**Python2-2**

**def 语句**

**• 函数用def语句创建,语法如下:**

**def func(on\_name(arguments):**

**"func(on\_documenta(on\_string"**

**func(on\_body\_suite**

**• 标题行由def关键字,函数的名字,以及参数的集合(如果有的话)组成**

**• def子句的剩余部分包括了一个虽然可选但是强烈推荐的文档字串,和必需的函数体**

**• 函数不允许在函数未声明之前对其进行引用或者调用内部函数**

**• 在函数体内创建另外一个函数是完全合法的,这种函数叫做内部/内嵌函数**

**• 使用一对圆括号()调用函数,如果没有圆括号,只是对函数的引用**

**• 任何输入的参数都必须放置在括号中**

**• 关键字参数的概念仅仅针对函数的调用**

**• 这种理念是让调用者通过函数调用中的参数名字来区分参数**

**• 这样规范允许参数缺失或者不按顺序**

**• python允许程序员执行一个没有显式定义参数的函数**

**• 相应的方法是通过一个把元组(非关键字参数)或字典(关键字参数)作为参数组传递给函数**

**1. 随机生成两个100以内的数字**

**2. 随机选择加法或是减法**

**3. 总是使用大的数字减去小的数字**

**4. 如果用户答错三次,程序给出正确答案**

**import** random  
  
# def add(x, y):  
# return x + y  
# def sub(x,y):  
# return x - y  
**def exam():** cmds **= {'+':lambda** x,y**:**x **+**y,**'-':lambda** x,y**:**x **-**y**}** nums**=[**random.randint**(**1,100**) for** i **in** range**(**2**)]** nums.sort**(**reverse**=True)** op **=** random.choice**('+-')** promot**= '%s%s%s=' %(**nums**[**0**]**,op,nums**[**1**])** result **=** cmds**[**op**](\***nums**)** # answer = int(input(promot))  
 tries **=** 0  
 **while** tries **<**3**:  
 try:** answer **=** int**(**input**(**promot**))  
 except :  
 continue  
 if** result**==** answer**:** print**('good')  
 break  
 else:** print**('lose')** tries **+=**1  
 **else:** print**('%s%s' % (**promot,result**))  
  
if** \_\_name\_\_ **== '\_\_main\_\_':  
 while True:** exam**()  
 try:** yn **=** input**('continue(y/n)')  
 except** IndexError**:  
 continue  
 except (**KeyboardInterrupt,EOFError**):  
 break  
 if** yn **in 'Nn':** print**('\nBye')  
 break**

**lambda**

**• python允许用lambda关键字创造匿名函数**

**• 匿名是因为不需要以标准的def方式来声明**

**• 一个完整的lambda“语句”代表了一个表达式,这个表达式的定义体必须和声明放在同一行**

**filter()函数**

**• filter(func, seq):调用一个布尔函数func来迭代遍历每个序列中的元素;返回一个使func返回值为true的元素的序列**

**• 如果布尔函数比较简单,直接使用lambda匿名函数就显得非常方便了**

# from random import randint  
# def func1(n):  
# return n % 2  
# def func2(n):  
# return n +1  
# if \_\_name\_\_ == '\_\_main\_\_':  
# nums = [randint(1,100) for i in range(10)]  
# print(nums)  
# print(list(filter(func1,nums)))  
# print(list(filter(lambda n : n%2,nums)))  
# print('#'\*20)  
# print(list(map(func1,nums)))  
# print(list(map(func2,nums)))  
# print(list(filter(func2,nums)))  
# print(list(filter(func1,nums)))

**全局变量**

**• 标识符的作用域是定义为其声明在程序里的可应用范围,也就是变量的可见性**

**• 在一个模块中最高级别的变量有全局作用域**

**• 全局变量的一个特征是除非被删除掉,否则它们的存活到脚本运行结束,且对于所有的函数,他们的值都是可以被访问的**

**局部变量**

**• 局部变量只时暂时地存在,仅仅只依赖于定义它们的函数现阶段是否处于活动**

**• 当一个函数调用出现时,其局部变量就进入声明它们的作用域。在那一刻,一个新的局部变量名为那个对象创建了**

**• 一旦函数完成,框架被释放,变量将会离开作用域**

**• 如果局部与全局有相同名称的变量,那么函数运行时,局部变量的名称将会把全局变量名称遮盖住**

**• 因为全局变量的名字能被局部变量给遮盖掉,所以为了明确地引用一个已命名的全局变量,必须使用global语句**

**• 任何时候,总有一个到三个活动的作用域(内建、全局和局部)**

**• 标识符的搜索顺序依次是局部、全局和内建**

**• 提到名字空间,可以想像是否有这个标识符**

**• 提到变量作用域,可以想像是否可以“看见”这个标识符**

**偏函数:相当于是改造现有的函数 将其一部分参数固定下来**

**• 偏函数的概念是将函数式编程的概念和默认参数以及可变参数结合在一起**

**• 一个带有多个参数的函数,如果其中某些参数基本上固定的,那么就可以通过偏函数为这些参数赋默认值**

**• 如果函数包含了对其自身的调用,该函数就是递归的**

**• 在操作系统中,查看某一目录内所有文件、修改权限等都是递归的应用**

**def fun(**n**):  
 if** n **==** 1 **:  
 return** 1  
 **return** n**\***fun**(**n**-**1**)  
if** \_\_name\_\_ **== '\_\_main\_\_':** n**=** int**(**input**('s'))** print**(**fun**(**n**))**

**1. 随机生成10个数字**

**2. 利用递归,实现快速排序**

**from** random **import** randint  
**def quci(**number**):  
 if** len**(**number**) <**2**:  
 return** number  
 mid **=** number**[**0**]** lager **= []** samll **= []  
 for** item **in** number**[**1**:]:  
 if** item **>** mid**:** lager.append**(**item**)  
 else:** samll.append**(**item**)  
 return** quci**(**samll**)+[**mid**]+**quci**(**lager**)  
if** \_\_name\_\_ **== '\_\_main\_\_':** nums **= [**randint**(**1,100**) for** i **in** range**(**10**)]** print**(**nums**)** print**(**quci**(**nums**))**