第三章：数据类型与对象

Table of Contents

[1. 序言 3](#_Toc52639483)

[1.1 章节简介 3](#_Toc52639484)

[2. 常量与变量 4](#_Toc52639485)

[2.1 变量是什么 4](#_Toc52639486)

[2.2 常量是什么 5](#_Toc52639487)

[3. 基础类型-数值类型 5](#_Toc52639488)

[3.1 布尔型 5](#_Toc52639489)

[3.2 整型 6](#_Toc52639490)

[3.3 浮点型 6](#_Toc52639491)

[3.4 空值类型 7](#_Toc52639492)

[4. 基础类型-字符串类型 7](#_Toc52639493)

[4.1 下标和切片 8](#_Toc52639494)

[4.1.1 下标索引 8](#_Toc52639495)

[4.1.2 切片 8](#_Toc52639496)

[4.2 字符串的常见操作 8](#_Toc52639497)

[4.2.1 find 9](#_Toc52639498)

[4.2.2 index 9](#_Toc52639499)

[4.2.3 count 9](#_Toc52639500)

[4.2.4 replace 9](#_Toc52639501)

[4.2.5 split 9](#_Toc52639502)

[4.2.6 startwith 10](#_Toc52639503)

[4.2.7 endswith 10](#_Toc52639504)

[4.2.8 upper 10](#_Toc52639505)

[4.2.9 lstrip 10](#_Toc52639506)

[5. 容器类型-列表list 11](#_Toc52639507)

[5.1 列表简介 11](#_Toc52639508)

[5.2 列表的常见操作 11](#_Toc52639509)

[5.2.1 列表的长度 11](#_Toc52639510)

[5.2.2 列表的访问 11](#_Toc52639511)

[5.2.3 列表的切片 12](#_Toc52639512)

[5.2.4 给列表添加元素 12](#_Toc52639513)

[5.2.5 修改元素 13](#_Toc52639514)

[5.2.6 查找元素 13](#_Toc52639515)

[5.2.7 删除元素 14](#_Toc52639516)

[5.2.8 排序 14](#_Toc52639517)

[5.2.9 列表遍历 15](#_Toc52639518)

[6. 容器类型-元组tuple 15](#_Toc52639519)

[7. 容器类型-字典dict 16](#_Toc52639520)

[7.1 字典简介 16](#_Toc52639521)

[7.2 字典的常见操作 17](#_Toc52639522)

[7.2.1 修改元素 17](#_Toc52639523)

[7.2.2 添加元素 17](#_Toc52639524)

[7.2.3 删除元素 18](#_Toc52639525)

[7.2.4 查看字典中有多少个元素 18](#_Toc52639526)

[7.2.5 返回字典中的所有key 19](#_Toc52639527)

[7.2.6 返回字典中的所有value 19](#_Toc52639528)

[7.2.7 返回一个包含所有元素(键值对)的列表 19](#_Toc52639529)

[8. 容器类型-集合(set) 19](#_Toc52639530)

[9. 变量，对象，深拷贝与浅拷贝 21](#_Toc52639531)

[9.1 不可变对象和可变对象 21](#_Toc52639532)

[9.2 深拷贝与浅拷贝 22](#_Toc52639533)

[9.3 小结 25](#_Toc52639534)

# 序言

## 章节简介

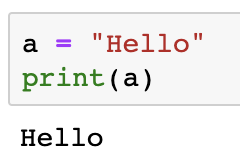
Python程序中保存的所有数据都是围绕对象这个概念来构建的。对象包括一些基本的数据类型，如数字、字符串、列表和字典。同时也可以通过类的形式创建用户定义的对象。本章介绍Python对象模型的内部原理，并介绍主要的内置数据类型。本章主要内容有：

1. 常量和变量
2. 基础类型：数值类型
3. 基础类型：字符串类型
4. 容器类型：列表list
5. 容器类型：元组tuple
6. 容器类型：字典dict
7. 容器类型：集合set
8. 变量类型转换

