**函数基础**

创建函数

def语句

函数用def语句创建，语法如下：

def function\_name(arguments):

“function\_documentation\_string”

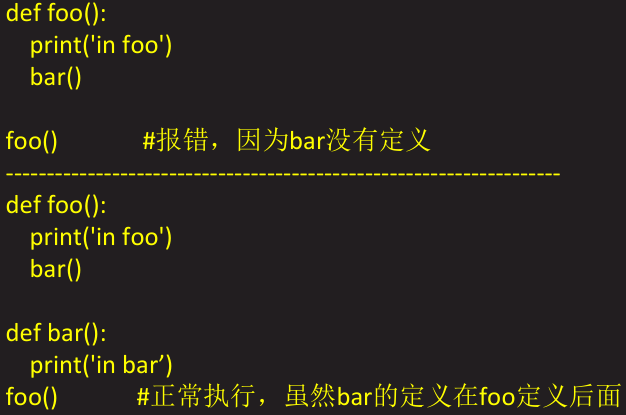
function\_body\_suite

标题行由def关键字组成，函数名字，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符串，和必须的函数体

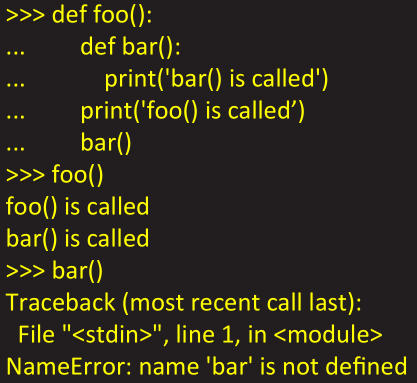
前向引用

函数不允许在函数申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

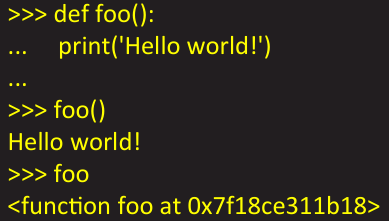


调用函数

函数操作符

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

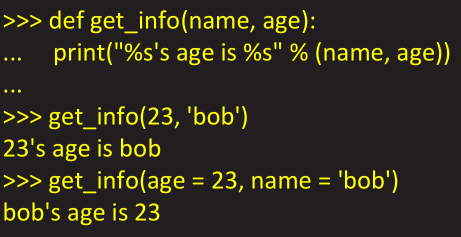


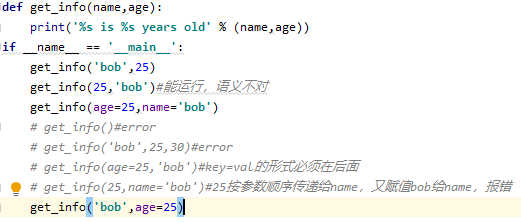
关键字参数

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



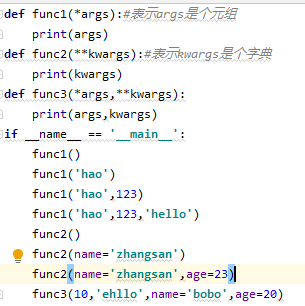


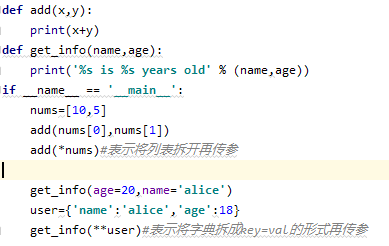
参数组

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







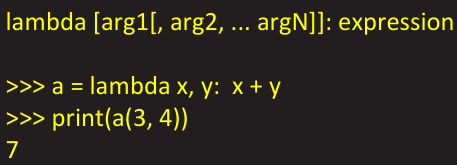
匿名函数

lambda

python允许用lambda关键字创造匿名函数

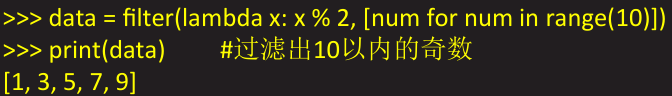
匿名是因为不需要以标准的def方式来申明

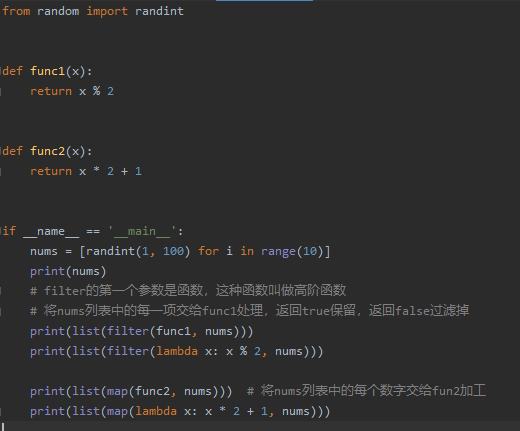
一个完整的lambda“语句”代表了一个表达式，这个表达式的ingyi体必须和申明放在同一行



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





**函数的高级应用**

变量作用域

全局变量

标识符的作用域是定义为其申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掉，否则它们存活到脚本运行结束，且对于所有的函数，它们都是可以被访问的

