**Python基础I**

金融科技协会 2020年10月16日

目录

[一、基础知识 1](#_Toc54077368)

[1.1 解释器和编译器的区别 1](#_Toc54077369)

[1.2 算法的定义 1](#_Toc54077370)

[1.3 数 1](#_Toc54077371)

[1.4 变量 2](#_Toc54077372)

[1.5 表达式和语句 2](#_Toc54077373)

[1.5.1 表达式：值、变量和运算符的组合 3](#_Toc54077374)

[1.5.2 语句：执行说明 3](#_Toc54077375)

[1.6 获取用户输入 3](#_Toc54077376)

[1.7 内置函数及其调用 3](#_Toc54077377)

[1.8 模块的导入及其简单使用 4](#_Toc54077378)

[1.9 保存与执行程序 5](#_Toc54077379)

[1.10 注释 5](#_Toc54077380)

[1.10.1 单行注释与多行注释 5](#_Toc54077381)

[1.10.2 Spyder 与 Jupyter Notebook 中的使用方法 6](#_Toc54077382)

[二、字符串 6](#_Toc54077383)

[2.1 单引号、双引号、三引号 6](#_Toc54077384)

[2.1.1 单引号和双引号 6](#_Toc54077385)

[2.2.2 三引号 7](#_Toc54077386)

[2.2 转义字符 8](#_Toc54077387)

[2.3 字符串拼接(+) 8](#_Toc54077388)

[2.4 str()与 repr() 8](#_Toc54077389)

[2.5 用r快速转义(路径变量) 9](#_Toc54077390)

[2.6 encode()与 decode() 9](#_Toc54077391)

[2.7 chr()与ord() 10](#_Toc54077392)

[2.8 字符串基本操作 10](#_Toc54077393)

[2.8.1 索引 10](#_Toc54077394)

[2.8.2 切片 11](#_Toc54077395)

[2.8.3 乘法 12](#_Toc54077396)

[2.8.4 成员资格检查 13](#_Toc54077397)

[2.8.5 长度 13](#_Toc54077398)

[2.8.6 最大值和最小值 13](#_Toc54077399)

[2.9 format格式化函数 14](#_Toc54077400)

[2.9.1 替换字段名 14](#_Toc54077401)

[2.9.2 基本转换 15](#_Toc54077402)

[2.9.3 宽度、精度和千位分隔符 16](#_Toc54077403)

[2.9.4 符号、对齐和用0填充 17](#_Toc54077404)

[2.10 字符串常用函数 18](#_Toc54077405)

[2.10.1 center 18](#_Toc54077406)

[2.10.2 find 18](#_Toc54077407)

[2.10.3 join 18](#_Toc54077408)

[2.10.4 split 19](#_Toc54077409)

[2.10.5 upper 19](#_Toc54077410)

[2.10.6 lower 19](#_Toc54077411)

[2.10.7 title 20](#_Toc54077412)

[2.10.8 capitalize 20](#_Toc54077413)

[2.10.9 isupper 20](#_Toc54077414)

[2.10.10 islower 21](#_Toc54077415)

[2.10.11 istitle 21](#_Toc54077416)

[2.10.12 isspace 22](#_Toc54077417)

[2.10.13 isdigit 22](#_Toc54077418)

[2.10.14 isdecimal 22](#_Toc54077419)

[2.10.15 isnumeric 23](#_Toc54077420)

[2.10.16 isprintable 23](#_Toc54077421)

[2.10.17 replace 24](#_Toc54077422)

[2.10.18 translate 24](#_Toc54077423)

[2.10.19 strip 24](#_Toc54077424)

[2.10.20 lstrip 24](#_Toc54077425)

[2.10.21 rstrip 25](#_Toc54077426)

[三、列表与元组 25](#_Toc54077427)

[3.1 序列的定义与分类 25](#_Toc54077428)

[3.2 序列常用操作 25](#_Toc54077429)

[3.2.1 索引 25](#_Toc54077430)

[3.2.2 切片 26](#_Toc54077431)

[3.2.3 相加与乘法 26](#_Toc54077432)

[3.2.4 成员资格检查 26](#_Toc54077433)

[3.3 列表 27](#_Toc54077434)

[3.3.1 list() 27](#_Toc54077435)

[3.3.2 增、删、查 27](#_Toc54077436)

[3.4 元组：不可修改的序列 30](#_Toc54077437)

[四、字典 31](#_Toc54077438)

[4.1 字典的定义与结构特征 31](#_Toc54077439)

[4.2 空字典的创建 31](#_Toc54077440)

[4.3 dict() 31](#_Toc54077441)

[4.4 作为key的要求 31](#_Toc54077442)

[4.5 字典的基本操作 31](#_Toc54077443)

[4.5.1 len() 31](#_Toc54077444)

[4.5.2 增（dict[key]=value update） 32](#_Toc54077445)

[4.6 format\_map() 33](#_Toc54077446)

[4.7 复制（copy() deepcopy） 33](#_Toc54077447)

[4.8 keys() values() items() 34](#_Toc54077448)

[4.9 setdefault() 35](#_Toc54077449)

[五、流程控制与其他Trick 35](#_Toc54077450)

[5.1 再谈print与import 35](#_Toc54077451)

[5.1.1 打印多个参数 35](#_Toc54077452)

[5.1.2 导入时重命名 36](#_Toc54077453)

[5.2 赋值魔法 36](#_Toc54077454)

[5.2.1 序列解包 36](#_Toc54077455)

[5.2.2 链式赋值 38](#_Toc54077456)

[5.2.3 增强赋值 38](#_Toc54077457)

[5.3 代码块与缩进 39](#_Toc54077458)

[5.4 条件与条件语句 39](#_Toc54077459)

[5.4.1 if语句中的真与假 39](#_Toc54077460)

[5.4.2 单独的if语句 40](#_Toc54077461)

[5.4.3 if-else语句 40](#_Toc54077462)

[5.4.4 if-elif-else语句 41](#_Toc54077463)

[5.4.5 嵌套的if语句 41](#_Toc54077464)

[5.5 比较运算符 42](#_Toc54077465)

[5.6 布尔运算符 43](#_Toc54077466)

[5.7 断言 44](#_Toc54077467)

[5.8 循环 45](#_Toc54077468)

[5.8.1 while循环 45](#_Toc54077469)

[5.8.2 for循环 45](#_Toc54077470)

[5.9 列表与字典的推导 50](#_Toc54077471)

[5.9.1 列表推导式 50](#_Toc54077472)

[5.9.2 字典推导式 51](#_Toc54077473)

[5.10 占位符 51](#_Toc54077474)

[5.11 del 51](#_Toc54077475)

[5.12 exec与eval 52](#_Toc54077476)

[5.12.1 exec 52](#_Toc54077477)

[5.12.2 eval 53](#_Toc54077478)

# 

# 一、基础知识

## 1.1 解释器和编译器的区别

解释器：直接执行用编程语言编写的指令的程序 （eg.python, javascript）

编译器：把源代码转换成（翻译）低级语言的程序（eg. C语言代码被编译成二进制代码，在windows平台上执行）

## 1.2 算法的定义

算法：面向计算机，一系列解决问题的清晰指令。

菜谱和算法都由原料(对象)和操作说明(语句)组成。

鸡蛋火腿肠的菜谱： 先取一些火腿肠； 再加些鸡蛋； 如果喜欢吃辣，加些辣味火腿肠； 煮熟为止。记得每隔10分钟检查一次。

## 1.3 数

交互式的Python解释器可用作功能强大的计算器：

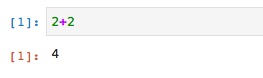


图1 加法



图2 除法

除法运算的结果为小数，即浮点数(float或floating-point number)