# 常量与变量

## 2.1 变量是什么

在前面的学习中，我们实际已经使用到了变量。下面的代码中，我们声明了一个变量a并且赋值为字符串”Hello”，在代码的第二行，我们打印出a的值。



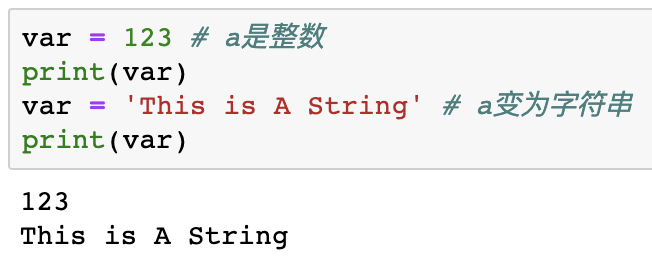
变量的概念基本上和初中代数方程里的变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意类型。我们在程序中用一个变量名来代表一个变量，变量名必须是大小写英文、数字和\_的组合，且不能用数字开头，关于详细的命名规范我们在上一小节有过详细的介绍。下面的代码声明了变量a，它是一个整数：



而下面我们又声明了变量str,它的值是一个字符串：

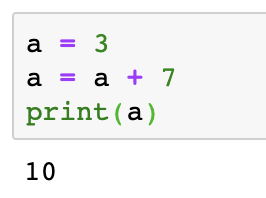


在Python中，等号=是赋值语句，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，例如下面我们声明了一个变量var,它的值开始是一个整数123，代码的第三行我们又把字符串”This is A String”赋值给var，现在它的值变成了一个字符串。



跟其他静态类型的高级语言(比如Java)相比，这种变量类型随时变化的特性，我们又把Python叫做动态语言。

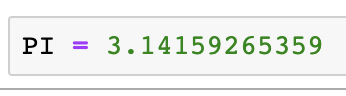
我们再来解释下赋值运算符=，它在Python中表示的意义是将等号右边的值赋值给左边的变量，而不是在数学中我们熟悉的“等于“的意思。在下面的代码中，实际上Python先计算等号右边，将变量a的值加7，再将计算得到的值重新赋值给a：



关于变量的最后，我们再来多了解一下当我们给一个变量赋值的时候，Python都在计算机内存中做了哪些事情：当我们编写一段程序a=888时，Python首先在内存中创建了一个888的数字，然后在内存中创建了一个变量a，并将a指向888。

## 2.2 常量是什么

所谓常量就是不能变的变量，比如常用的数学常数π就是一个常量。在Python中，通常用全部大写的变量名表示常量：



但事实上PI仍然是一个变量，Python根本没有任何机制保证PI不会被改变。所以，用全部大写的变量名表示常量只是一个习惯上的用法，如果你一定要改变变量PI的值，也没人能拦住你。

# 基础类型-数值类型

本小节介绍可用于表示程序中会使用的大多数数据类型。Python使用5中数字类型：布尔型、整数、长整数、浮点数以及负数。我们会介绍布尔型、整数、浮点数以及空值类型。

## 布尔型

布尔值包括True和False两个值，分别映射为数值1和0。一个布尔值只有True、False两种值。在Python中，我们使用True和False表示布尔值，比如像下面：

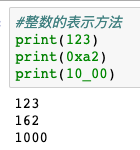


布尔类型的变量可以用来进行与或非的逻辑运算：



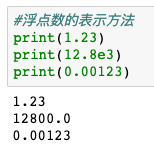
## 整型

Python3中可以处理任意大小的整数，在程序里表示一个整数的方式和我们平时一模一样，比如1，100，-123(负数)。计算机里因为是二进制保存数据，所以有时也会看到一十六进制表示的整数(十六进制里用0x表示前缀，值用0-9和a-f表示)，比如0xa2表示的整数就是162(2\*1+10\*16)。有时为了增加代码的可读性，我们也在数字中加入分隔符\_方便我们阅读，比如代码中写成10\_000\_000\_000和10000000000是完全一样的。



## 浮点型

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的。比如，1.23X10^9(1.23乘以10的九次方)和12.3X10^8是完全相等的。浮点数可以用数学写法，如1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把10用e替代，1.23x109就是1.23e9，或者12.3e8，0.000012可以写成1.2e-5，等等。整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的（除法难道也是精确的？是的！），而浮点数运算则可能会有四舍五入的误差。



