day13：系统模块，自定义模块

笔记本：	Python基础		
创建时间：	2018/6/23 11:14	**更新时间：**	2018/7/2 19:38
作者：	liuchang_0412@163.com		

1.系统模块 sys：与系统相关的信息

变量：

sys.path　模块搜索路径 path[0]是当前脚本程序的路径名，或者是''　**#返回列表**

sys.modules　已加载模块的**字典**

sys.version　版本信息

sys.version_info　版本信息的命名元组

sys.argv　命令行参数， argv[0]代表当前脚本程序路径名。示例见：code1.py

sys.copyright　获取python版权相关信息

sys.builtin_module_names　获得python内建模块名称（字符串元组）

#练习：写一个程序myadd.py，执行此程序。如果执行如下命令:

用法：./myadd.py 数字 运算符 数字

$./myadd.py 5 加 2　结果是7

$./myadd.py 5 乘 2　结果是10

函数：

sys.exit([code])　退出程序，正常退出时sys.exit(0)

sys.getrecursionlimit( )　得到递归的层次限制值

sys.setrecursionlimit(n)　设置递归的最大层次限制值

2.自定义模块的编写：

略

3.模块的搜索路径：

#import 模块名　#对应 模块名.py 去哪儿找

①查找顺序：搜索内置模块>sys.path提供的路径>搜索程序运行时路径（当前路径）

②添加地址至 sys.path　#示例：mod_path.py

#练习：把昨天的联系相应的函数写一个模块叫day12.py，再写一个主模块来调用相应的函数

4.PYTHONPATH环境变量

此环境变量的值会在Python3的解释进行器启动时，自动加载到sys.path列表中

$export PYTHONPATH=$PATHONPATH:/home/tarena/

printenv　#Linux/Unix下查看所有的环境变量

5.模块的加载过程

①在模块导入时，模块内的所有语句会执行
②如果一个模块已经导入，再次导入时不会重新执行所有语句
示例：module.py

6.模块化编程的优点：
①有利于多人共同开发
②使代码更加易于维护
③提高代码的复用率
④模块化编程有助于解决函数名和变量名冲突的问题

7.模块的属性：
①__name__属性：用于记录模块自身的名字
1)对于被导入模块，模块名为去掉路径前缀和".py"后缀的文件名
2)对于被执行的主模块，模块名为"__main__"
作用：
1)记录模块名
2)用来判断是否为主模块
说明：以双下划线开头，以双下划线结尾的标识符通常代表Python的特殊属性等
②__doc__属性：用来绑定模块的文档字符串
模块的文档字符串是模块中第一行出现的没赋值给变量的字符串
③__all__属性：用来存储可导出属性的列表

作用：
当用from import * 语句导入模块时，只导入__all__列表内的变量（属性）
④__file__属性：用来记录模块对应的文件路径名

8.模块的隐藏属性：
模块中以 '_' 或以 '__' 开头，不以 '__' 结尾的属性，在用from import * 语句导入时，将不被导入到其他模块
9.标准库模块：
①随机模块 random
#假设导入 import random as R
函数：
R.random( )　#返回 [0,1)之间的随机数
R.getrandbits(nbit)　#以长整型的形式返回用nbit位**（二进制）**来表示的随机数
R.uniform(a,b)　#返回[a,b)区间内的随机数
R.randrange([start,] stop [, step])　#返回range(start, stop, step)中的随机数
R.choice(seq)　#从序列中返回随意元素
R.shuffle(seq)　#随机指定序列的顺序（乱序序列）
R.sample(seq, n)　#从序列中选择个随机且不重复的元素

示例：

```
import random as R
print(R.randon( ))
```

#练习：写一个猜数字游戏的程序，随机生成一个0~100之间的整数，
保存在变量x内，让用户输入一个数y，输出猜数字的结果。
如果y=x，提示“恭喜你猜对了”并退出程序。
如果y>x，提示“您猜的数大了”，并继续输入猜测。
如果y<x，提示“您猜的数小了”，并继续输入猜测。
直到猜对为止。

