**第一章 快速改造：基础知识**

>>> from \_future\_ import division

让Python执行普通的除法

书本第8 - 9页

python 3.x里面，// 是地板除，/ 不管两边是不是整数得到的都是小数。

四舍五入请用

>>> round(5/3,2)

1.67

input函数

获取控制台的输入

input() 在对待纯数字输入时具有自己的特性，它返回所输入的数字的类型（ int, float ）。

Python3里input()默认收到的是str类型

<http://www.runoob.com/python/python-func-input.html>

书本第12页

pow 函数 把pow等标准函数称之为内建函数

计算乘方

<http://www.runoob.com/python/func-number-pow.html>

abs函数

可以得到数的绝对值

round函数会把浮点数四舍五入到最为接近的整数值

书本第13页

命令import用来导入模块

模块的使用方式：

>>> import math 使用import导入模块math

>>> math.floor(32.9) 按照“模块 . 函数”的格式使用这个模块的函数

32.0

使用form模块import函数这种形式的import命令后，就可以直接使用函数，而不用模块名作为前缀

可以使用变量来引用函数（或者Python中大多数的对象）。比如，通过foo=math.sqrt进行赋值，然后就可以用foo来计算平方根了：foo(4)的结果为2.0。

floor函数

返回数字的下舍整数

int函数

把参数转化为整数，会自动向下取整

书本第14页

>>> import cmath

>>> cmath.sprt(-1)

1j

1j是个虚数，虚数均以j（或者J）结尾

cmath 模块（即complex math, 复数）

为了防止命名冲突应该尽量使用普通的import

书本第15页

\’ \” 转义引号

书本第19 - 20页

str和repr是将Python值转换为合法的Python表达式

书本第21页

raw\_input函数

此函数会把所有的输入当作原始数据（raw data），然后将其放入字符串

除非对input有特别的需要，否则就应该尽可能的使用raw\_input函数

Python3里面没有这个函数

书本第22页

使用三个引号代替普通引号可以跨越多行来写字符串

普通字符串的最后一个如果是\（反斜线），那么，换行符本身就被转义了，会被忽略

书本第22页

/n换行符

r’ ’ 原始字符串

原始字符串以r开头。原始字符串不会把反斜线当做特殊字符。在原始字符串中输入的每个字符都会与书写的方式保持一致。原始字符串的结尾不能是\。

书本第23 - 24页

**第二章 列表和元组**

六种内建的序列：列表、元组、字符串、Unicode字符串、buffer对象和xrange对象

序列

列表

格式：

a = [‘a’, ‘abc’, 1]

b = [‘b’, ‘bcd’, 2]

c = [a, b, ‘a’, 1]

书本第26 - 27页

索引

从0开始递增, 使用负数索引时，Python会从右边开始计数

访问方式：

>>> a = ‘hello’

>>> a[0]

‘h’

>>> a[-1]

‘o’

>>> ‘hello’[1]

E

>>> a = input(‘year: ’)[3]

year: 2005

>> a

‘5’

书本第27 - 28页

分片

以两个索引为边界，第一个包括在分片里面，第二个不包括

格式：

>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> numbers[3:6]

[4, 5, 6]

>>> numbers[-3:-1]

[8, 9]

>>> numbers[1:-3]

[2, 3, 4, 5, 6, 7]

>>> numbers[-3:]

[8, 9, 10]

>>> numbers[:3]

[1, 2, 3]

>>> numbers[:]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

步长

步长不能为0

格式：

>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> numbers[0:3:1]

[1, 2, 3]

>>> numbers[0:3:2]

[1, 3]

>>> numbers[3:0:-1]

[3, 2, 1]

>>> numbers[3::-1]

[3, 2, 1]

>>> numbers[:3:2]

[1, 3]

>>> numbers[::2]

[1, 3, 5, 7, 9]

书本第29 - 31页

空列表

None是一个Python的內键值

格式：

Sequence = [None] \* 10

书本第31 - 32页

in 运算符 （in 操作符）

格式

>>> permissions = ‘rw’

>>> ‘w’ in permissions

True

>>> ‘x’ in permissions

False

>>> users = [‘mlh’, ‘foo’, ‘bar’]

>>> input(‘Enter your user name: ’) in users

Enter your user name: mlh

True

书本第32 - 33页

内建函数len、min和max

Len返回序列中所包含的元素数量

min 返回序列中最小的元素

max 返回序列中最大的元素

格式：

>>> numbers = [100, 34, 678]

>>> len(numbers)

3

>>> min(numbers)

34

>>> max(numbers)

678

>>> max(2,3)

3

>>> min(9, 3, 2, 5)

2

书本第33 - 34页

list函数 list其实是一个类型

将序列转换成列表

格式：

>>> list(‘hello’)