如果想丢弃小数部分，即执行整除运算，可使用双斜杠



图3 整除



图4 余数



图5 乘方



图6 乘方（无括号）

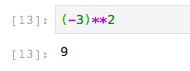


图7 乘方（有括号）

## 1.4 变量

变量：没有固定的值，可以改变的数。



图8 赋值

这称为赋值(assignment)，我们将值3赋给了变量x。

## 1.5 表达式和语句

Python代码由表达式和语句组成，并由Python解释器负责执行。

表达式 是 一些东西，而语句 做 一些事情。

### 1.5.1 表达式：值、变量和运算符的组合



图9 表达式

### 1.5.2 语句：执行说明

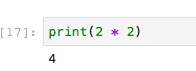


图10 语句

## 1.6 获取用户输入



图11 输入

## 1.7 内置函数及其调用

内置函数：编程语言中预先定义的函数

函数pow：幂运算

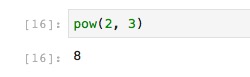


图12 pow

函数abs：计算绝对值

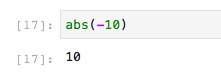


图13 abs

函数round：圆整到最接近的整数



图14 round

## 1.8 模块的导入及其简单使用

模块：可将模块视为扩展，通过将其导入可以扩展Python功能

先使用import导入模块，再以module.function的方式使用模块中的函数

模块math：



图15 math模块

模块cmath：专门用于处理复数（由实部和虚部组成）的模块

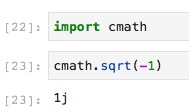


图16 cmath模块

## 1.9 保存与执行程序

（1）Spyder 中保存程序和执行程序的方法

（2）Jupyter Notebook 中保存程序和执行程序的方法

（3）cmd 中执行已有 python 脚本(\*.py)的方法

Windows： 在“开始”菜单中点击“运行”，打开运行窗口（win+R），输入cmd打开命令窗口

Mac：command+space，输入终端

## 1.10 注释

### 1.10.1 单行注释与多行注释

井号(#)在代码中，井号后面到行尾的所有内容都将被忽略



图17 注释

### 1.10.2 Spyder 与 Jupyter Notebook 中的使用方法

command+1/4/5

# 二、字符串

字符串：数字、字母、下划线组成的一串字符

## 2.1 单引号、双引号、三引号

### 2.1.1 单引号和双引号

Python在打印字符串时，在大多数情况下，单引号和双引号是没有区别的



图18 单引号与双引号

特例1：

"Let's go!"

'Let's go!'

这时候就不适合用单引号：解释器将报错，不知道如何处理第二个单引号后余下的内容

特例2：

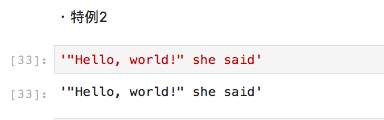


图19 双引号的错误

这时候就不适合用双引号：解释器将报错，不知道如何处理第二个双引号后余下的内容

### 2.2.2 三引号

多行字符串/注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来

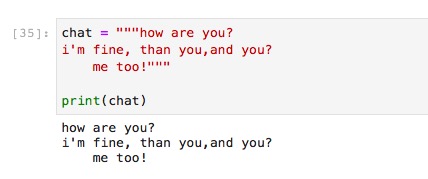
****

图20 三引号

## 2.2 转义字符

使用反斜杠\对引号进行转义(将“\”加在需要转义的引号前)

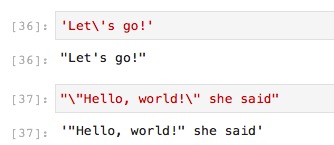


图21 转义字符

## 2.3 字符串拼接(+)



图22 字符串拼接

## 2.4 str()与 repr()

使用str能以合理的方式将值转换为用户能够看懂的字符串



图23 str

使用repr时，通常会获得值的合法Python表达式表示



图24 repr

## 2.5 用r快速转义(路径变量)



图25 转义字符\

对于很长的路径，具有大量的反斜杠？

原始字符串：在字符串前加r，它们根本不会对反斜杠做特殊处理

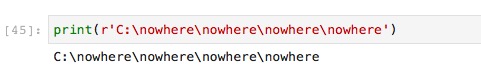


图26 原始字符串

## 2.6 encode()与 decode()



图27 字符编码

## 2.7 chr()与ord()

字符是根据顺序值排列的，要获悉字母的顺序值，可使用函数ord



图28 ord

函数chr与函数ord相反



图29 chr

## 2.8 字符串基本操作

### 2.8.1 索引

索引：序列中的所有元素都有编号——从0开始递增

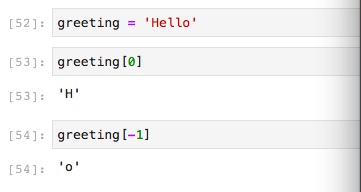
****

图30 索引

### 2.8.2 切片

切片(slicing)：访问特定范围内的元素

numbers[x:y] 第一个索引x是包含的第一个元素的编号，第二个索引y是切片后余下的第一个元素的编号(包括头不包括尾)

