**shell相关模块**

shutil模块

复制和移动

shutil.copyfileobj(fsrc,fdst[,length])

将类似文件的对象fsrc的内容复制到类似文件的对象fdst

shutil.copyfile(fsrc,dst,\*,follow\_symlinks=True)

将名为src的文件的内容（无元数据）复制到名为dst的文件，然后返回dst。

shutil.copy(src,dst,\*,follow\_symlinks=True)

将文件src复制到文件或目录dst。src和dst应为字符串。如果dst指定目录，则文件将使用src的基本文件名复制到dst中。返回新创建的文件的路径。

shutil.copy2(src,dst,\*,follow\_symlinks=True)

与copy()相同，但copy2()也尝试暴力所有文件元数据

shutil.move(src,dst,copy\_function=copy2)

递归地将文件或目录（src）移动到另一个位置（dst），并返回目标。



目录操作

shutil.copytree( src,dst,symlinks=False,ignore=None,copy\_function=copy2,ignore\_dangling\_symlinks =False)

递归地复制以src为根的整个目录树，返回目标目录。由dst命名的目标目录不能已经存在。

shutil.rmtree(path,ignore\_errors=Flase,onerror=None)

删除整个目录树；路径必须指向目录（而不是指向目录的符号链接）

权限管理

shutil.copymode(src,dst,\*,follow\_symlinks=True)

将权限位从src复制到dst。文件内容，所有者和组不受影响。src和dst是以字符串的形式给出路径名称。

shutil.copystat(src,dst,\*,follow)symlinks=True)

将权限位，最后访问时间，上次修改时间和标志从src移动到dst

shutil.chown(path,user=None,group=Nnoe)

更改给定路径的所有用户和/或组



语法风格

变量赋值

python支持链式多重赋值

x=y=10

另一种将多个变量同时赋值的方法称为多元赋值，采用这种方式赋值，等号两边的对象都是元组

a,b=10,20

合法标识符

python标识符字符串规则和其他大部分用C编写的高级语言相似

第一个字符必须是字母或下划线（\_）

剩下的字符可以是字母和数字或下划线

区分大小写

关键字

和其他的高级语言一样，python也拥有一些被称为关键字的保留字符

任何语言的关键字应该保持相对稳定，但是因为python是一门不断成长和进化的语言，其关键字偶尔会更新

关键字列表和iskeyword()函数都放入了keyword模块以便查询

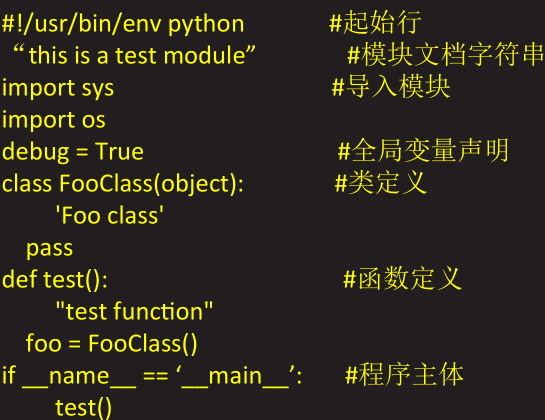
内建

除了关键字之外，python还可以在任何一级代码使用“内建”的名字集合，但是应该把它当做“系统保留字”

保留的常如：True、Flase、None

模块结构及布局

编写程序时，应该建立一种统一且容易阅读的结构，并将它应用到每一个文件中去



**字符串详解**

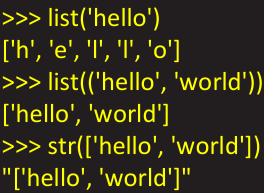
序列

序列类型操作符

|  |  |
| --- | --- |
| 序列操作符 | 作用 |
| seq[ind] | 获取下标为ind的元素 |
| seq[ind1:ind2] | 获取下标从ind1到ind2间的元素集合 |
| seq\*expr | 序列重复expr次 |
| seq1+seq2 | 连接序列seq1和seq2 |
| obj in seq | 判断obj元素是否包含在seq中 |
| obj not in seq | 判断obj元素是否不包含在seq中 |

内建函数

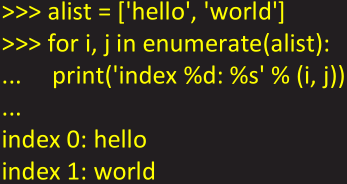
|  |  |
| --- | --- |
| 函数 | 意义 |
| list(iter) | 把可迭代对象转换为列表 |
| str(obj) | 把obj对象转换成字符串 |
| tuple(iter) | 把一个可迭代对象转换成一个元组 |



len(seq):返回seq的长度

max(iter,key=None):返回iter中的最大值

enumerate:接受一个可迭代对象作为参数，返回一个enumerate对象



reversed(seq):接受一个序列作为参数，返回一个以逆序访问的迭代器

sorted(iter):接受一个可迭代对象作为参数，返回一个有序的列表

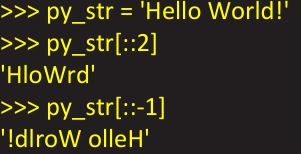
字符串

字符串操作符

比较操作符：字符串大小按ASCII码值进行大小进行比较

切片操作符：[]、[:]、[::]

