# 《Python语言与SageMath》课程练习题

## 1.    如何使用PyCharm创建一个Python程序，并运行它？

## 2.    在使用PyCharm中，你发现了哪些方便开发软件的功能，请至少举三个例 子。

## 3.    如何使用SageMath的网页笔记本界面编辑、运行科学计算的脚本？如何查 看帮助信息？总结出你认为最常用和方便的快捷功能键。

## 4.    通过阅读print函数的帮助文档，总结其各种使用方法，给出代码实例。

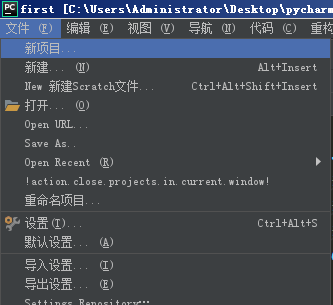
## 5.    查找资料，学习总结字符串format方法进行字符串格式化的各种使用方法， 给出代码实例。

6. 编写一个支持包括加法、减法、乘法、转置等矩阵运算的类，前三种运算在语法上直接支持使用运算符+、-、\*，矩阵元素所在的数域可自定。

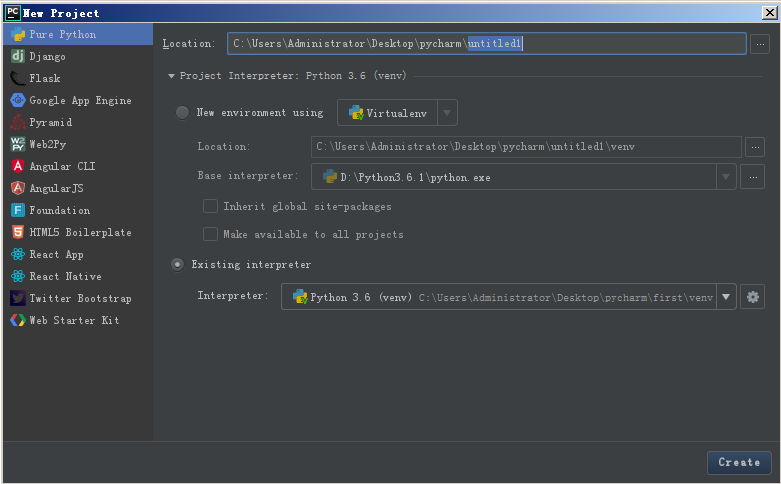
## 7.    选择一个小问题，编写一个小程序，代码、文档和单元测试都要有，尽可能 将所学的各种Python基本知识都运用进去（代码行数不少于100行，包括 文档和合理的格式化空行）。

## 1.如何使用PyCharm创建一个Python程序，并运行它？

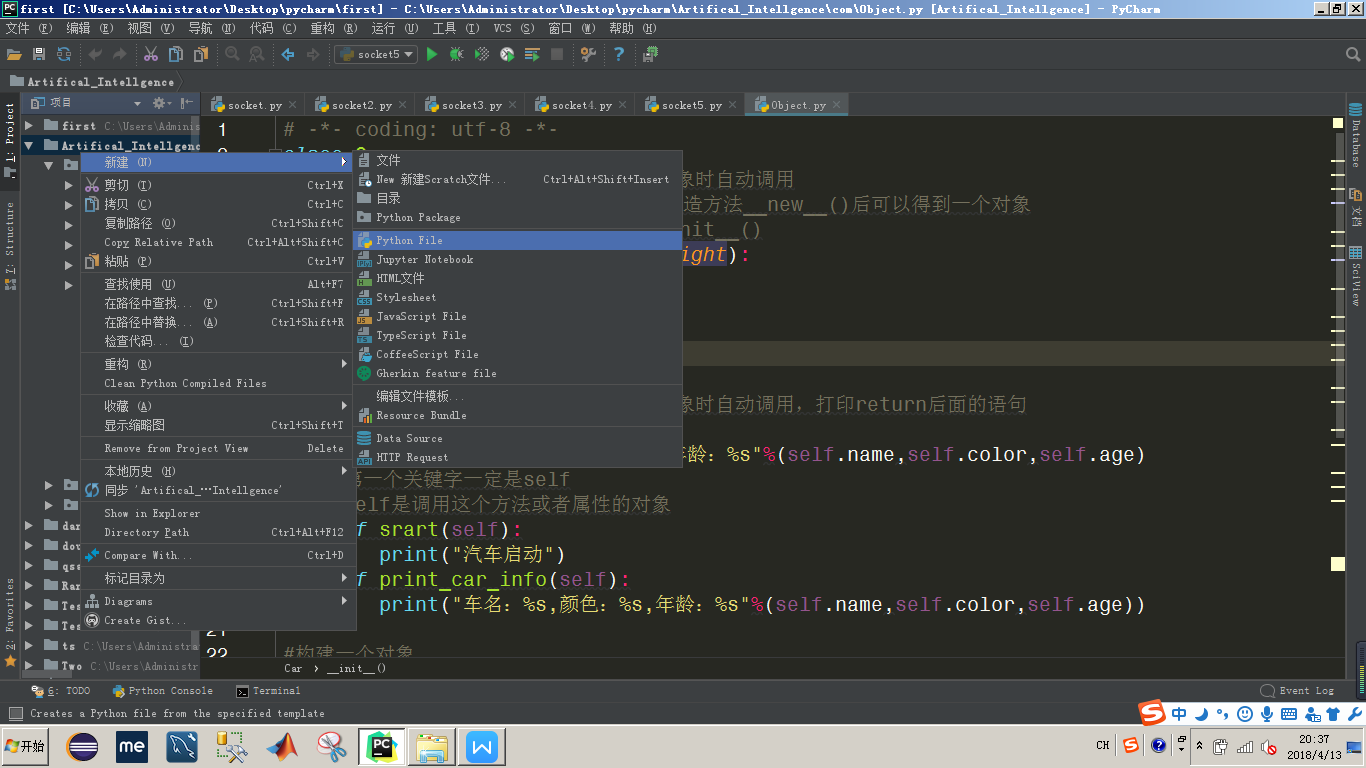
（1）单击菜单栏中的“文件”选择“新项目”