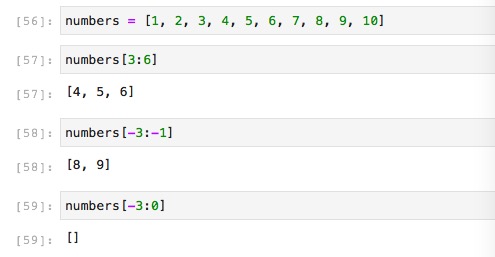
****

图31 切片

如果切片结束于序列末尾，可省略第二个索引

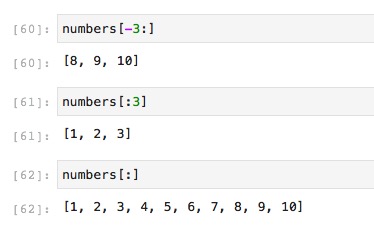
****

图32 切片时省略索引

更大的步长

****

图33 改变步长

### 2.8.3 乘法

将序列与数x相乘时，将重复这个序列x次来创建一个新序列

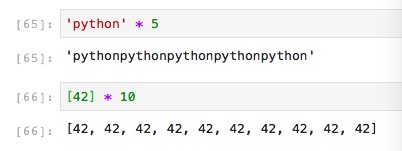
****

图34 序列乘法

### 2.8.4 成员资格检查

要检查特定的值是否包含在序列中，可使用运算符in

****

图35 成员资格检查

### 2.8.5 长度

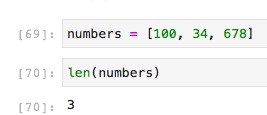
****

图36 序列长度

### 2.8.6 最大值和最小值

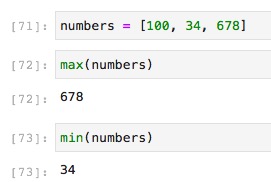
****

图37 序列最大最小值

## 2.9 format格式化函数

格式化字符串的函数：一种向命名替换字段提供值的方式

### 2.9.1 替换字段名

（1）按顺序将字段和参数配对 （2）给参数指定名称，这种参数将被用于相应的替换字段中



图38 format基本用法

不仅使用提供的值本身，而是可访问其组成部分

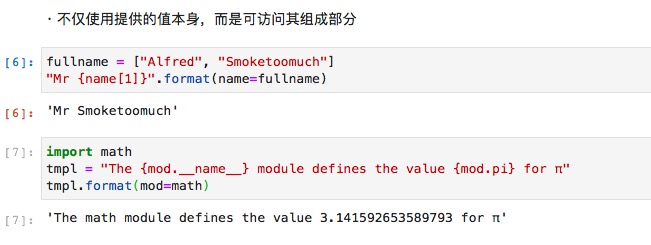


图39 format访问提供的值的组成部分

### 2.9.2 基本转换

三个标志(s、r和a)指定分别使用str、repr和ascii进行转换



图40 format参数之转换字符

字符串格式设置中的类型说明符



图41 format类型说明符

### 2.9.3 宽度、精度和千位分隔符



图42 format参数之宽度

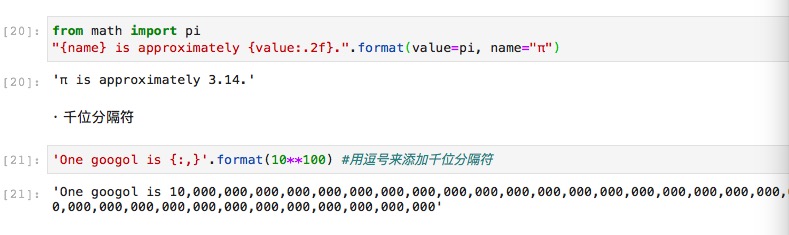


图43 format参数之精度

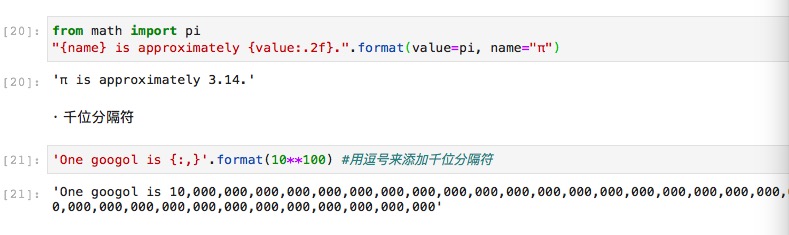


图44 format参数之千位分隔符

### 2.9.4 符号、对齐和用0填充

在指定宽度和精度的数前面，可添加一个标志。这个标志可以是零、加号、减号或空格，其中零表示使用 0来填充数字。

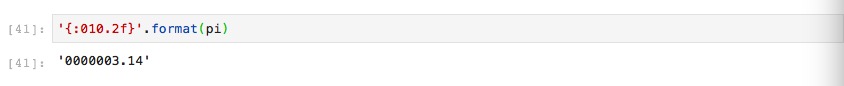


图45 format参数之填充

要指定左对齐、右对齐和居中，可分别使用<、>和^



图46 format参数之对齐

说明符=，它指定将填充字符放在符号和数字之间

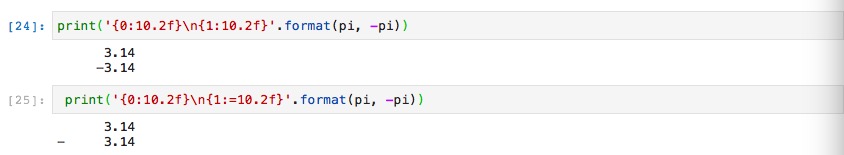


图47 format参数之“=”

给正数加上符号，可使用说明符+，而不是默认的-。如果将符号说明符指定为空格，会在正数前面加上空格而不是+。

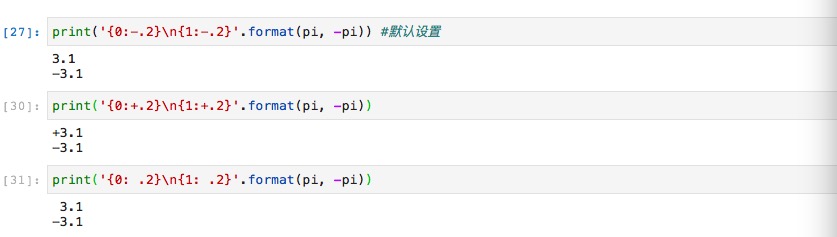


图48 format参数之“+”

#号：可将其放在符号说明符和宽度之间，这个选项将触发另一种转换方式，转换细节随类型而异。



图49 format参数之“#”

## 2.10 字符串常用函数

### 2.10.1 center

通过在两边添加填充字符(默认为空格)让字符串居中

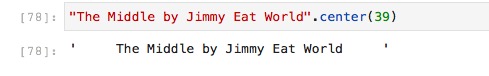


图50 center

### 2.10.2 find

方法find在字符串中查找子串。如果找到，就返回子串的第一个字符的索引，否则返回-1



图51 find

### 2.10.3 join

字符串.join(列表):将字符串插入到join传入列表的字符串元素之间，生成一个新的字符串



图52 join

### 2.10.4 split

split：其作用与join相反，用于将字符串拆分为序列

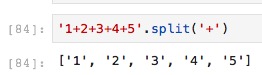


图53 split

### 2.10.5 upper

string.upper()：将字符串中所有的字母都转换为大写，并返回结果

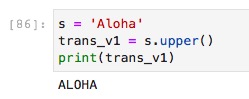


图54 upper

### 2.10.6 lower

string.lower()：将字符串中所有的字母都转换为小些，并返回结果

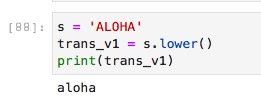


图55 lower

### 2.10.7 title

string.title()：将字符串中所有单词的首字母都大写，并返回结果

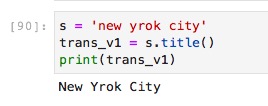


图56 title

### 2.10.8 capitalize

string.capitalize():返回字符串的副本，但将第一个字符大写

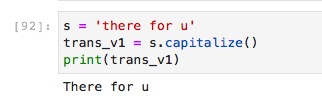


图57 capitalize

### 2.10.9 isupper

检查是否大写

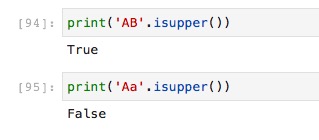


图58 isupper

### 2.10.10 islower

检查是否小写

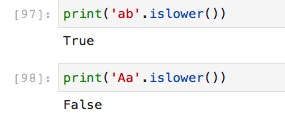


图59 islower

### 2.10.11 istitle

检查是否首字母大写



图60 istitle

### 2.10.12 isspace

判断字符串是否是空白

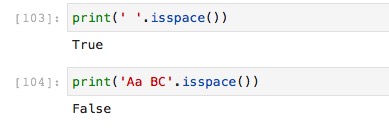


图61 isspace

### 2.10.13 isdigit

检查字符串是否是数字



图62 isdigit

### 2.10.14 isdecimal

检查字符串是否是十进制数



图63 isdecimal

### 2.10.15 isnumeric

检查字符串中的所有字符是否都是数字字符

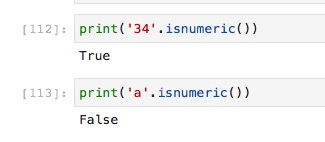


图64 isnumeric

### 2.10.16 isprintable

判断是否是可打印字符（例如制表符、换行符不是可打印字符，空格是可打印字符）

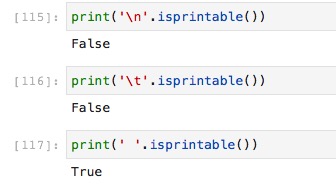


图65 isprintable

### 2.10.17 replace

将指定子串都替换为另一个字符串，并返回替换后的结果



图66 replace

### 2.10.18 translate

能够同时替换多个字符，因此效率比replace高

将一段英语文本转换为带有德国口音的版本，为此必须将字符c和s分别替换为k和z



图67 translate

### 2.10.19 strip

移除左右两边字符

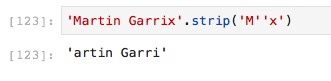


图68 strip

### 2.10.20 lstrip

移除左边字符



图69 lstrip

### 2.10.21 rstrip

移除右边字符

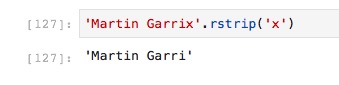


图70 rstrip

# 三、列表与元组

## 3.1 序列的定义与分类

数据结构是以某种方式（如通过编号）组合起来的数据元素（如数、字符乃至其他数据结构）集合。在Python中，最基本的数据结构为序列。序列中的每个元素都有编号，即其位置或索引，其中第一个元素的索引为0，第二个元素的索引为1，依此类推。