成员关系操作符：in、not in



格式化操作符

字符串可以使用格式化符号来表示特定含义

|  |  |
| --- | --- |
| 格式化字符 | 转换方式 |
| %c | 转换成字符 |
| %s | 优先用str()函数进行字符串转换 |
| %d/%i | 转换成有符号十进制数 |
| %o | 转换成无符号八进制数 |
| %e/%E | 转换成科学计数法 |
| %f/%F | 转换成浮点数 |

字符串可以使用格式化符号来表示特定含义

|  |  |
| --- | --- |
| 辅助指令 | 作用 |
| \* | 定义宽度或者小数点精度 |
| - | 左对齐 |
| + | 在正数前面显示加号 |
| <sp> | 在正数前面显示空格 |
| # | 在八进制数前面显示零0，在十六进制前面显示0x或者0X |
| 0 | 显示的数字前面填充0而不是默认的空格 |

format函数

使用位置参数

‘my name is {},age {}’.format(‘hoho’,18)

使用关键字参数

‘my name is {name},age is {age}’.format({‘name’:’bob’,’age’:23})

填充与格式化

{:[填充字符][对齐方式 <^>][宽度]}

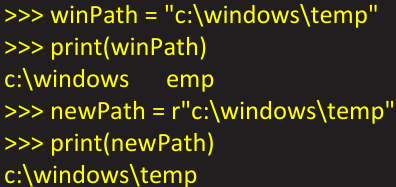
使用索引

'name is {0[0]} age is {0[1]}'.format(['bob', 23])

原始字符串操作符

原始字符串操作符是为了对付那些在字符串中出现的特殊字符

在原始字符串里，所有的字符都是直接按照字面的意思来使用，没有转移特殊或不能打印的字符



内建函数

string.capitalize():把字符串的第一个字符大写

string.center(width):返回一个源字符串居中，并使用空格填充至长度width的新字符串

string.count(str,beg=0,end=len(string)):返回str在string里面出现的次数，如果beg活着end指定则返回指定范围内str出现的次数

string.endswith(obj, beg=0,end=len(string)):检查字符串是否以obj结束，如果beg活着end指定则检查指定范围内是否以obj结束，如果是，则返回true，否则返回false

string.islower():如果string中包含至少一个区分大小写的字符，并且所有这些字符都是小写，则返回true，否则返回false

string.strip():删除string字符串两端的空白

string.upper():转换string中的小写字母为大写

string.split(str=””,num=string.count(str)):以str为分隔符切片string，如果num有指定，则仅分隔num个子字串

**列表和元组**

列表

创建及访问列表

列表是有序、可变的数据类型

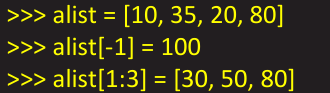
列表中可以包含不同类型的对象

列表可以由[]或工厂函数创建

支持下标及切片操作

更新列表

通过下标只能更新值，不能使用标添加新值



列表内建函数

|  |  |
| --- | --- |
| 列表方法 | 操作 |
| list.append(obj) | 向列表中添加一个对象obj |
| list.count(obj) | 返回一个对象obj在列表中出现的次数 |
| list.extend(seq) | 把序列seq的内容添加到列表中 |
| list.index(obj) | 返回obj对象的下标 |
| list.insert(index.obj) | 在索引为index的位置插入对象obj |
| list.reverse() | 原地反转列表 |
| list.sort() | 排序 |

元组

创建元组

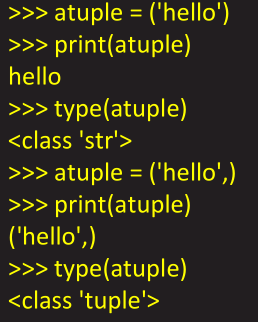
通过()或工厂函数tuple()创建元组

元组是有序的、不可变类型

与列表类似，作用于列表的操作，绝大多数也可以作用于元组

单元素元组

如果一个元组中只有一个元素，那么创建该元组的时候，需要加上一个逗号



“更新”元组

虽然元组本身是不可变的，但是因为它同时属于容器类型，也就意味着元组的某一个元素是可变的容器类型，那么这个元素中的项目仍然可变

