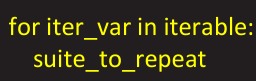
**for循环**

for循环详解

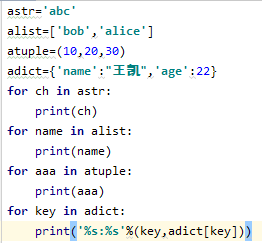
for循环语法结构

python中的for接受可迭代对象（例如序列或迭代器）作为其参数，每次迭代其中一个参数



与while循环一样，支持break、continue、else语句

一般情况下，循环次数未知采用while循环，循环次数已知，采用for循环



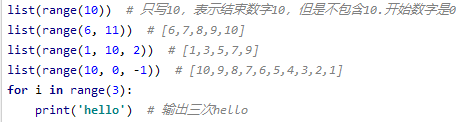
range函数

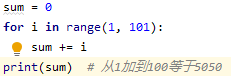
for循环常与range函数一起使用

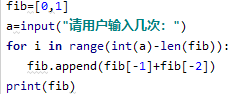
range函数提供循环条件

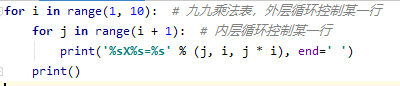
range函数的完整语法为：

rang(start,end,step=1)









列表解析

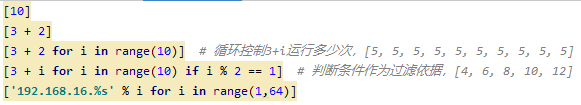
它是一个非常有用、简单、而且灵活的工具，可以动态的创建列表

语法：



这个语句的核心是for循环，它迭代iterable对象的所有条目

expr应用于序列的每个成员，最后的结果值是该表达式产生的列表



**文件对象**

文件打开方法

open及file内建函数

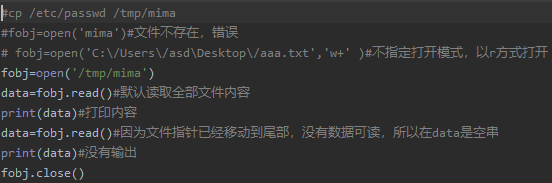
作为打开文件之门的“钥匙”，内建函数open()以及file()提供了初始化输入/输出（I/O）操作的通用接口

成功打开文件时候会返回一个文件对象，否则引发一个错误

open()方法和file()方法可以完全相互替换

基本语法：





文件对象访问方法

|  |  |
| --- | --- |
| 文件模式 | 操作 |
| r | 以读的方式打开（文件不存在则报错） |
| w | 以写方式打开（文件存在则清空，不存在则创建） |
| a | 以追加模式打开（必要时创建新文件） |
| r+ | 以读写模式打开（参见r） |
| w+ | 以读写模式打开（参见w） |
| a+ | 以读写模式打开（参见a） |
| b | 以二进制模式打开 |