Python内置了多种序列，包括列表、元组和字符串。

## 3.2 序列常用操作

### 3.2.1 索引

序列中的所有元素都有编号——从0开始递增。通过使用编号来访问各个元素，这称之为索引。这种索引方式适用于所有序列。当你使用负数索引时，Python将从右（即从最后一个元素）开始往左数，因此-1是最后一个元素的位置。



图71 索引

### 3.2.2 切片

除使用索引来访问单个元素外，还可使用切片来访问特定范围内的元素。为此，可使用两个索引，并用冒号分隔。

切片适用于提取序列的一部分，其中的编号非常重要：第一个索引是包含的第一个元素的编号，但第二个索引是切片后余下的第一个元素的编号。一般格式为List[首位元素：末位元素：步长]，末位数为-1，如果步长为负表述反向索引。如果切片结束于序列末尾，可省略第二个索引。同样，如果切片始于序列开头，可省略第一个索引。

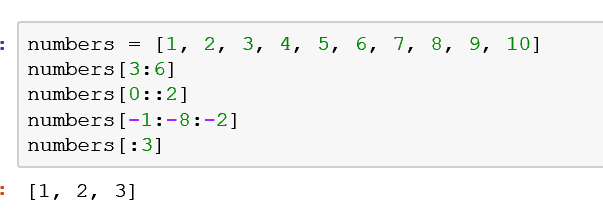


图72 切片

### 3.2.3 相加与乘法

可使用加法运算符来拼接序列。从错误消息可知，不能拼接列表和字符串，虽然它们都是序列。一般而言，不能拼接不同类型的序列。

将序列与数x相乘时，将重复这个序列x次来创建一个新序列。

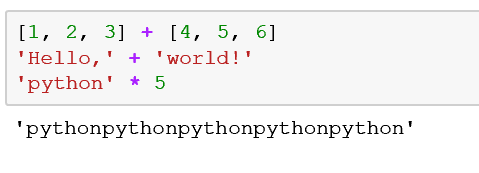


图73 相加与乘法

### 3.2.4 成员资格检查

要检查特定的值是否包含在序列中，可使用运算符in。它检查是否满足指定的条件，并返回相应的值：满足时返回True，不满足时返回False。

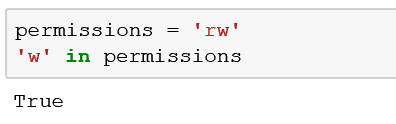


图74 成员资格检查

## 3.3 列表

### 3.3.1 list()

鉴于不能像修改列表那样修改字符串，因此在有些情况下使用字符串来创建列表很有帮助。为此，可使用函数list。请注意，可将任何序列（而不仅仅是字符串）作为list的参数。

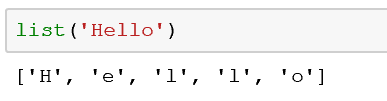


图75 list

### 3.3.2 增、删、查

方法append用于将一个对象附加到列表末尾。

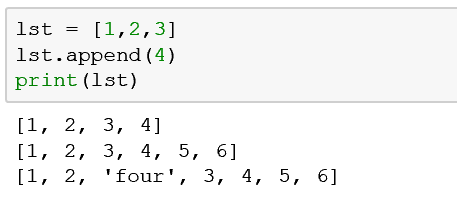


图76 append

方法extend让你能够同时将多个值附加到列表末尾，为此可将这些值组成的序列作为参数提给方法extend。换而言之，你可使用一个列表来扩展另一个列表。

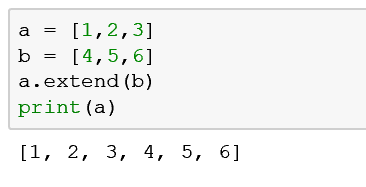


图77 extend

方法insert用于将一个对象插入列表。

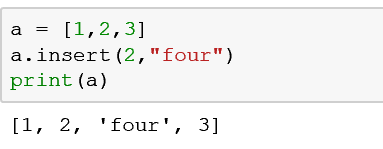


图78 insert

方法pop从列表中删除一个元素（末尾为最后一个元素），并返回这一元素, 但是也可以加位置。

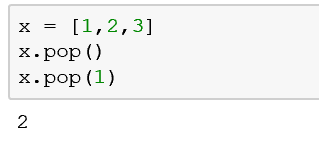


图79 pop

方法remove用于删除第一个为指定值的元素, 如果值不存在，报错。remove是就地修改且不返回值的方法之一。不同于pop的是，它修改列表，但不返回任何值。

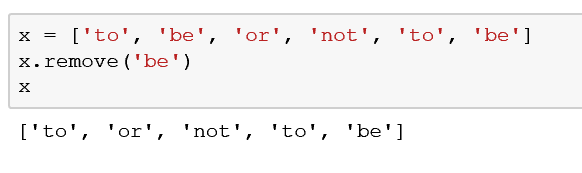


图80 remove

方法del从列表中删除元素。

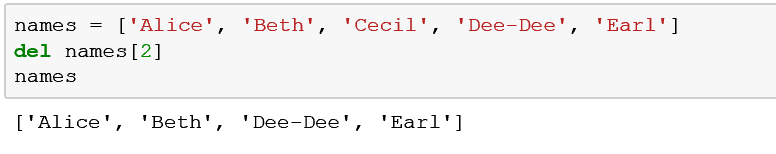


图81 del

方法clear就地清空列表的内容。

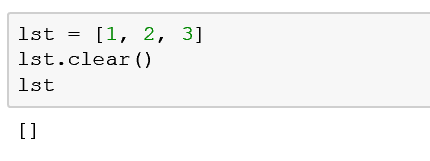


图82 clear

方法count计算指定的元素在列表中出现了多少次。

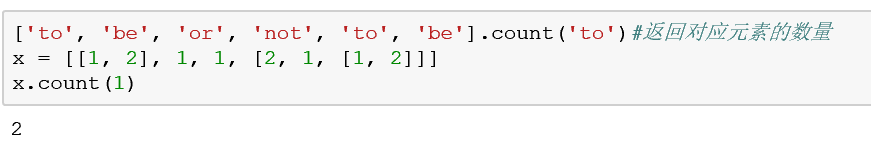


图83 count

方法index在列表中查找指定值第一次出现的索引。



图84 index

方法 copy 复制列表。前面说过，常规复制只是将另一个名称关联到列表。要让a和b指向不同的列表，就必须将b关联到a的副本。

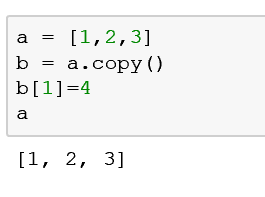


图85 copy

方法reverse按相反的顺序排列列表中的元素。注意到reverse修改列表，但不返回任何值。

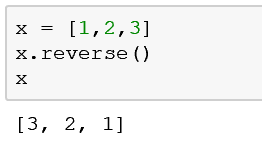


图86 reverse

如果要按相反的顺序迭代序列，可使用函数reversed。这个函数不返回列表，而是返回一个迭代器（迭代器将在第9章详细介绍）。你可使用list将返回的对象转换为列表。