## 空值类型

None类型表示一个null对象(没有值的对象)。Python提供了一个null对象，在程序中表示为None。如果一个函数没有显式地返回值，则返回该空值对象。None经常用作可选参数的默认值，以便让函数检测调用者(关于函数和参数在后续章节详细介绍)是否为该参数实际传递了值。None没有任何属性，在布尔表达式中求值时为False。

注意None不能理解为0，0是有意义的，而None表示什么也没有，是一个特殊的空值。

# 基础类型-字符串类型

我们见到过单个字符’a’、’@’。由这些单个字符组成的变量类型就是字符串。在新的Python3中，字符串是以Unicode编码的(关于Unicode编码方式这里不做过多介绍)，也就是说Python的字符串支持多语言。因为字符串是Python语言中特别重要的概念(不仅仅是Python，在很多编程语言中字符串也非常重要)，我们详细的讲解一下字符串的用法。我们已经知道，单引号，双引号，包括三引号包围的字符组，就是字符串，例如：

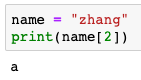


## 下标和切片

### 下标索引

下标在Python中的概念就是编号的意思，字符串、元组、列表都会经常到下标的概念，我们可以根据下标找到它们所对应的元素。就好像生活中你要准备去看电影，电影票上的座位号找到对应的位置．

我们现在考虑这样的一个问题，例如我们创建了一个字符串name=zhang，我现在想取到名为name字符串里面的a字符，如何去取呢？Python给我们提供了更为简便的方法，我们就可以下标来取出a字符：



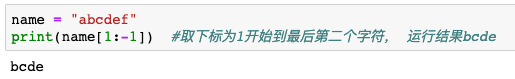
### 切片

我们可以利用下表索引取到字符串里面对应的一个元素，但如果想要截取一段元素就要用到切片。**切片**是指对操作的对象截取其中一部分的操作。字符串、列表、元组都支持切片操作。  
 切片的语法：[起始:结束:步长]

如果取出一部分，则可以在中括号[]中使用：



支持负数：



## 字符串的常见操作

假设有字符串mystr = “hello world python”,以下是常见的操作：

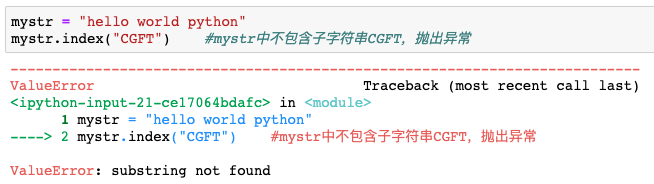
### find

检测str是否包含在字符串中，如果包含则返回开始的索引值，否则返回-1。例如：



### index

跟find()方法一样，只不过如果str不在mystr中会报一个异常(异常的概念会在后续章节介绍)。例如：



### count

返回str在mystr里面呢出现的次数，例如：



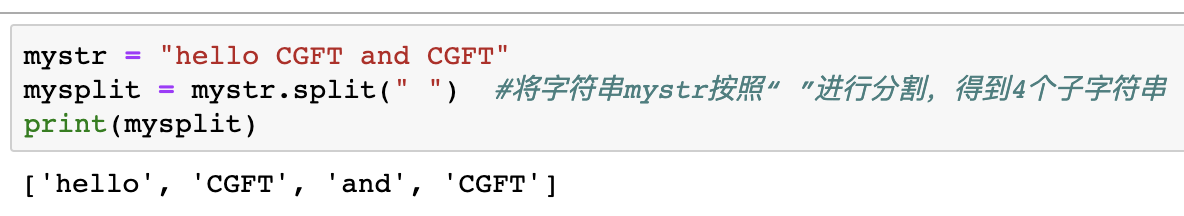
### replace

把mystr中的str1字符串替换为其他字符串，例如：



### split

将字符串按照指定字符或字符串进行分裂，例如：



### startwith

检查字符串是否以某字符或字符串开头，是则返回True,否则返回False。例如：



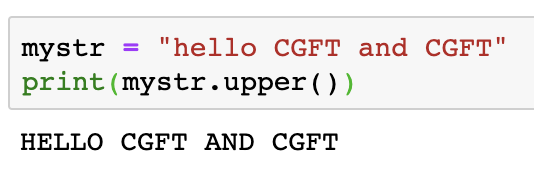
### endswith

检查字符串是否以某字符串结束，如果是返回True，否则返回False。



### upper

转换字符串中的字符为大写，例如：



### lstrip

删除字符串左边的空白字符，例如：



# 容器类型-列表list

## 列表简介

Python内置的一种数据类型是列表list。List是一种有序的集合，可以随时添加和删除其中的元素，写在方括号之间、用逗号分隔开的数值列表。列表内的项目不必全是相同的类型。例如：