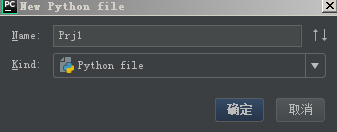
1. 选择新项目的保存路径、文件名和程序解释器



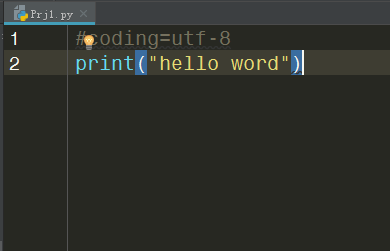
1. 在已经新建好的项目名中单击右键选择“新建”——“Python File”来新建 python文件



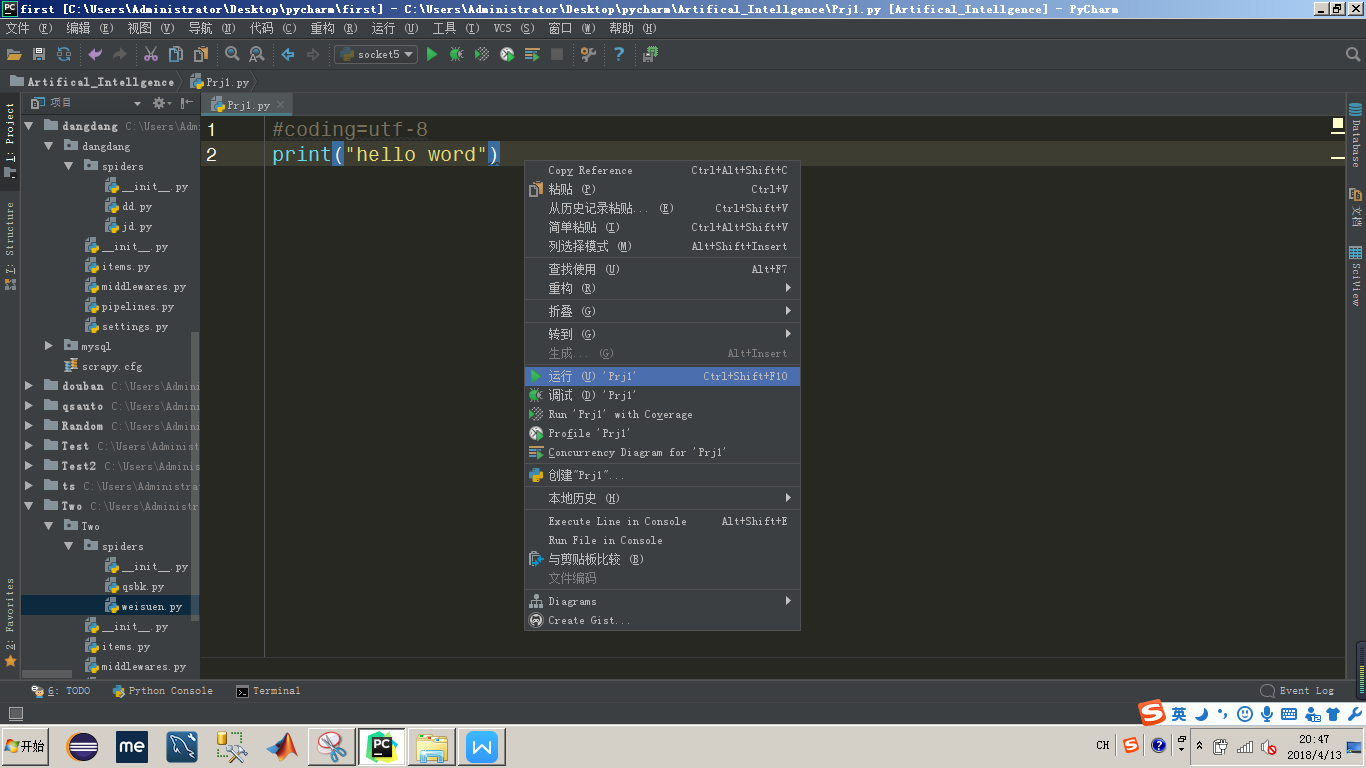
1. 输入python程序文件名单击确定就会在对应项目的路径下产生一个python 程序文件



