个不同列表中的内容

---

class Bike():

"""docstring for bike"""

def \_\_init\_\_(self, ride\_class,ride\_time):

self.ride\_class=ride\_class

self.ride\_time = ride\_time

def be\_ride(self,weekday,distance):

speed=float(distance)/self.ride\_time

print "%s的时候骑%s,平均时速为%0.2f km/h"%(weekday,self.ride\_class,speed)

def main():

lst1=['周一','周二',"周三"]

lst2=['电动车','自行车','电动车']

lst3=[0.5,2,0.6]

distance=20.0

for i in lst1:

bike1=Bike("自行车",0.5) 这里如果希望能够从lst2和lst3里循环调用其中的项，依次打印，应该怎么来写？

bike1.be\_ride(i,distance)

if \_\_name\_\_ == '\_\_main\_\_':

main()

上网搜索“同时遍历多个列表”后找到了答案，<http://www.cnblogs.com/hwlcathy/p/4367390.html>

使用zip命令就可以把几个list来打包遍历了

class Bike():

"""docstring for bike"""

def \_\_init\_\_(self, ride\_class,ride\_time):

self.ride\_class=ride\_class

self.ride\_time = ride\_time

def be\_ride(self,weekday,distance):

speed=float(distance)/self.ride\_time

print "%s的时候骑%s,平均时速为%0.2f km/h"%(weekday,self.ride\_class,speed)

def main():

lst1=['周一',"周二","周三"]

lst2=['电动车','自行车','电动车']

lst3=[0.5,2,0.6]

distance=20

for (i,l,m) in zip(lst1,lst2,lst3): #同时遍历多个列表，可以使用zip命令

bike1=Bike(l,m)

bike1.be\_ride(i,distance)

if \_\_name\_\_ == '\_\_main\_\_':

main()

启发：遇到不懂的地方善于放狗搜一搜，有很多伟大的、有共享精神的好人会分享他们对于相似问题的解决办法

第四周

本周学习了git

git clone：从某个地址克隆代码

cd ：进入待上传的目录路径

git status:路径状态，如果和在线目录不同，就会提示git add

git add:增加文件

git commit:提示上传文件

git push：上传文件

大作业

写一个程序之前，要首先思考，如果想要达到目标，需要通过哪几步来完成

要将文件夹中的数据全部读取后，拆分成一个个单词，接着统计每个单词出现了多少次，并将结果输出到csv文件中，分为这样四个步骤

1. 读取文件
2. 拆分单词
3. 统计单词
4. 输出数据

需要引入两个类：

import codecs #引入 codecs 类

import os #引入 os 类

f=codecs.open(file\_path,'r',"utf-8")#使用codecs类中的open，括号里分别放路径、读取方法、编码，把file\_path路径所指向的文件用utf-8编码打开读取，并赋予给f变量

lines=f.readlines()#readlines表示读取所有行，组成列表后，赋予给lines变量，这里读取f变量里的那个文件，并变成一个列表lines

line=line.strip()#移除line中每个单词前后的空格

words=line.split(" ") #按空格划分单词

通过空格拆分单词后，发现一些非英文字符，因此还需要把单词格式化，此外，一些连接符连起来的单词也需要去掉，还要另外写一个分割单词模块

总的来说，大作业中，要熟悉列表、字典的用法，尤其是如何添加列表，在循环过程中为列表和字典赋值等等，在看着视频做完以后，需要自己独立再尝试做一次，加深记忆。

在大作业中，犯了两个错误：一是直接复制粘贴示例代码时，缩进方式不一致导致的问题；二是在为列表添加项时，习惯性加等号

此外，对于“如何排序”这个问题，通过上网搜索，了解了字典中的sorted命令，然而这个命令执行的结果是一个列表文件，其中的字典键和值变成了元祖，无法使用原来的命令将它写入csv文件中

这时，继续搜索，找到了一个新的类collections，通过使用其中的OrderedDict命令，生成了有序字典，成功写入了csv文件

启示：善用搜索