## 列表的常见操作

### 列表的长度

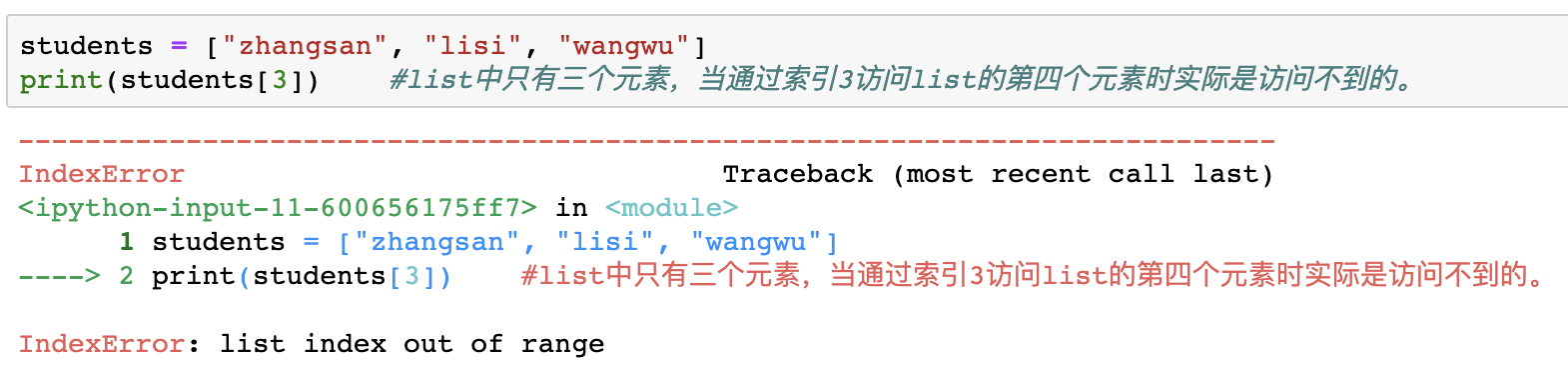


### 列表的访问

我们可以用索引来访问list中每一个位置的元素，注意索引是从0开始的：



注意：当索引超出了范围时，Python会报一个IndexError的错误。在编程时要确保通过索引访问列表元素时不要越界。记得最后一个元素的索引是len(students) – 1.



如果想获取list的最后一个元素，除了计算索引位置外，还可以使用-1直接获取最后一个元素：



### 列表的切片

切片：根据下标的范围获取一部分数据，像列表和之前介绍的字符串都可以使用切片。切片的格式为list[起始下标:结束下标:步长]。默认起始下标0，结束下标不包含，步长默认是1.