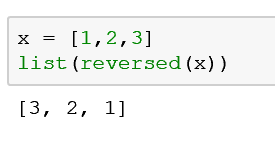


图87 reversed

方法sort用于对列表就地排序。就地排序意味着对原来的列表进行修改，使其元素按顺序排列，而不是返回排序后的列表的副本。

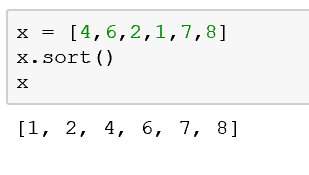


图88 sort

## 3.4 元组：不可修改的序列

与列表一样，元组也是序列，唯一的差别在于元组是不能修改的（你可能注意到了，字符串也不能修改）。元组语法很简单，只要将一些值用逗号分隔，就能自动创建一个元组。

空元组和单元素元组的定义

空元组用两个不包含任何内容的圆括号表示。

单元素元组表示只包含一个值的元组。虽然只有一个值，也必须在它后面加上逗号。

****

图89 单元组

函数tuple的工作原理与list很像：它将一个序列作为参数，并将其转换为元组。如果参数已经是元组，就原封不动地返回它。

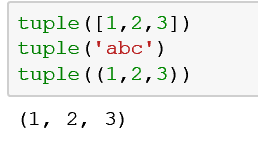
****

图90 tuple

序列解包（或可迭代对象解包）：将一个序列（或任何可迭代对象）解包，并将得到的值存储到一系列变量中。下面用例子进行解释。

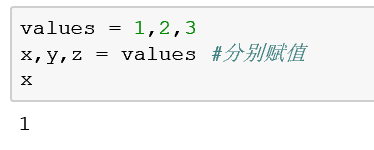


图91 序列解包

# 四、字典

## 4.1 字典的定义与结构特征

字典的名称指出了这种数据结构的用途。字典（日常生活中的字典和Python字典）旨在让你能够轻松地找到特定的单词（键），以获悉其定义（值）。字典由键及其相应的值组成，这种键-值对称为项。在前面的示例中，键为名字，而值为电话号码。每个键与其值之间都用冒号（:）分隔，项之间用逗号分隔，而整个字典放在花括号内。

## 4.2 空字典的创建

空字典（没有任何项）用两个花括号表示，类似于下面这样：{}。

## 4.3 dict()

可使用函数dict()从其他映射（如其他字典）或键-值对序列创建字典。

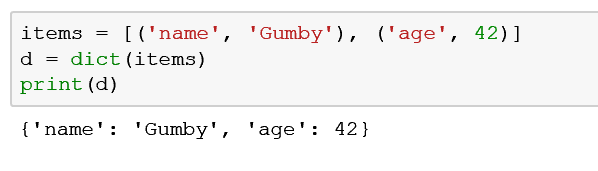
****

图92 dict

## 4.4 作为key的要求

字典中的键可以是整数，但并非必须是整数。字典中的键可以是任何不可变的类型，如浮点数（实数）、字符串或元组。

## 4.5 字典的基本操作

### 4.5.1 len()

方法len()计算字典的长度。



图93 计算字典长度

### 4.5.2 增（dict[key]=value update）

方法dict[key]=value，用于创建一个新的字典，返回一个新的字典。

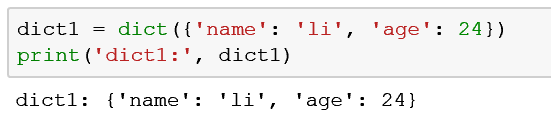


图94 字典添加新值

方法update使用一个字典中的项来更新另一个字典。

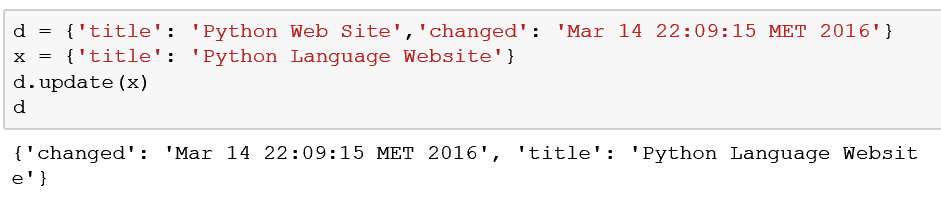


图95 update

4.5.3 删（del clear pop popitem）

方法clear删除所有的字典项，这种操作是就地执行的（就像list.sort一样），因此什么都不返回（或者说返回None）。

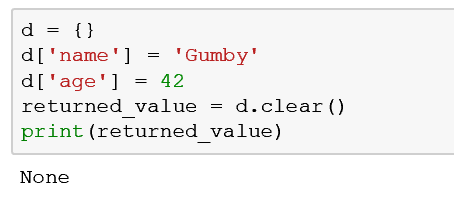


图96 clear

方法pop可用于获取与指定键相关联的值，并将该键-值对从字典中删除。

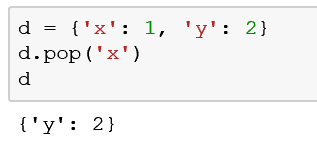


图97 pop

方法popitem类似于list.pop，但list.pop弹出列表中的最后一个元素，而popitem随机地弹出一个字典项，因为字典项的顺序是不确定的，没有“最后一个元素”的概念。如果你要以高效地方式逐个删除并处理所有字典项，这可能很有用，因为这样无需先获取键列表。



图98 popitem

查（成员资格检查 get）

方法get为访问字典项提供了宽松的环境。通常，如果你试图访问字典中没有的项，将引发错误，而使用get不会这样。

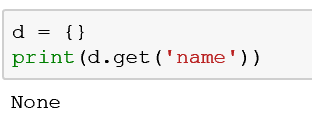


图99 get

## 4.6 format\_map()

可使用字符串格式设置功能来设置值的格式，这些值是作为命名或非命名参数提供给方法format的。我们也可在字典中包含各种信息，这样只需在格式字符串中提取所需的信息即可。为此，必须使用format\_map来指出你将通过一个映射来提供所需的信息。

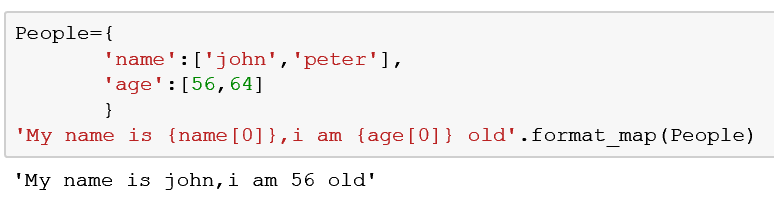
****

图100 format\_map

## 4.7 复制（copy() deepcopy）

方法copy返回一个新字典，其包含的键-值对与原来的字典相同（这个方法执行的是浅复制，因为值本身是原件，而非副本）。

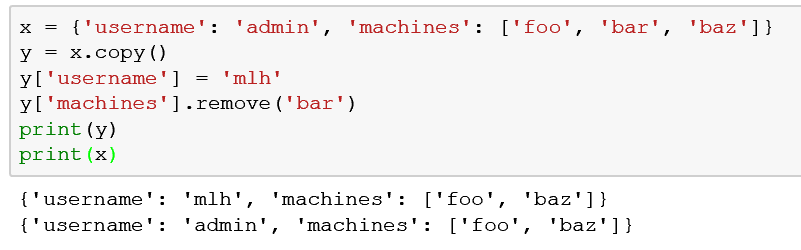
****

图101 copy

当替换副本中的值时，原件不受影响。然而，如果修改副本中的值（就地修改而不是替换），原件也将发生变化，因为原件指向的也是被修改的值（如这个示例中的'machines'列表所示）。避免这种问题，方法深复制即同时复制值及其包含的所有值。可使用模块copy中的函数deepcopy。