10.包（模块包） package：

①定义：将模块以文件夹的组织形式进行分组管理的方法

②作用：

将一系列模块进行分类管理，有利于防止名字冲突

可以在需要时加载一个或部分模块而不是全部模块

③包的加载：

```
import 包名 as 包别名
import 包名.模块名 as 模块别名
import 包名.子包名.模块名 as 模块别名
.....
from 包名 import 模块名 as 模块别名
from 包名.子包名 import 模块名 as 模块别名
from 包名.模块名 import 函数名 as 函数别名
from 包名 import *
.....
```

④包内的__init__.py文件

作用：在包被加载时自动调用

1.在内部填写包的文档字符串

2.加载此包所依赖的一些模块或其他包

__init__.py内的__all__属性

作用：用来记录哪些包需要导入，当from import *语句导入时，自查找__all__中所列出的模块

⑤包的加载路径：同模块相同，设置方法：

1)可以设置sys.path

2)可以设置PATHONPATH环境变量

⑥模块的加载过程：

```
          编译              解释执行
menu.py  -->  menu.pyc  -->  python3
```

示例：

```
mypack/
    __init__.py
```

```
menu.py
games/
    __init.py
    contra.py
    supermario.py
    tanks.py
office/
    __init__.py
    excel.py
    word.py
    powerpoint.py
```

#练习：

1.猜数字游戏2：有0~9十个数字，分别放在4个盒子内（列表中放四个元素，不能重复），

让用户每次输入四个数字，如果[4,6,0,3]：

#4603　输出全队，程序结束

#4601　3A0B（A代表位置对，数字也对；B代表数字对，位置不对）

#1046　0A3B

直到猜对为止

#结题思路：

#初始化

利用 random.sample(seq, n) 函数取出4个数，存入列表中

利用 random.shuffle(seq) 函数打乱次序，目标数字定义完成

#数字录入函数

录入四个数字（如果录入的不是数字报错），并把数字从字符串转换成四个数字并存入列表中

#判断函数

将录入的数字逐位进行比较，先确定是否位置相同，再确定是否存在。

如果位置相同，则A计数器+=1，如果位置不相同但存在，则B计数器+=1，否则pass。

如果四个数字都是位置相同，则程序结束。

#显示函数

对判断函数得出的结果进行显示

#主函数

不断循环数字录入函数，判断函数和显示函数，直至程序结束。

2.模拟斗地主发牌：扑克牌共54张，黑桃（'\u2660'）梅花（'\u2663'）方块（'\u2665'）红桃（'\u2666'）

2-10JQKA，大小王。三个人每个人发17张牌，底牌留三张：

输入<回车>打印第一个人的17张牌

输入<回车>打印第二个人的17张牌
输入<回车>打印第三个人的17张牌
输入<回车>打印三张底牌

#结题思路：
#初始化
创建一个列表，将代表54张牌的元组分别存入到列表中。
利用 random.shuffle(seq) 函数打乱列表次序，目标数字定义完成
#发牌函数
创建三个列表，分别代表三个玩家的牌库
将打乱次序后的列表弹出末尾的一个元组，并依次存入玩家的牌库中，每个玩家存入17张，留下3张
#打印函数
发牌完成后，依次打印三个玩家的牌库和留下的三张底牌

3.已知有五位朋友在一起，
第五位朋友比第四位大2岁，
第四位朋友比第三位大2岁，
第三位朋友比第二位大2岁，
第二位朋友比第一位大2岁，
第一位朋友说他10岁，试写递归程序算出第五位朋友几岁？

#结题思路：
#初始化录入
录入几个朋友见的年龄差，并录入第一位朋友的年龄（假设朋友数量固定为5人）
#年龄求解
创建一个函数fn(x)其返回值是第x位朋友的年龄
运用递归方法求解第五位朋友的年龄