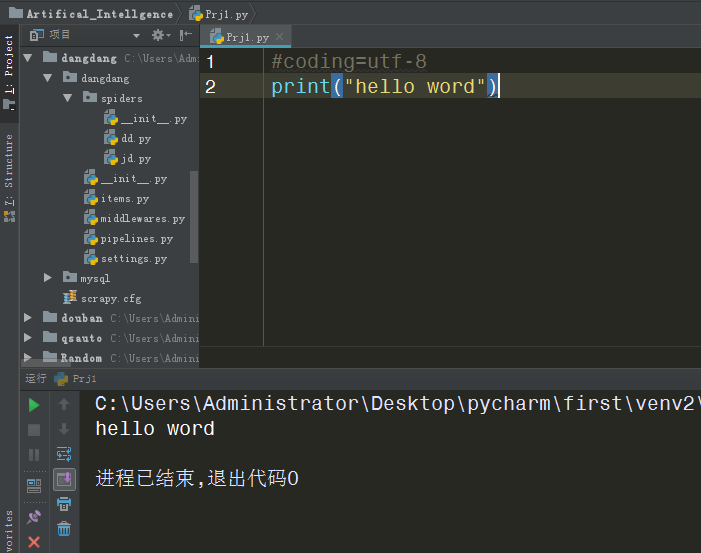
1. 现在就可以在刚刚创建好的python文件中编写代码了



1. 编写好代码后，可以在编辑域中任意单击右键，选择“运行”，就可以运行 当前编辑的文件了



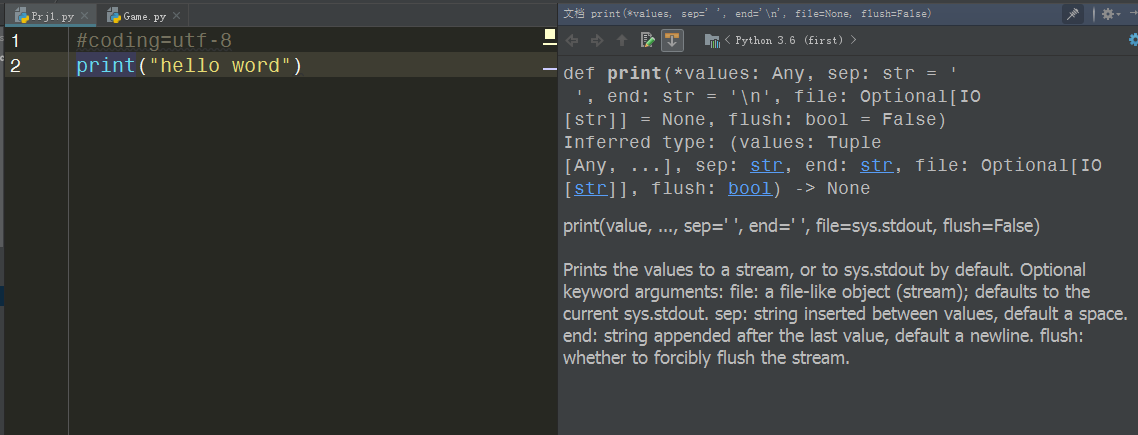
1. 最后的运行结果会在控制台中打印出来



## 2.在使用PyCharm中，你发现了哪些方便开发软件的功能，请至少举三个例子。

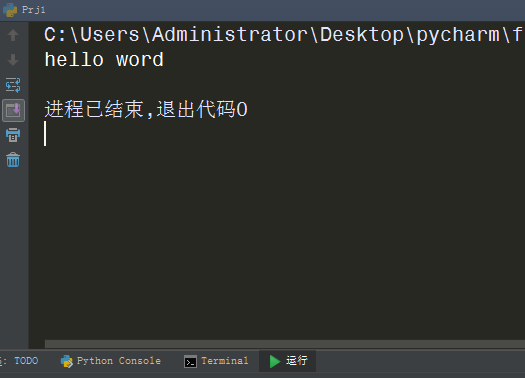
### 快速查看文档

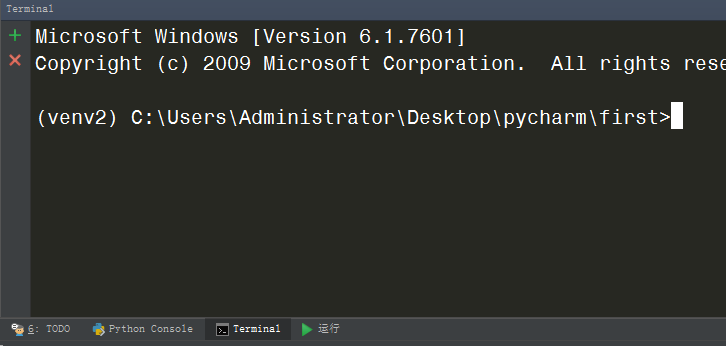
PyCharm中，如果遇到一个内建函数不知道怎么使用可以将光标点击到该函数，按下“Ctrl+Q”就可以立即看到该函数的使用方法，这样就我们就可以很方便地查看帮助文档。

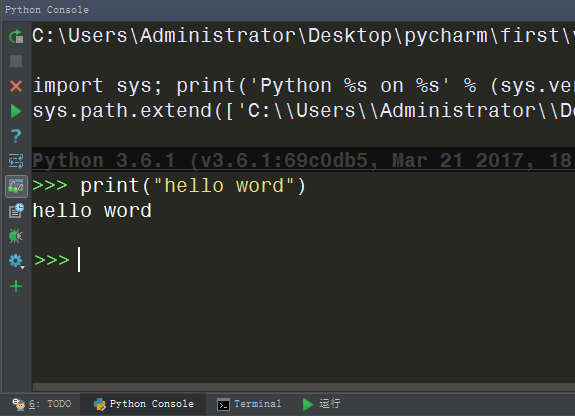


### 集成平台丰富

在pycharm的下方我们可以看到，pycharm把程序运行的控制台、windows的终端输入和python自带的“Python Console”都结合到了一起，这样大大提高了开发效率，可以很便捷地在下方自由切换相应的模式