### 给列表添加元素

我们可以通过方法append, extend, insert为列表添加元素，例如：

通过append可以向列表添加元素：



通过extend方法可以将另一个集合中的元素全部添加到列表中：

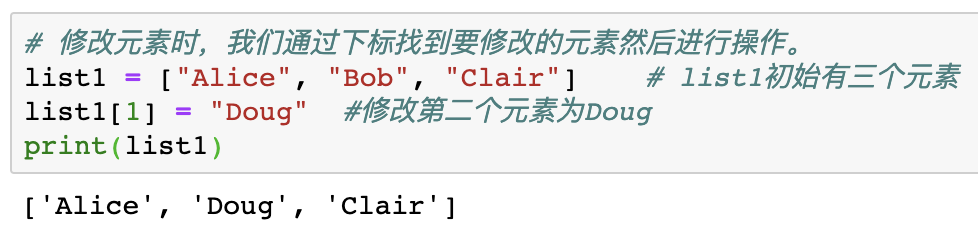


同样可以通过insert方法在指定位置插入新的元素：



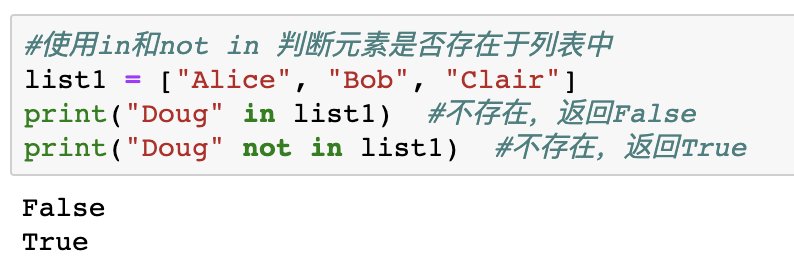
### 修改元素

修改元素时，我们通过下标找到要修改的元素然后进行操作。



### 查找元素

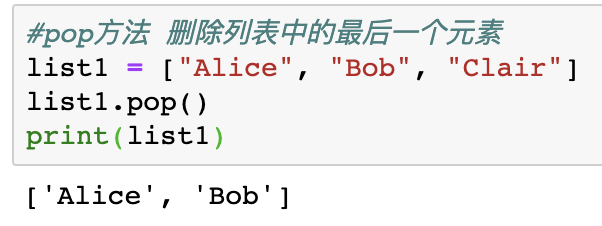
列表里的查找，就是看某个元素是否在列表中。Python的查找方式是运算符in，如果存在那么该运算返回True否则返回False；not in运算符用来判断元素是否不存在，如果不存在返回True否则返回False。



### 删除元素

删除列表中的元素通常由del,pop,remove三种方法，我们来分别介绍。

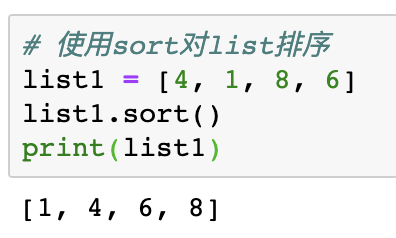






### 排序

sort方法可以将list按照特定顺序重新排列，默认为有小到大，参数reverse=True可以改为倒序。



### 列表遍历

我们也可以使用for循环来遍历列表里的所有元素。关于for循环会在后续章节详细介绍。



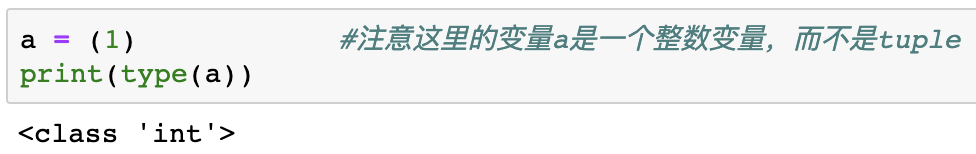
# 容器类型-元组tuple

另一种有序列表叫元组(tuple)。tuple和list非常类似，但是tuple一旦初始化就不能更改，比如同样是列出同学的名字。

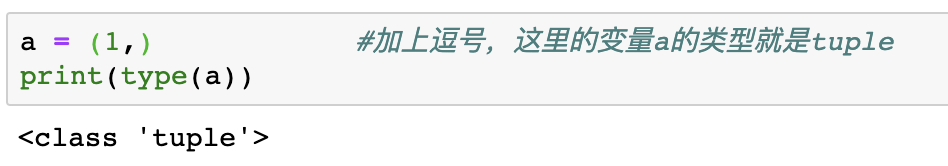


现在，tuple1这个tuple不能变了，它也没有append(), insert()这样的方法。其他获取元素的方法和list是一样的，我们可以正常的使用tuple1[0], tuple[-1]，但不能赋值成另外的元素。同学们可能会问，那不可变的集合有什么意义？因为tuple不可变，所以代码更安全。如果可能，程序中能用tuple或list的地方尽量使用tuple。

如何定义一个tuple？我们用括号表示一个tuple，括号内的所有元素用逗号分隔。注意，如果定义的是只有一个元素的tuple，如果我们这样定义：



定义的不是tuple，而是一个整形变量a！这是因为括号既可以表示tuple，又可以表示数学公式中的小括号(Python对被括号包含的表达式会作为一个整体处理)，这就产生了歧义。因此，Python规定，这种情况下，定义的是整形变量1。所以，只有一个元素的tuple定义时必须加一个逗号，来消除歧义：