[‘h’, ‘e’, ‘l’, ‘l’, ‘o’]

书本第34页

元素赋值

格式：

>>> x= [1, 1, 1]

>>> x[1] = 2

>>> x

[1, 2, 1]

删除元素

del语句，删除元素后，列表的长度也相应的缩短。出了删除列表中的元素，del还可用于删除其他元素。

格式：

>>> names = [‘Alice’, ‘Beth’, ‘Cecil’, ‘Dee-Dee’, ‘Earl’]

>>> del names[2]

>>> names

[‘Alice’, ‘Beth’, ‘Dee-Dee’, ‘Earl’]

分片赋值

格式：

>>> name=list("Prel")  
>>> name  
['P', 'r', 'e', 'l']  
>>> name[2:]=list("ar")  
>>> name  
['P', 'r', 'a', 'r']

使用分片赋值的时候，可以使用与源序列不等长的序列将分片替换：  
>>> name=list("perl")  
>>> name  
['p', 'e', 'r', 'l']  
>>> name[1:]=list("ython")  
>>> name  
['p', 'y', 't', 'h', 'o', 'n']

分片赋值语句可以在不需要替换任何原有元素的情况下插入新的元素

>>> numbers=[1,5]  
>>> numbers[1:1]=range(2, 3, 4)  
>>> numbers  
[1, 2, 3, 4, 5]

这里看起来是“替换”了一个空的分片，但是实际上是插入了一个新的序列。因此，通过分片赋值来删除元素也是可行的。

>>> number=[1, 2, 3, 4, 5]

>>> number

[1, 2, 3, 4, 5]

>>> number[1:4]=[]

>>> number

[1,5]

如果说步长为1以外的数字，那么分区内要替换几个数值等号后面就要写几个数值。

>>> numbers = list('1111111111')

>>> numbers[::2] = [‘2’]\*5

>>> print(numbers)

['2', '1', '2', '1', '2', '1', '2', '1', '2', '1']

书本第34 - 36页

列表方法

方法是一个与某些对象有紧密联系的函数

格式：

对象.方法(参数)

*网络资料：*

*Python列表操作的函数和方法*

*列表操作包含以下函数:*

*1、cmp(list1, list2)：比较两个列表的元素*

*2、len(list)：列表元素个数*

*3、max(list)：返回列表元素最大值*

*4、min(list)：返回列表元素最小值*

*5、list(seq)：将元组转换为列表*

*列表操作包含以下方法:*

*1、list.append(obj)：在列表末尾添加新的对象*

*2、list.count(obj)：统计某个元素在列表中出现的次数*

*3、list.extend(seq)：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）*

*4、list.index(obj)：从列表中找出某个值第一个匹配项的索引位置*

*5、list.insert(index, obj)：将对象插入列表*

*6、list.pop(obj=list[-1])：移除列表中的一个元素（默认最后一个元素），并且返回该元素的值*

*7、list.remove(obj)：移除列表中某个值的第一个匹配项*

*8、list.reverse()：反向列表中元素*

*9、list.sort([func])：对原列表进行排序*

1. append方法

用于在列表末尾追加新的对象

>>>1st = [1, 2, 3]

>>>1st.append(4)

>>>1st

[1, 2, 3, 4]

只能一次添加一个元素

1. count方法

统计某个元素在列表中出现的次数：

>>> [‘to’, ‘be’, ‘or’, ‘not’, ‘to’, ‘be’].count(‘to’)

2

>>> x = [[1, 2], 1, 1, [2, 1, [1, 2]]]

>>> x.count(1)

2

>>> x.count([1, 2])

1

1. extend 方法

在列表的末尾追加另一个序列的中的多个值。换句话说，可以用新的列表拓展原有列表。

>>> a = [1, 2, 3]

>>> b = [4, 5, 6]

>>> a.extend(b)

>>> a

[1, 2, 3, 4, 5, 6]

extend方法修改了被扩展的序列，而原始的连接操作（+）则不然，它会返回一个全新的列表

可以使用分片赋值来实现相同的效果

>>> a = [1, 2, 3]

>>> b = [4, 5, 6]

>>> a[len(a):] = b

>>> a

[1, 2, 3, 4, 5, 6]

虽然这么做是可行的，但是代码的可读性没有使用extend好。

1. Index方法

从列表中找出某个值第一个（注意是第一个）匹配项的索引位置

>>> a=['one', 'two']

>>> a.index('one')

0

>>> a.index(‘three’)

File "<stdin>", line 1

a.index(‘three’)

^

SyntaxError: invalid character in identifier

当搜索three时，就会发生一个异常，因为这个单词没有被找到。

1. insert方法

将对象插入到列表中。

>>> a=[1, 2]