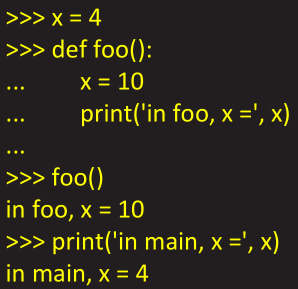
局部变量

局部变量只是暂时存在，仅仅只依赖于定义它们的函数现阶段是否处于活动状态

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

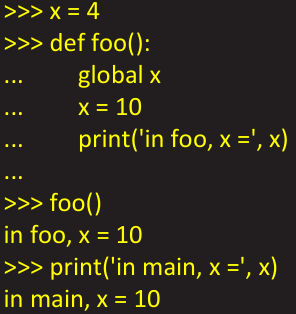
一旦函数创建完成，框架被释放，变量将会离开作用域

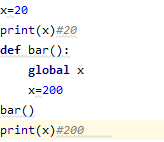
如果局部变量与全局变量由相同名称的变量，那么函数运行时，局部变量名称将会把全局变量名称盖住



global语句

因为全局变量的名字能被局部变量给覆盖掉，所以为了明确的引用一个已命名的全局变量，必须使用global语句





名字空间

任何时候，总有一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、全部和内建

提到名字空间，可以想像是否有这个标识符

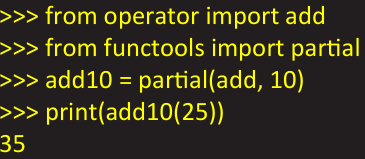
提到变量的作用域，可以想像是否可以“看见”这个标识符

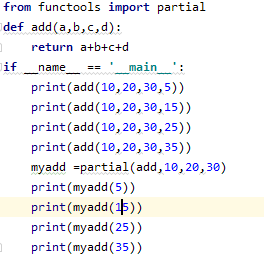
函数式编程

偏函数

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



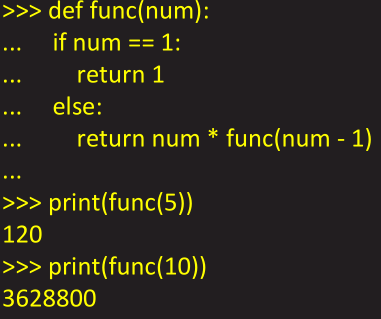


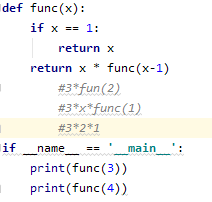


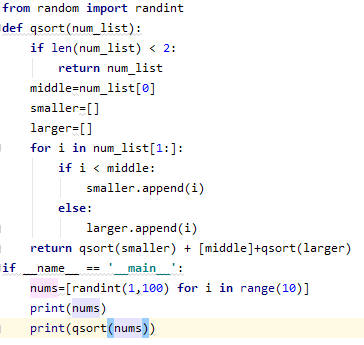
递归函数

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调用







生成器

从句法上讲，生成器是一个带yield语句的函数

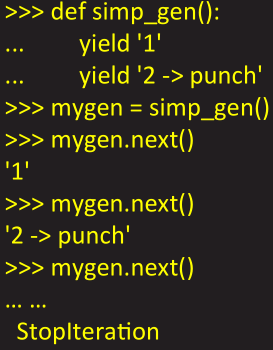
一个函数或者子程序只返回依次，但一个生成器能暂停执行并返回一个中间的结果

yield语句返回一个值给调用者并暂停执行

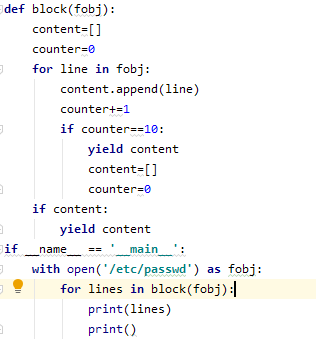
当生成器的next()方法被调用的时候，它会准确的从离开的地方继续

与迭代器相似，生成器以另外的方式来运作

当到达一个真正的返回或者函数结束没有更多的值返回，stopiteration异常就会被抛出







内部函数

闭包（函数调用函数）

闭包将内部函数自己的代码和作用域以及外部函数的作用结合起来

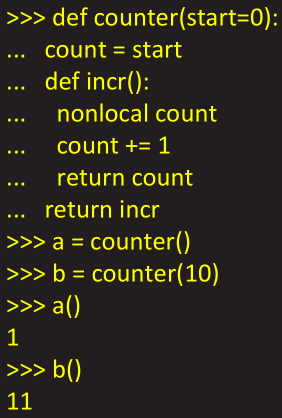
闭包的词法变量不属于全局名字空间域或者局部的名字空间域，而属于其它的名字空间，带着“流浪”的作用域

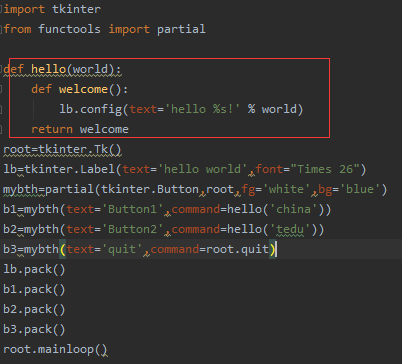
闭包对于安装计算，隐藏状态，以及在函数对象和作用域中随意地切换是很有用的

闭包也是函数，但是他们能携带一些额外的作用域

闭包事例

创建通用的计数器





装饰器

装饰器是在函数调用之上的修饰

这些修饰仅是当申明一个函数或者方法的时候，才会应用的额外调用

使用装饰器的情形有：

引入日志

增加计时逻辑来检测性能

给函数加入事务的能力