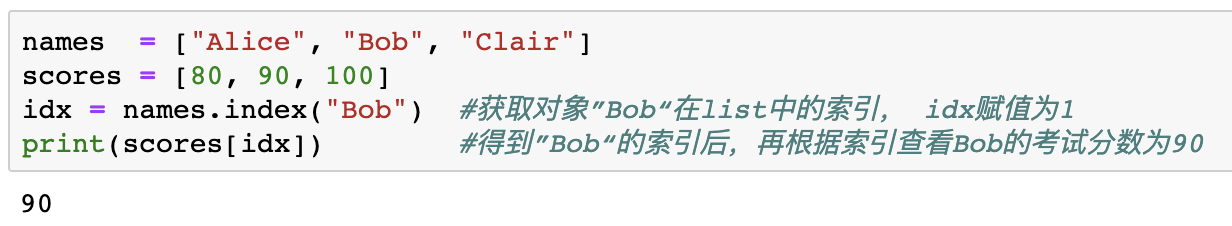
Python在显示只有一个元素的tuple时，也会加一个逗号，以免你误解为数学计算意义上的括号。

# 容器类型-字典dict

## 字典简介

字典是另一种可变容器模型，且可存储任意类型对象。

字典的每个键值对(key=>value)用冒号(:)分隔，每个对之间用逗号(,)分隔，整个字典包含在花括号{}中。因为我们使用key搜索查找键值对，所以字典里的key不能重复。举个例子，假设要根据同学的名字查找对应的成绩，如果用list实现，需要两个list:

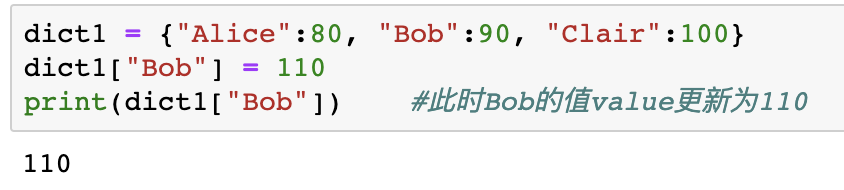


给定一个名字，要查找对应的成绩，就先要在names中找到对应的位置，再从scores取出对应的成绩，list越长，耗时越长。

如果用dict实现，只需要一个**”名字”=》”分数”**的对照表，直接根据名字查找成绩，无论这个表多大，查找名字并获取对应的分数都在这个集合里完成，查询速度相比list提高很多。用Python定义一个dict如下：