文件输入

read方法

read()方法用来直接读取字节到字符串中，最多读取给定数目个字节

如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取至末尾



readline方法

读取打开文件的一行（读取下个行结束符之前的所有字节）

然后整行，包括行结束符，作为字符串返回

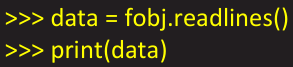
它也有一个可选的size参数，默认为-1，代表读至行结束符

如果提供了该参数，那么在超过size个字节后返回不完整的行



readlines方法

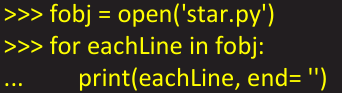
readlines()方法读取所有（剩余的）行然后把它们作为一个字符串列表返回

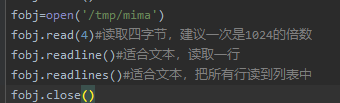


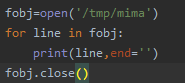
文件迭代

如果需要逐行处理文件，可以结合for循环迭代文件

迭代文件的方法与处理其他序列类型的数据类似







文件输出

write方法

write()内建方法功能与read()和readline()相反。它包含有文本数据或二进制数据块的字符串写入到文件中去

写入文件时，不会自动添加行结束标志，需要程序员手工输入



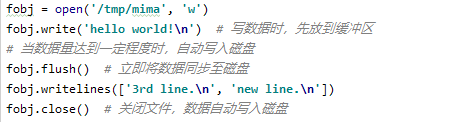
writelines方法

和readlines()一样，writelines()方法主要是针对列表的操作

它接受一个字符串列表作为参数，将它们写入文件

行结束符并不会被自动加入，所以如果需要的话，必须在调用writelines()前每行结尾加上行结束符



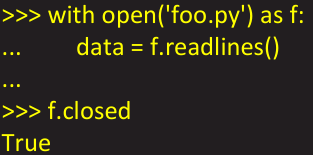


操作文件

with子句

with语句是用来简化代码的

再将打开文件的操作放在with语句中，代码块结束后，文件将自动关闭





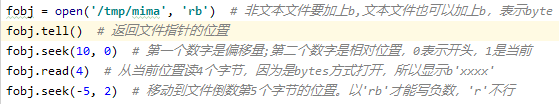
文件内移动

seek(offset[,whence])：移动文件指针到不同的位置

offset是相当于某个位置的偏移量

whence的值，0表示文件开头，1表示当前位置，2表示文件的结尾

tell()：返回当前文件指针的位置



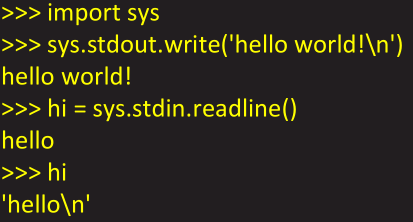
标准文件

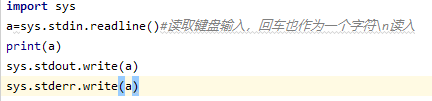
程序一执行，就可以访问三个标准文件

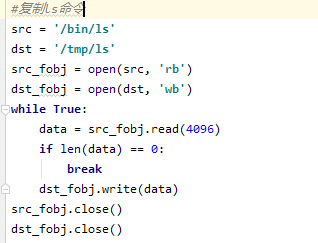
标准输入：一般是键盘，使用sys.stdin

标准输出：一般是显示器缓存输出，使用sys.stdout

标准错误：一般是显示器的非缓存输出，使用sys.stderr







**函数基础**

函数基础操作

函数基本概念

函数是对程序逻辑进行结构化或过程化的一种编程方法

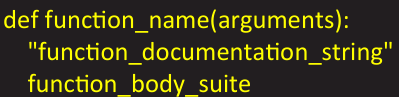
将整个代码巧妙的隔离成易于管理的小块

将重复代码放到函数中而不是进行大量的拷贝，这样既能节省空间，也有助于保持一致性

通常函数都是用于实现某一功能

创建函数

函数是用def语句来创建的，语法如下：