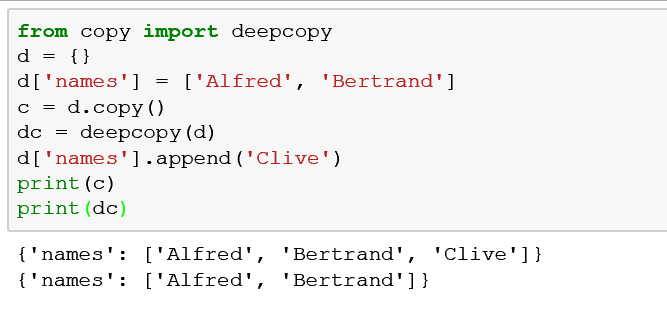
****

图102 deepcopy

## 4.8 keys() values() items()

方法keys返回一个字典视图，其中包含指定字典中的键。

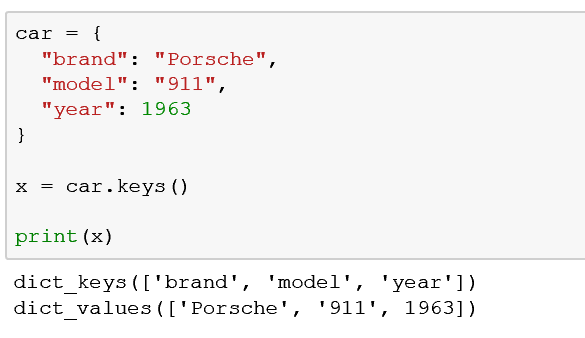
****

图103 keys

方法items返回一个包含所有字典项的列表，其中每个元素都为(key, value)的形式。字典项在列表中的排列顺序不确定。

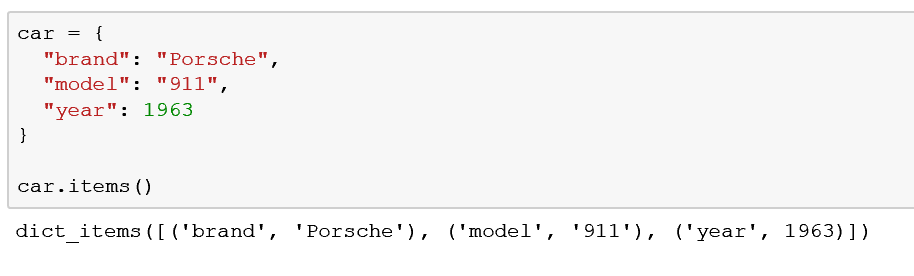
****

图104 items

方法values返回一个由字典中的值组成的字典视图。不同于方法keys，方法values返回的视图可能包含重复的值。

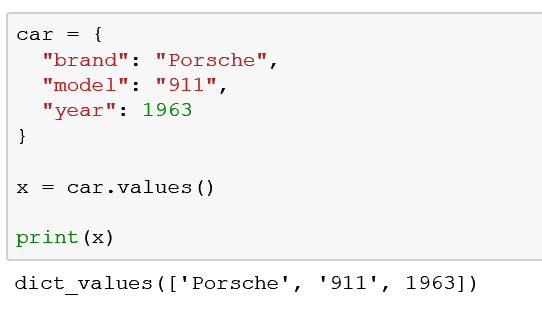
****

图105 values

## 4.9 setdefault()

方法setdefault有点像get，因为它也获取与指定键相关联的值，但除此之外， setdefault还在字典不包含指定的键时，在字典中添加指定的键-值对。

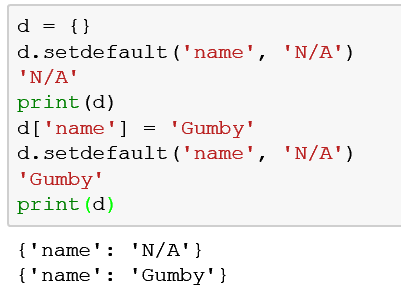


图106 setdefault

# 五、流程控制与其他Trick

* 1. 再谈print与import

### 5.1.1 打印多个参数

print实际上可同时打印多个表达式，条件是用逗号分隔它们，这时print会自动在参数之间插入了一个空格字符。

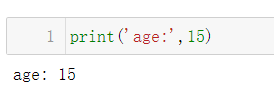
****

图107 print打印多个对象

如果我们不想包含空格怎么办呢？print还可以自定义分隔符。

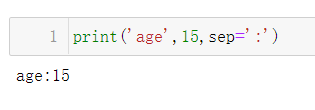
****

图108 print自定义分隔符

### 5.1.2 导入时重命名

我们可以用import导入一个模块或者使用import from的形式导入一个模块下的某个子模块，也可以用from import \*的形式导入一个模块下的全部子模块。

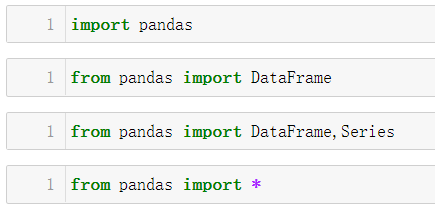
****

图109 import

当有两个模块的子模块同名时，如果仅仅用import导入容易造成混乱，这时我们一般采用import as的形式为模块重命名，以避免命名空间的混乱。



图110 导入时重命名

## 5.2 赋值魔法

### 5.2.1 序列解包

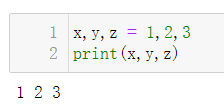
****

图111 多变量赋值

在python中我们可以给多个变量一起赋值。

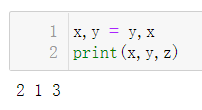
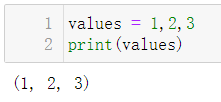
****

图112 交换赋值

使用这种方式还可以交换两个变量的值。

****

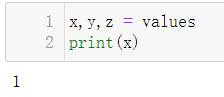
****

图113 序列解包

实际上，这里执行的操作称为序列解包（或可迭代对象解包）：将一个序列（或任何可迭代对象）解包，并将得到的值存储到一系列变量中。以上的赋值就是在解包values这个元组。

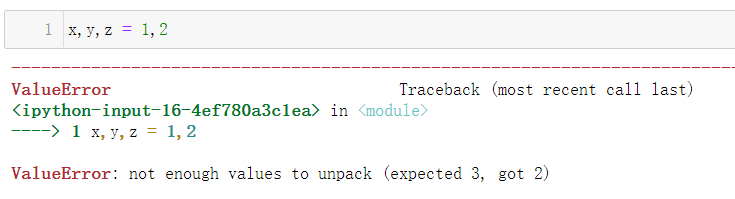
****

图114 序列解包时变量个数不同

注意序列解包时变量的个数必须要与序列中值的个数相等，否则会造成错误。

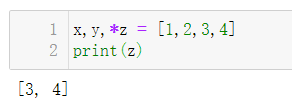
****

图115 使用\*吸收多余的值

为了避免出现变量个数不匹配的问题，我们可以在最后加一个\*，它会将多余的值都包含在这个星号中。

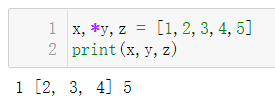
****

图116 \*放在中间

\*也可以放在中间，这时它会收集除第一个和最后一个之外的值。

### 5.2.2 链式赋值

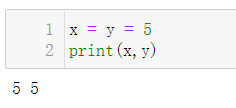
****

图117 链式赋值

链式赋值是一种快捷方式，用于将多个变量关联到同一个值。

### 5.2.3 增强赋值

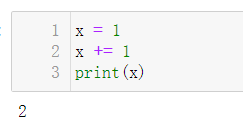
****

图118 增强赋值

可以不编写代码x = x + 1，而将右边表达式中的运算符（这里是+）移到赋值运算符（ =）的前面，从而写成x += 1。这称为增强赋值，适用于所有标准运算符，如\*、 /、 %等。

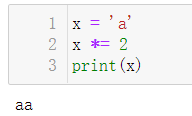
****

图119 序列乘法

增强赋值也可以用于其它数据类型，使用增强赋值会让代码看起来更简洁。

## 5.3 代码块与缩进

代码块是一组语句，可在满足条件时执行（ if语句），可执行多次（循环），等等。代码块是通过缩进代码（即在前面加空格）来创建的。

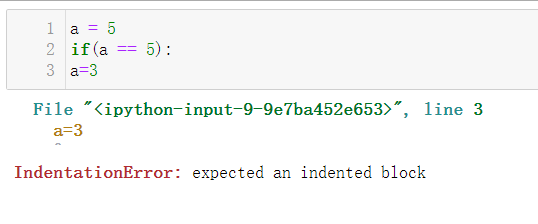


图120 错误缩进

这是一个没有缩进的例子，会报错”expected an indented block”。

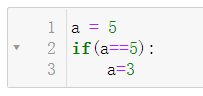


图121 正确缩进

这是一个正确缩进的例子。

## 5.4 条件与条件语句

### 5.4.1 if语句中的真与假

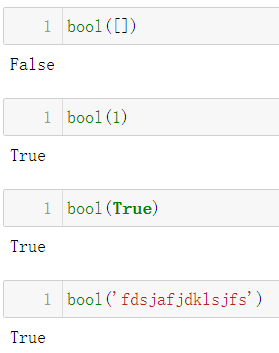
****

图122 布尔值

在if语句中，标准值False和None、各种类型（包括浮点数、复数等）的数值0、空序列（如空字符串、空元组和空列表）以及空映射（如空字典）都被视为假，而其他各种值都被视为真，包括特殊值True。