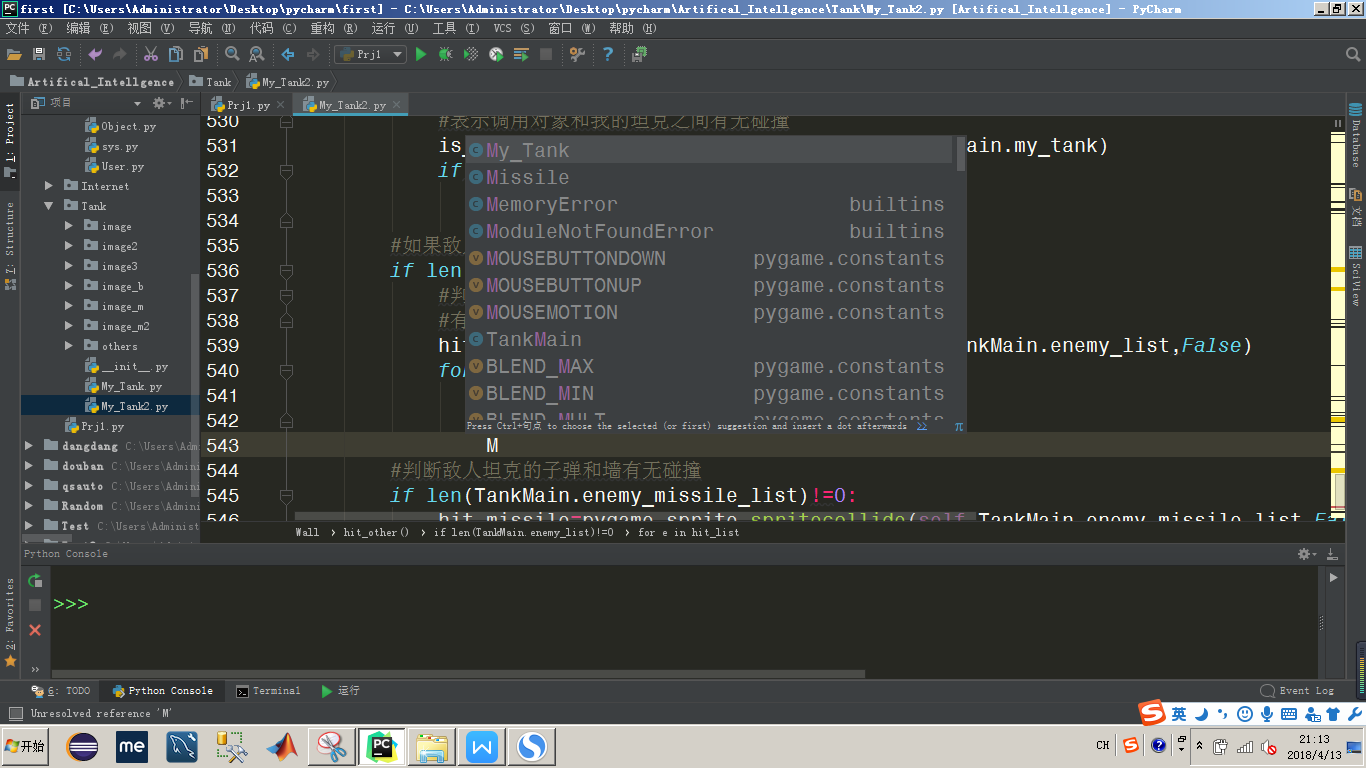


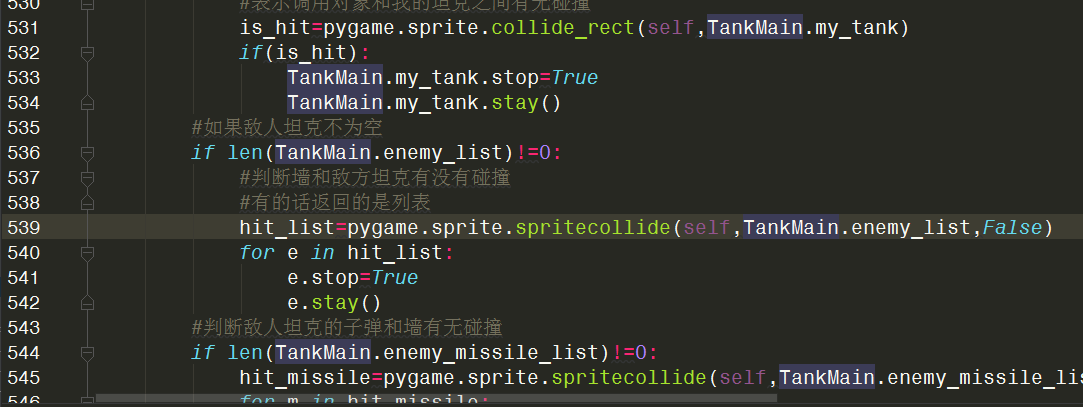




### 自动补全和快速提示

在编写一个大型程序时，我们往往会使用到许多变量，在类与类之间，函数与函数之间等的许多变量我们有时会忘记，这时我们只要输入变量的开头的几个字母，pycharm就会自动在当前光标先列出你可能会用到的变量；此外，当我们测试大型程序代码时，我只要把光标移动到一个变量中，pycharm就会自动高亮起相同变量的位置，这样大大增强了代码的可读性。

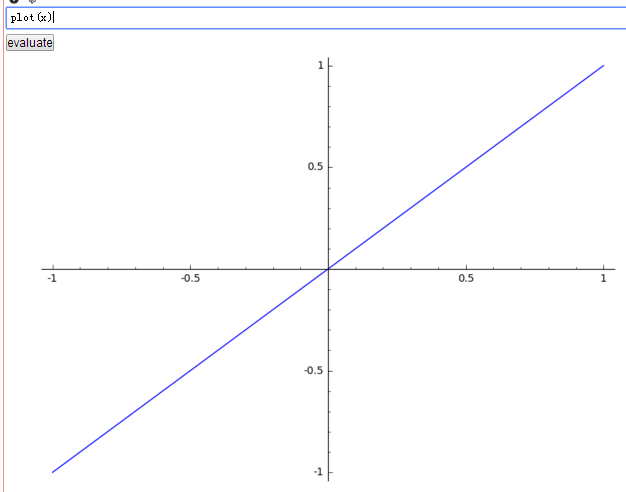


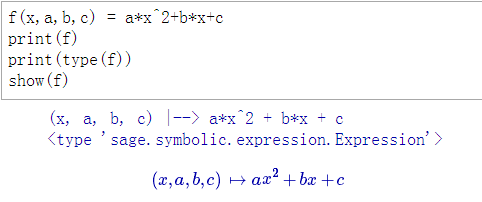


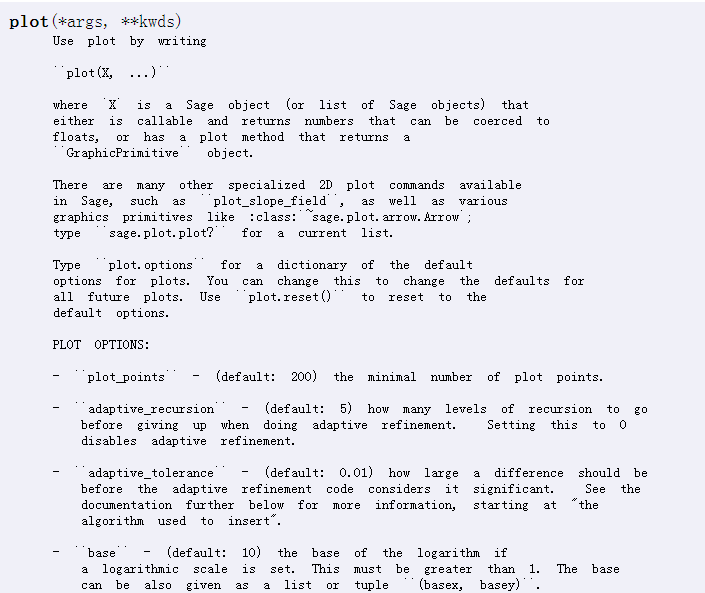
## 3.如何使用SageMath的网页笔记本界面编辑、运行科学计算的脚本？如何查看帮助信息？总结出你认为最常用和方便的快捷功能键。