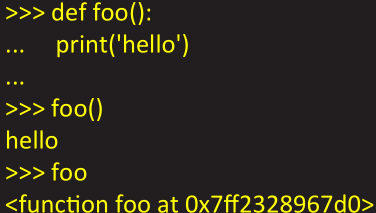
标题行由def关键字，函数的名字，以及参数的集合（如果有的话）组成

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符串，和必须的函数体

调用函数

同大多数语言相同，python用一对圆括号调用函数

如果没有加圆括号，只是对函数的引用

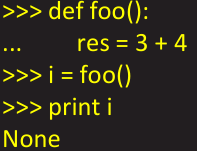


函数的返回值

多数情况下，函数并不直接输出数据，而是向调用者返回值

函数的返回值使用return关键字

没有return的话，函数默认返回none



函数参数

定义参数

形式参数

函数定义时，紧跟在函数名后（圆括号内）的参数被称为形式参数，简称形参。由于它不是实际存在变量，所以又称虚拟变量

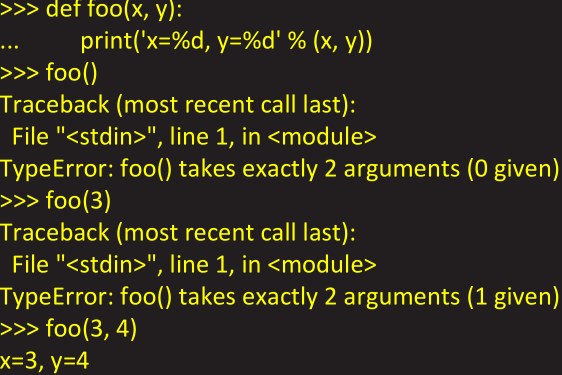
实际参数

在主调函数中调用一个函数时，函数名后面括弧中的参数（可以是一个表达式）称为“实际参数”，简称实参

传递参数

调用函数时，实参的个数需要与形参个数一致

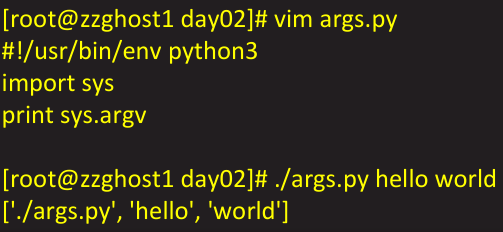
实参将以此传递给形参



位置参数

与shell脚本类似，程序名以及参数都以位置参数的方式传递给python程序

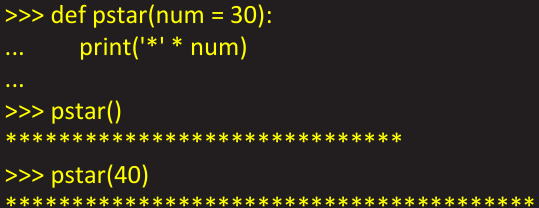
使用sys模块的argv列表接收

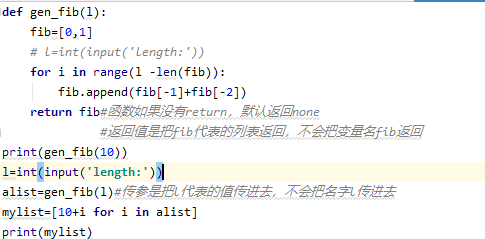


默认参数

默认参数就是声明了默认值的参数

因为给参数赋予了默认值，所以在函数调用时，不向该参数传值也是允许的









**模块基础**

定义模块

模块基本概念

模块从逻辑上组成python代码的形式

当代码量变的相当大的时候，最好把代码分成一些有组织的代码段，前提是保证它们的彼此交互

这些代码片段相互间有一定的联系，可能是一个包含数据成员和方法的类，也可能是一组相关但彼此独立的操作函数

创建模块

模块物理层面上组织模块的方法是文件，每一个以.py结尾的python文件都是一个模块

模块名称切记不要与系统中已存在的模块重名

模块文件名字去掉后面的扩展名（.py）即为模块名

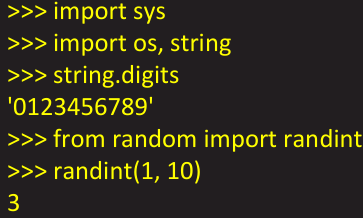
导入模块（import）

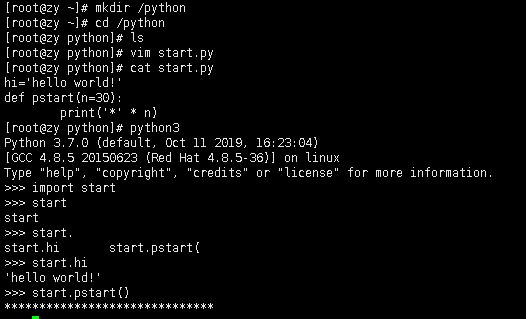
使用import导入模块

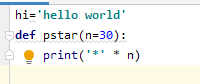
模块被导入后，程序会自动生成pyc的字节码文件以提高性能

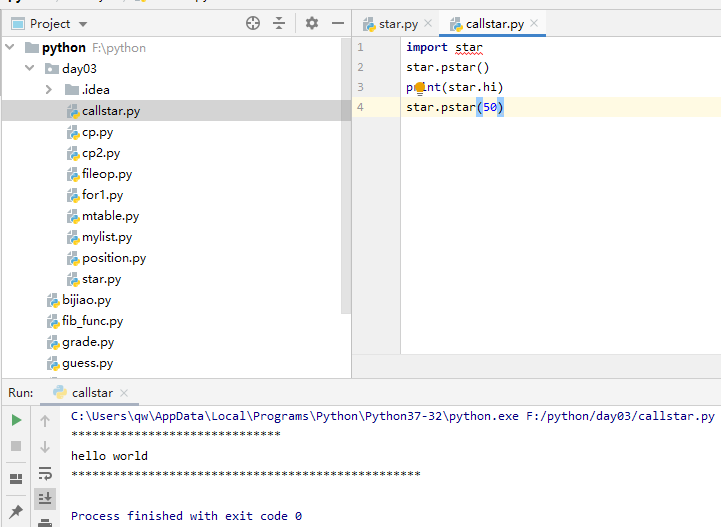
模块属性通过“模块名.属性”的方法调用

如果仅需要模块中的某些属性，也可以单独导入









模块加载（load）

一个模块只允许被加载一次，无论被导入多少次

只加载一次可以组织多重导入时代码被多次执行

如果两个文件互相导入，放置了无线互相加载

模块加载时，顶层代码会自动执行，所以只将函数放入模块的顶层是良好的编程习惯

模块导入的特性

模块具有一个\_\_name\_\_特殊属性

当模块文件直接执行时，\_\_name\_\_的值为’\_\_main\_\_’

当模块被另一个文件导入时，\_\_name\_\_的值就是该模块的名字