### 5.4.2 单独的if语句

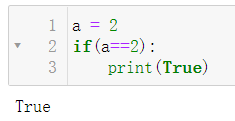
****

图123 单独的if语句

### 5.4.3 if-else语句

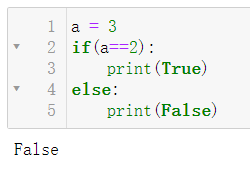
****

图124 if-else

### 5.4.4 if-elif-else语句

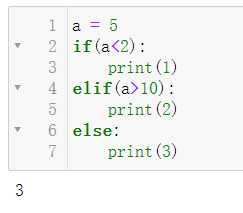
****

图125 if-elif-else

### 5.4.5 嵌套的if语句

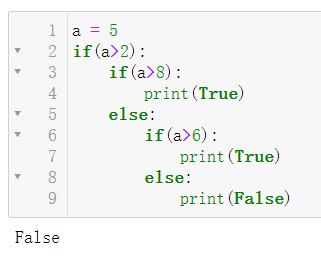
****

图126 嵌套的if语句

## 5.5 比较运算符

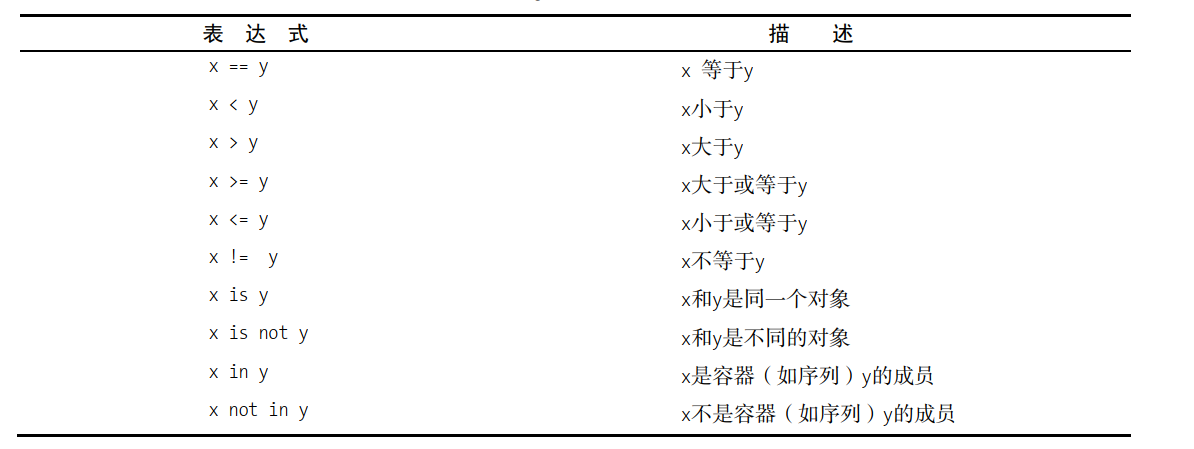


图127 常见的比较运算符

以上是python常见的比较运算符，下面我们简单介绍几个。

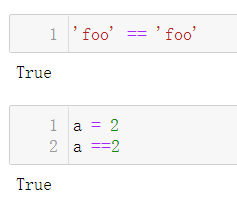


图128 “==”

==是比较两个对象的值是否相等。

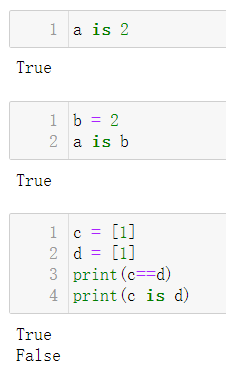


图129 is

is的用法有些复杂，对于list这种对象，尽管它们值相同，但并不是一个对象，所以is的结果是False，但对于单独一个数，分别赋予不同的变量，用is比较一下得到的结果是True。一般在实践中还是要避免使用is。