在SageMath的网页笔记本界面中，在“compute cell”中可以编辑各种数学数学公式，在有效地结合python语言后可以直接在compute cell中编写代码，结合SageMath实现各种数学运算。

在SageMath的“compute cell”中，输入help(内建函数名)，如help(plot)就可以查看关于该函数的使用方法。



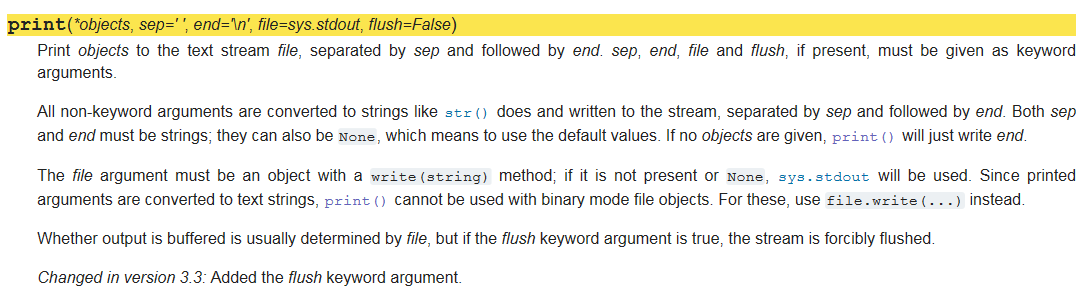




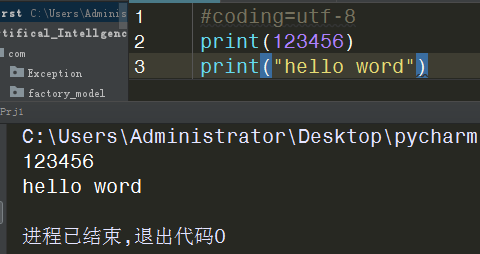
最常用和方便的快捷功能键主要有：自动补全键，只要在“compute cell”中输入pri+<TAB>SageMa就会自动补全相应的函数或者列出可能的函数；如果要查看当前会话中用户定义的所有变量的类型，可以使用“%whos”来查看等

## 4.通过阅读print函数的帮助文档，总结其各种使用方法，给出代码实例。

Python中print()函数帮助文档截图

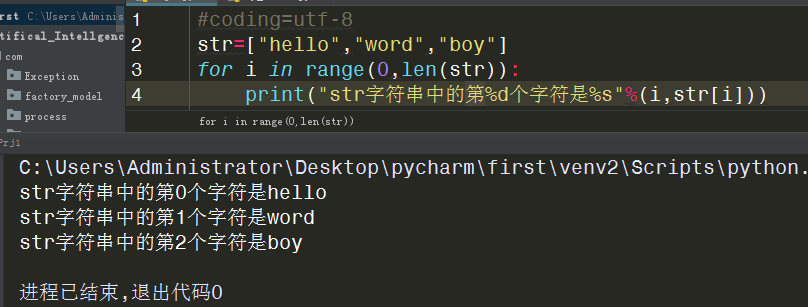


### 输出字符串和数值



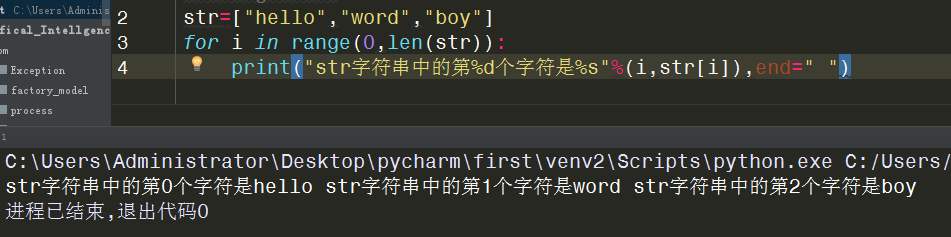
### 格式化输出

Print()中，如果要输出字符串中有带有参数，则需要格式化输出，python的print函数方便在于可以在字符串后面加：%(参数1，参数2……)来统一格式化



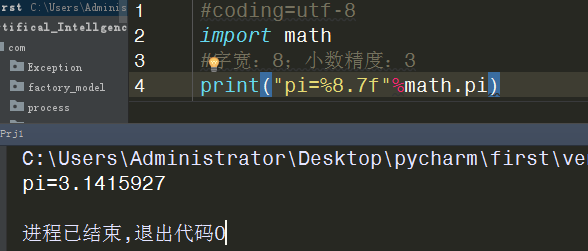
### 不换行

Print()会自动在行末加上回车, 如果不需回车，则要修改print(end=””)中的end属性值，end参数表示把什么字符串加到输出字符串的后面作为结束，默认是“\n”



### 格式化浮点数(float)

如果要求输出的小数需要指定位数和精度就要用到，格式化浮点数输出



## 5.查找资料，学习总结字符串format方法进行字符串格式化的各种使用方法，给出代码实例。

### （1）str.format()的引入

在 Python 中，我们可以使用 + 来连接字符串，在简单情况下这种方式能够很好的工作。但是当我们需要进行复杂的字符串连接时，如果依然使用 + 来完成，不仅会使代码变得晦涩难懂，还会让代码变得难以维护，此时这种方式就显得力不从心了。

例如，我们想打印这样一条记录：

User:Hutao has completed Action:payment at Time:13:30:00

如果使用加号实现，会是下面这种形式：



如果以后回过头来阅读这段代码，我们很难直观看出它的输出格式，且修改起来也相对麻烦。Python 为我们提供了另一种简洁优雅的实现方式：使用 str.format()来实现字符串的格式化：



str.format 既能够用于简单的场景，也能够胜任复杂的字符串替换，而无需繁琐的字符串连接操作。Python 的内置类型 str 和 unicode 均支持使用 str.format() 来格式化字符串。

### str.format()基本语法

格式化字符串使用花括号 {} 来包围替换字段，也就是待替换的字符串。而未被花括号包围的字符会原封不动地出现在结果中。

1. 使用位置索引

以下两种写法是等价的：

print("Hello,{} and {}".format("Hutao","Tom"))

print("Hello,{0} and {1}".format("Hutao","Tom"))