>>> a.insert(1, 'insert')

>>> a

[1, 'insert', 2]   
可以使用分片赋值来实现相同的效果

>>> a=[1, 2]

>>> a[1:1] = [‘insert’]

>>> a

[1, 'insert', 2]

虽然这么做是可行的，但是代码的可读性没有使用insert好。

1. pop 方法

移除列表中的一个元素（默认是最后一个）。是列表操作中唯一一个，既能修改列表，又能返回元素值的列表方法。

>>> x=[1, 2, 3]

>>> x.pop()

3

>>> x

[1, 2]

>>> x.pop(0)

1

>>> x

[2]

使用pop方法可以实现一个常见的数据结构——栈。可以使用append方法来入栈，使用pop方法来出栈。

1. remove方法

移除列表中某个值的第一个匹配项。

>>> x=[1, 2, 3]

>>> x.remove(2)

>>> x

[1, 3]

当搜索一个不存在的值时，就会发生一个异常，因为这个单词没有被找到。

1. reverse方法

将列表中的元素反向存放。

>>> x=[1, 2, 3]

>>> x.reverse()

>>> x

[3, 2, 1]

1. sort方法

在改变原有列表的情况下，对列表进行排序。

>>> x = [4, 6, 2, 1, 7, 9]

>>> x.sort()

>>> x

[1, 2, 4, 6, 7, 9]

当用户需要一个排列好序的列表副本，又要保留原有列表不变的时候，需要先把x的副本赋值给y，然后在对y进行排序。

>>> x = [4, 6, 2, 1, 7, 9]

>>> y = x[:]

>>> y.sort()

>>> x

[4, 6, 2, 1, 7, 9]

>>> y

[1, 2, 4, 6, 7, 9]

调用x[:]得到的是包含了x所有元素的分片，这是一种很有效率的赋值整个列表的方法。只是简单的把x赋值给y是没有用的，因为这样就让x和y都指向同一个列表了

>>> y = x

>>> y.sort()

>>> x

[1, 2, 4, 6, 7, 9]

>>> y

[1, 2, 4, 6, 7, 9]

还有一种获取已排序的列表副本的方法是，使用sorted函数：

>>> x = [4, 6, 2, 1, 7, 9]

>>> y = sorted(x)

>>> x

[4, 6, 2, 1, 7, 9]

>>> y

[1, 2, 4, 6, 7, 9]

sorted函数可以用于任何序列，却总是返回一个列表。

1. 高级排序

sort 方法可以改变排序方式

内建函数cmp可以提供给sort方法作为参数

>>> cmp(42, 32)

1

>>> cmp(99, 100)

-1

>>> cmp(10, 10)

0

>>> numbers = [5, 2, 9, 7]

>>> numbers.sort(cmp)

>>> numbers

[2, 5, 7, 9]

sort方法有另外两个可选参数——key和reverse。如果要使用它，就要通过名字来指定（这叫关键字参数）参数key与参数cmp类似--必须提供一个在排序过程中使用的函数。然而，该函数并不是直接用来确定对象的大小，而是为每个元素创建一个键，然后所有元素根据键来排序。因此，如果要根据元素的长度进行排序，那么可以使用len作为键函数：

>>> x = ['aardvark', 'abalone', 'acme', 'add', 'aerate']

>>> x.sort(key=len)

>>> x

['add', 'acme', 'aerate', 'abalone', 'aardvark']

另一个关键字参数reverse是简单的布尔值（True或者是False），用来指明列表是否要进行反向排序。

>>> x = [4, 6, 2, 1, 7, 9]

>>> x.sort(reverse=True)

>>> x

[9, 7, 6, 4, 2, 1]

cmp、key、reverse参数都可用于sorted函数。在多数情况下，为cmp或key提供自定义函数是非常有用的。

书本第36 - 41页

元组

元组和列表一样，也是一种序列。区别就在于元组不能修改（字符串也是如此）。创建元组的方法很简单：只要用逗号分隔一些值，就自动创建了元组。

>>> 1, 2, 3

(1, 2, 3)

元组也是（大部分时候是）通过圆括号括起来的

>>> (1, 2, 3)

(1, 2, 3)

空元组可以用没有包含内容的两个圆括号来表示

>>> ()

()

创建包含一个值的元组。必须加个逗号，即使只有有一个值：

>>> 42

42

>>> 42,

(42,)

>>> (42,)

(42,)

最后两个生成了一个长度为1的元组，而第一个例子根本不是元组。

>>> 3\*(40+2)

126

>>> 3\*(40+2,)

(42, 42, 42)

书本第41 - 42页

tuple函数 其实是一种类型

以一个序列为参数并把它转换为元组。如果原来就是元组，就原样返回。

书本第42页