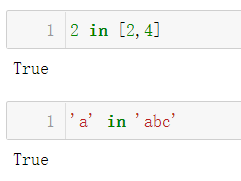


图130 in

in的作用是检查一个对象是否在一个序列中，这个序列可以是list，也可以是字符串。

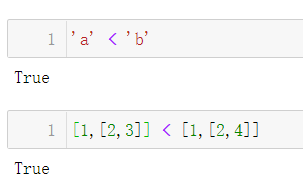


图131 <

大于小于号的用法与常识相符，只不过在python中进一步扩宽了它们的应用对象。

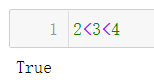


图132 链式比较

Python同样支持链式比较。

## 5.6 布尔运算符

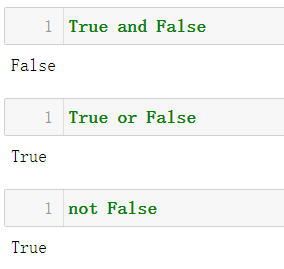


图133 逻辑运算符

在python中布尔运算符和其它语言中的逻辑是一样的，包含“与”“或”“非”三种逻辑运算，分别用and or not来表示。

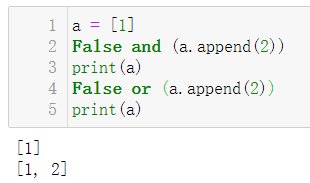


图134 短路逻辑

同样地，短路逻辑在python中一样适用。

## 5.7 断言

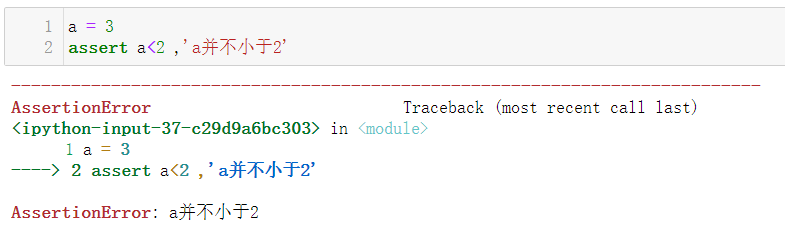


图135 assert断言

当我们需要核实函数参数满足要求或为初始测试和调试提供帮助时，可以考虑使用assert判断，假如条件不满足的话程序会直接崩溃。

## 5.8 循环

### 5.8.1 while循环

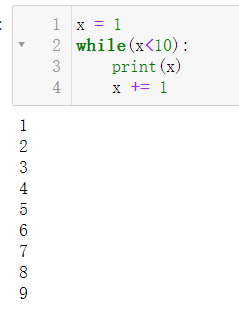
****

图136 while语句

While循环会一直进行，只有在括号中的条件不满足时才会退出。

### 5.8.2 for循环

for是使用频率最高的python关键字之一，其主要用途是用来迭代。

迭代列表与元组

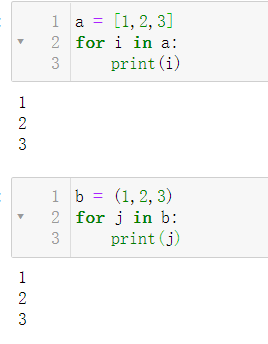
****

图137 for循环迭代列表与元组

使用for循环可以输出列表或者元组中的每一个对象。

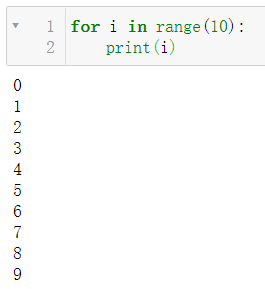
****

图138 range

鉴于我们经常需要迭代0~n之间的数值，python为此发明了range关键字，range(n)即返回一个0~n-1的列表。

迭代字典

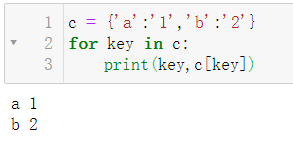
****

图139 迭代字典

直接迭代字典实际上只是在迭代其键值。

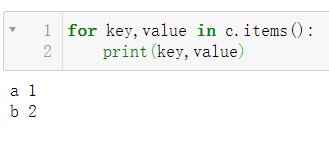
****

图140 字典的items方法

如果想同时迭代key和value，我们可以使用字典的items()方法。

并行迭代

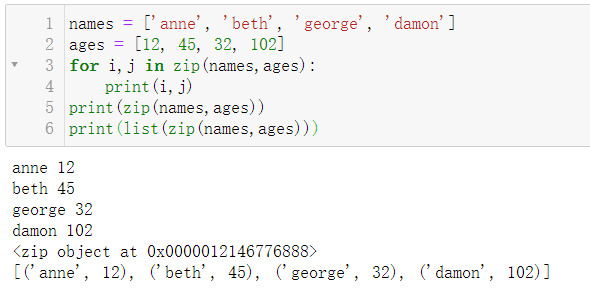
****

图141 zip

使用zip方法我们可以将两个list组合成一个键值对形式的list来进行迭代。Zip生成的是一个可迭代对象，直接观察不到，我们可以将其转化成list来观察。

迭代时获取索引

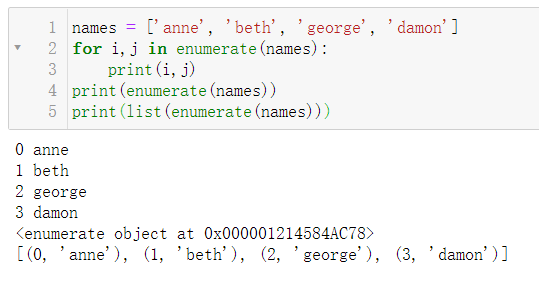
****

图142 enumerate

使用enumerate方法我们将一个列表的索引和值生成一个键值对序列。

反向迭代与排序后迭代

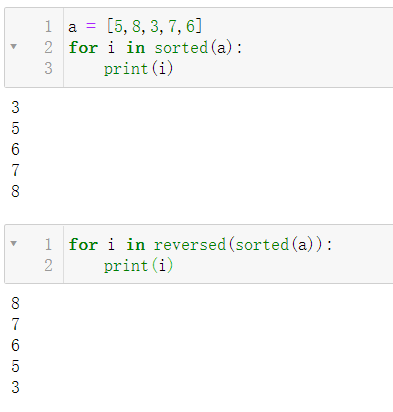
****

图143 反向迭代和排序后迭代

使用sorted方法和reversed方法可以对list进行排序和反向，然后再进行迭代。

break与continue

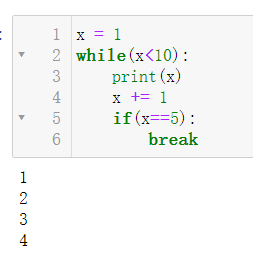
****

图144 break

break可以在满足一定的条件时直接跳出循环。

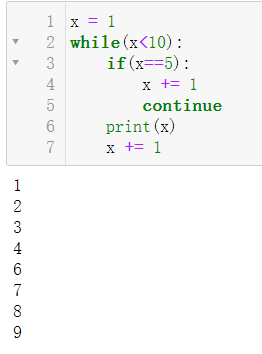
****

图145 continue

continue可以将满足循环的本轮条件跳过，如上面并没有打印5.

for-else语句

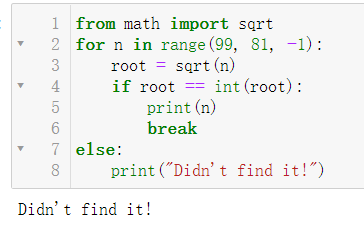


图146 for-else语句

else只有在没有执行break时才会执行。

## 5.9 列表与字典的推导

### 5.9.1 列表推导式

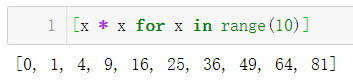
****

图147 简单的列表推导式

我们可以利用这种形式从一个列表创造另一个列表。

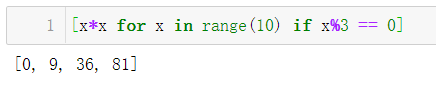
****

图148 带条件的列表推导式

也可以设置一些条件。

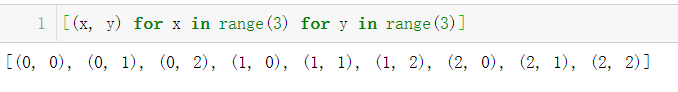
****

图149 并行的列表推导式

也可以从多个列表创造一个列表。

### 5.9.2 字典推导式

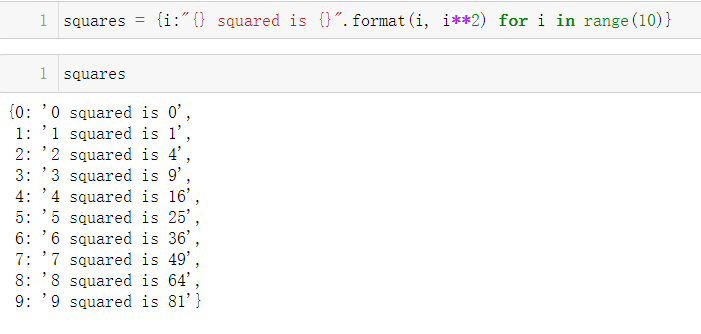
****

图150 字典推导式

在列表推导中， for前面只有一个表达式，而在字典推导中，for前面有两个用冒号分隔的表达式，这两个表达式分别为键及其对应的值。

## 5.10 占位符



图151 pass占位符

是的，运行pass什么都没有发生，它经常作为占位符来使用，等待程序员以后再补充代码。

## 5.11 del

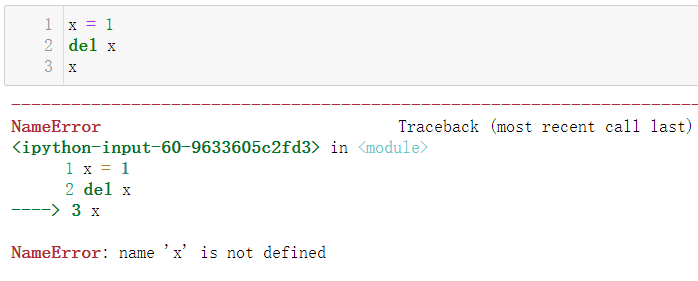


图152 del

del会将变量名删除，但del无法删除值本身，例如下面的例子：

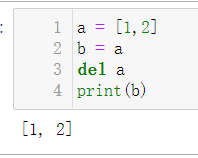


图153 del只能删除变量名

这里仅仅是将a这个变量名删除了，并不会影响b。

## 5.12 exec与eval

### 5.12.1 exec



图154 exec

exec将字符串作为代码执行。

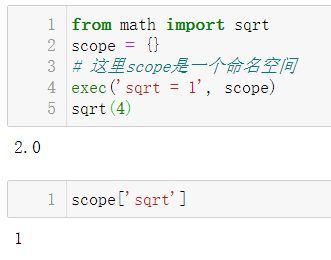
****

图155 exec设置命名空间

有时候exec执行一些赋值命令会造成命名空间混乱，因此在exec可以为其字符串中的变量单独设置一个命名空间。

### 5.12.2 eval

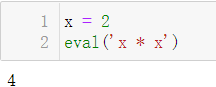


图156 eval

exec是执行语句，而eval仅仅计算表达式的值。