花括号内部可以写上目标字符串的索引，也可以省略。如果省略，则按 format 括号里的目标字符串顺序依次替换。

1. 使用关键字索引

除了通过位置来指定目标字符串，我们还可以通过关键字来指定它。

使用关键字索引的好处是，我们无需关心参数的位置，且字符串的最终结果能够一目了然。在以后的代码维护中，我们能够快速地修改对应的参数，而不用对照字符串挨个去寻找相应的参数。

## 6. 编写一个支持包括加法、减法、乘法、转置等矩阵运算的类，前三种运算在语法上直接支持使用运算符+、-、\*，矩阵元素所在的数域可自定。

程序：

import numpy

#在pycharm可以运行，sagemath报错

class GF5():

size = 5

\_reciprocals = (0, 1, 3, 2, 4)

def \_\_init\_\_(self, value):

self.\_value = int(value) % GF5.size

#机器可读

def \_\_repr\_\_(self):

return 'GF5:{0}'.format(self.\_value)

#向外显示

def \_\_str\_\_(self):

return str(self.\_value)

def \_\_int\_\_(self):

return self.\_value

#判断两个对象是否相等

def \_\_eq\_\_(self, other):

return int(self) == int(other)

def \_\_add\_\_(self, other):

return GF5(int(self) + int(other))

def \_\_sub\_\_(self, other):

return GF5(int(self) - int(other) + GF5.size)

def \_\_mul\_\_(self, other):

return GF5(int(self) \* int(other))

def \_\_truediv\_\_(self, other):

other\_value = int(other)

if other\_value == 0:

raise ZeroDivisionError

return GF5(int(self) \* GF5.\_reciprocals[other\_value])

class Matria():

#复制矩阵

def copy\_mat(self,mat):

result = [[0] \* len(mat[0])] \* len(mat)

result = numpy.array(result)

for row\_index, row in enumerate(mat):

for col\_index, col in enumerate(row):

result[row\_index][col\_index] = GF5(int(col))

return result

def \_\_init\_\_(self, mat):

self.data = self.copy\_mat(mat)

print(self.data)

def \_\_repr\_\_(self):

return str(self.data)

def \_\_str\_\_(self):

return Matria.\_\_repr\_\_(self)

#获得列表中的元素

def \_\_getitem\_\_(self, item):

return self.data[item]

#设置列表值

def \_\_setitem\_\_(self, key, value):

self.data[key] = GF5(value)

def \_\_eq\_\_(self, other):

if (isinstance(other, Matria)):

for x, y in zip(self, other):

if x != y:

break

else:

return True

return False

def judge(self,other):

if ((len(self.data) != len(other.data)) or (len(self.data[0]) != len(other.data[0]))):

return True

else:

return False

#加法

def \_\_add\_\_(self, other):

print("矩阵相加")

if (self.judge(other)):

print("不满足矩阵的加法条件")

else:

#得到self的矩阵

result = self.copy\_mat(self.data)

for row\_index,other\_row in enumerate(other.data):

for col\_index,other\_col in enumerate(other\_row):

result[row\_index][col\_index]+=other\_col

return Matria(result)

#减法

def \_\_sub\_\_(self, other):

print("矩阵相减")

if (self.judge(other)):

print("不满足矩阵的减法条件")

else:

result = self.copy\_mat(self.data)

for row\_index,other\_row in enumerate(other.data):

for col\_index,other\_col in enumerate(other\_row):

result[row\_index][col\_index]-=other\_col

return Matria(result)

#乘法

def \_\_mul\_\_(self, other):

print("矩阵相乘")

#第一个矩阵的列和第二个矩阵的行相等才能相乘

if(len(self.data[0])!=len(other.data)):

print("不满足矩阵的乘法条件")

else:

#第一个矩阵的行

result\_row=len(self.data)

#第二个矩阵的列

resule\_col=len(other.data[0])

result = [[0] \* resule\_col] \* result\_row

result = numpy.array(result)

for row in range(result\_row):

for col in range(resule\_col):

for i in range(resule\_col):

result[row][col]+=self.data[row][i]\*other.data[i][col]

return Matria(result)

#转置

def tran(self,mat):

print("矩阵转置")

#获得原始矩阵的列给转置矩阵的行

res\_row=len(mat[0])

#获得原始矩阵的行给转置矩阵的列

res\_col=len(mat)

result=[[0] \* res\_col] \* res\_row

result = numpy.array(result)

for row in range(res\_row):

for col in range(res\_col):

result[row][col]=mat[col][row]

return Matria(result)

if \_\_name\_\_=="\_\_main\_\_":

print("有限域：5")

mat1=[[1,2,3],[4,5,6],[7,8,9]]

mat2=[[10,11,12],[13,14,15],[16,17,18]]

print("原始矩阵1：")

matria1=Matria(mat1)

print("原始矩阵2：")

matria2=Matria(mat2)

matria1+matria2

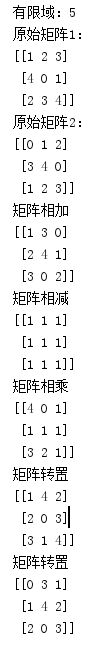
matria1-matria2

matria1\*matria2

matria1.tran(mat1)

matria1.tran(mat2)

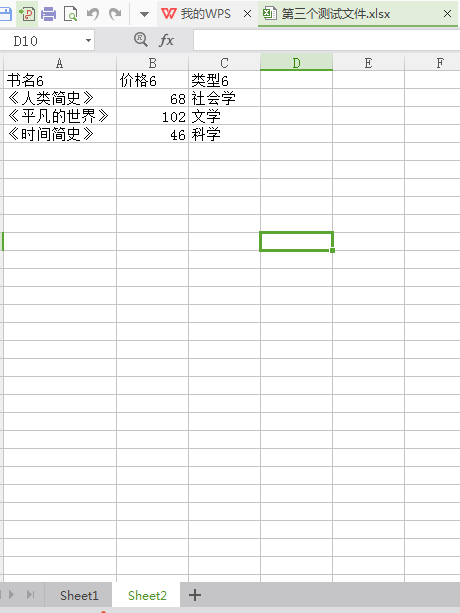
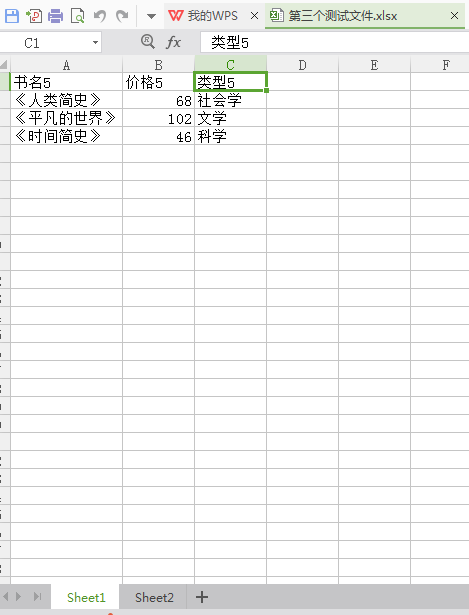
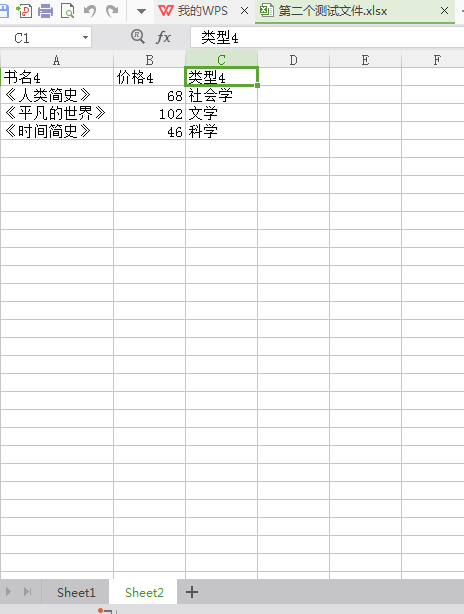
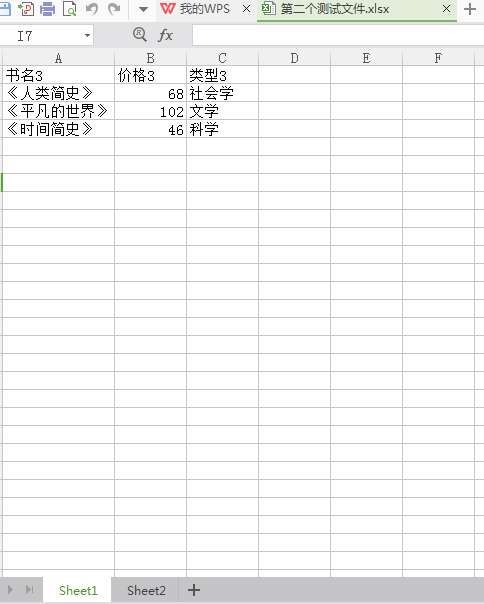
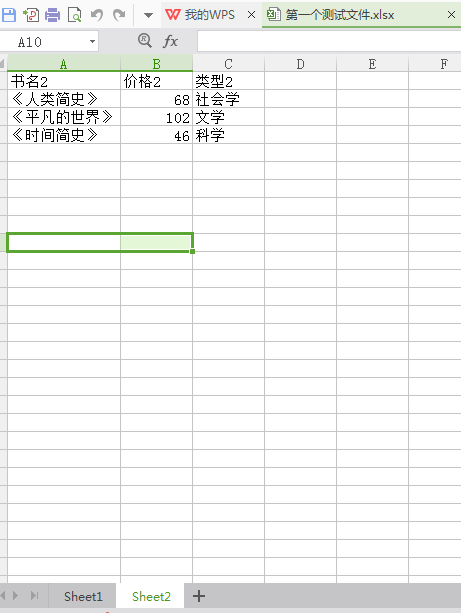
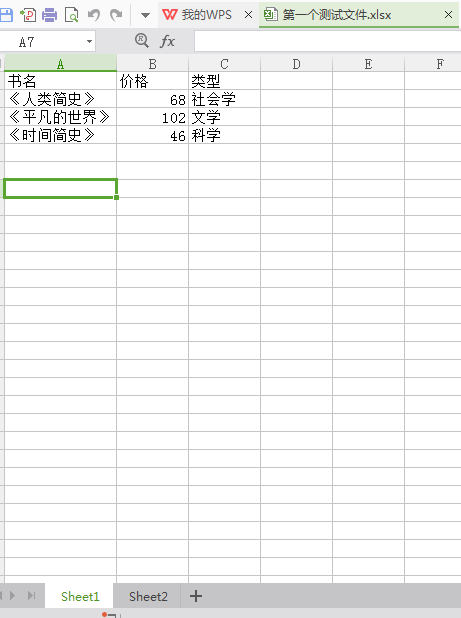
执行结果：



## 7. 选择一个小问题，编写一个小程序，代码、文档和单元测试都要有，尽可能将所学的各种Python基本知识都运用进去（代码行数不少于100行，包括文档和合理的格式化空行）。

问题：读取多个Excel文件中的多个sheet内容，最后把它们整合到一个Excel的一个sheet中，既是实现多个Excel表格的合并。

源文件：



程序：

# coding=utf-8

# xlrd是读excel，xlwt是写excel的库

import xlrd

import xlwt

'''

读取多个Excel文件中的多个sheet内容，最后把它们整合到一个Excel的一个sheet中

'''

#创建读写Excel类

class Excel(object):

'''

功能：创建读写xls类

参数：继承顶级父类object

'''

#定义各种类属性

# 设置要合并的所有文件名，放入一个列表之中

allxls = ["F:/第一个测试文件.xlsx", "F:/第二个测试文件.xlsx", "F:/第三个测试文件.xlsx"]

# 设置最后合并的文件名

endxls = "F:/endxls.xlsx"

# 存储所有读取结果，变成列表

filevalue = []

# 存储一个sheet的结果

svalue = []

# 存储sheet中的每一行结果

rvalue = []

# 存储各sheet名

shname = []

def \_\_init\_\_(self):

pass

# 打开表格，传入需要打开的xls文件

def open\_xls(self,file):

'''

打开表格，传入需要打开的xls文件

:param file: 需要打开的文件

:return: 打开文件的句柄

'''

try:

# 使用模块，创建/打开xls文件的句柄

fh = xlrd.open\_workbook(file)

return fh

except Exception as e:

# 如果打开文件出错，则执行下面语句，并输出具体错误

print("打开出错，错误是：")

print(e)

# 获取一个xls中的所有sheet

def getsheet(self,fh):

'''

获取一个xls中的所有sheet

:param fh: 打开的xls文件

:return: 一个xls中的所有sheet

'''

# 由于fh的类型是Excel，所以可以获取内部的sheet

return fh.sheets()

# 读取某个sheet的行数

def getnrows(self,fh, sheet):

'''

读取某个sheet的行数

:param fh: 打开的xls文件

:param sheet: 该xls文件的sheet

:return: sheet的行数

'''

# 打开该sheet，并给一个句柄变量

table = fh.sheets()[sheet]

# 获得某个sheet的行数

content = table.nrows

return content

# 读取某个xls文件的内容并返回所有内容

# 形参是文件句柄、文件、某个sheet号

def getfilect(self,fh, fl, shnum):

'''

读取某个xls文件的内容并返回所有行值

:param fh: 文件句柄

:param fl: 文件

:param shnum: 某个sheet号

:return: 返回所有内容

'''

# 调用打开xls文件函数，并把句柄返回

fh = self.open\_xls(fl)

# 把该文件的某个sheet的名字给table，通过shnum可以获取某个特定的sheet

table = fh.sheet\_by\_name(Excel.shname[shnum])

# 把句柄和sheet号，获得某个sheet的行数，就是共有几行

num = self.getnrows(fh, shnum)

# 把sheet中的某一行的长度求出来

lenrvalue = len(Excel.rvalue)

# 存储某个xls中某个sheet的某一行内容

# 循环行数后，就是把整个sheet的内容都加入rvalue

# 最后rvalue的列表的每个元素就存放了该sheet中的每行结果

for row in range(0, num):

rdata = table.row\_values(row)

Excel.rvalue.append(rdata)

# print(rvalue[lenrvalue:])

# 把某sheet一整行的内容加入到整个数据变量中

# 就是filevalue的某个元素（因为是列表）是某个sheet中的全部内容

# 防止重复代码

Excel.filevalue.append(Excel.rvalue[lenrvalue:])

return Excel.filevalue

#获取每个xls文件内容，并写入最后的xls文件中

def read(self):

'''

获取每个xls文件内容，并写入最后的xls文件中

:return:最后写入的xls文件

'''

# 读取第一个待读文件，获得sheet数

fh = self.open\_xls(Excel.allxls[0])

# 获得该文件中的sheet数量，最后是一个列表

sh = self.getsheet(fh)

x = 0

for sheet in sh:

# 获取每个sheet的名字

Excel.shname.append(sheet.name)

Excel.svalue.append([])

x = x + 1

# 依次读取各sheet的内容

# 依次读取各文件当前sheet的内容

for shnum in range(0, x):

# 遍历allxls中的每个文件名给fl

for fl in Excel.allxls:

print("正在读取文件：" + str(fl) + "的第" + str(shnum) + "个sheet的…")

Excel.filevalue = self.getfilect(fh, fl, shnum)

Excel.svalue[shnum].append(Excel.filevalue)

# print(svalue[0])

# print(svalue[1])

# 由于append具有叠加关系，分析可得所有信息均在svalue[0][0]中存储

# svalue[0][0]元素数量问sheet标签数(sn)\*文件数(fn)

# sheet个数

sn = x

# xls文件数

fn = len(Excel.allxls)

endvalue = []

# 设置一个函数专门获取svalue里面的数据，即获取各项sheet的数据

def getsvalue(k):

for z in range(k, k + fn):

endvalue.append(Excel.svalue[0][0][z])

return endvalue

# 打开最后写入的文件

wb1 = xlwt.Workbook(Excel.endxls)

# 创建一个sheet工作表对象,命名是1

ws = wb1.add\_sheet("1")

polit = 0

linenum = 0

# 依次遍历每个sheet中的数据

for s in range(0, sn \* fn, fn):

thisvalue = getsvalue(s)

# [sheet个数][文件编号][文件行数]

tvalue = thisvalue[polit:]

# 将一个标签的内容写入新的文件中

for a in range(0, len(tvalue)):

for b in range(0, len(tvalue[a])):

for c in range(0, len(tvalue[a][b])):

# print(linenum)

# print(c)

data = tvalue[a][b][c]

ws.write(linenum, c, data)

linenum = linenum + 1

# 叠加关系，需要设置分割点

polit = len(thisvalue)

print("endxls.xlsx保存成功")

wb1.save(Excel.endxls)

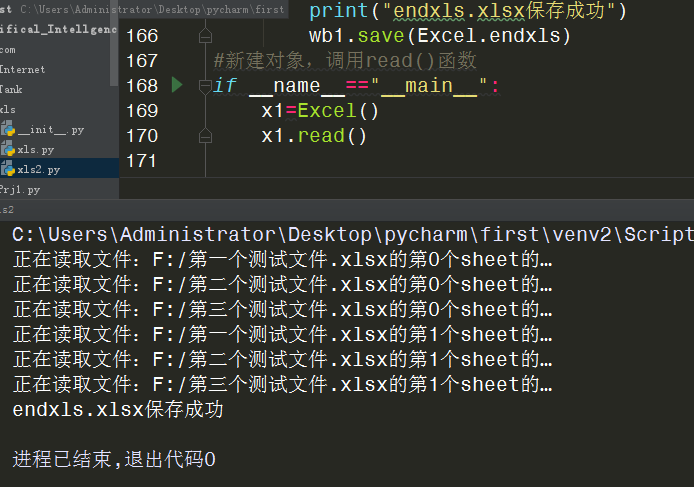
#新建对象，调用read()函数

if \_\_name\_\_=="\_\_main\_\_":

x1=Excel()

x1.read()

运行结果：



保存的Excel文件：