因为一个key只能对应一个value，所以，多次对一个key放入value，后面的值会把前面的值覆盖掉：



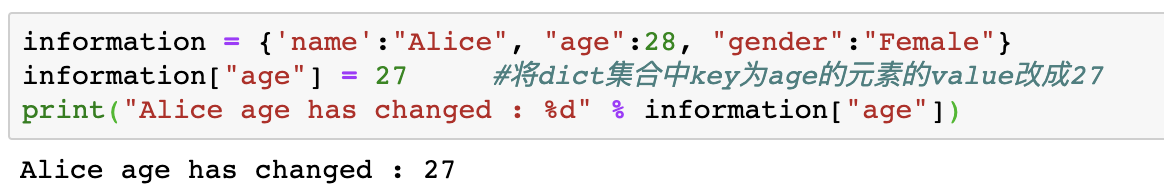
如果key不存在，dict就会报错：



## 字典的常见操作

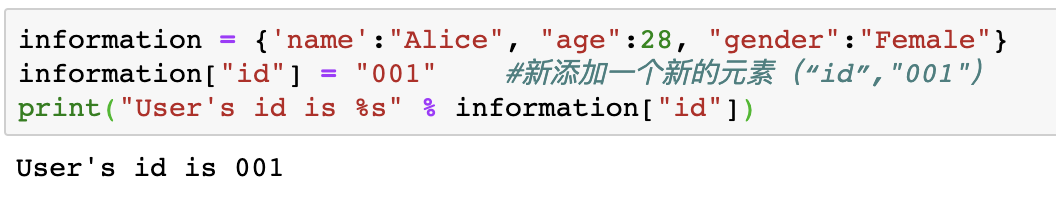
### 修改元素

字典中的元素的数据是可以修改的，只要通过key找到对应元素就可以对value进行修改。



### 添加元素

如果在使用变量名**[key] = value**时，这个键(key)在字典中不存在，那么就会在dict集合里新添加一个新的元素。



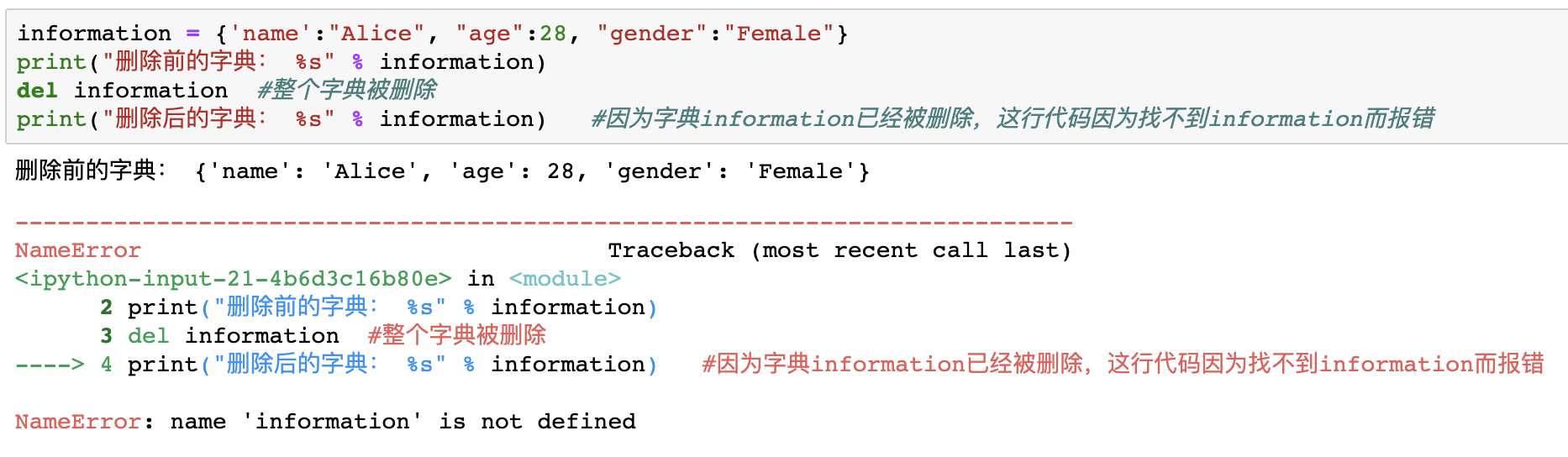
### 删除元素

对字典进行删除操作，有以下两种：**del()**和**clear()。**

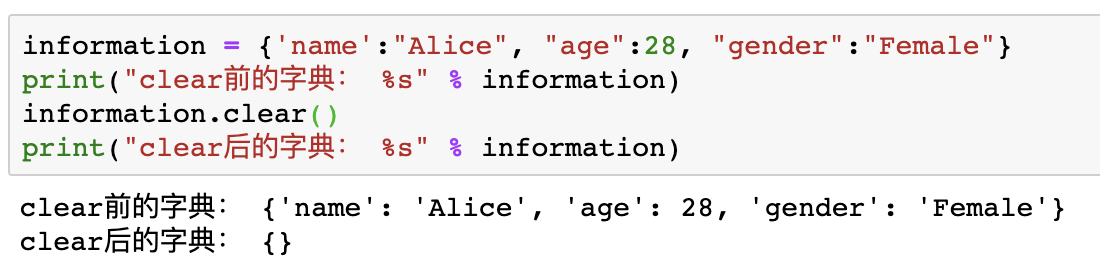
**del** 删除指定的元素：



**del** 删除整个字典：



**clear()**清空整个字典：



### 查看字典中有多少个元素

查看字典元素个数我们用方法len():



### 返回字典中的所有key

方法keys()返回一个包含所有key的列表：



### 返回字典中的所有value

方法values()返回字典的所有values：



### 返回一个包含所有元素(键值对)的列表



# 容器类型-集合(set)

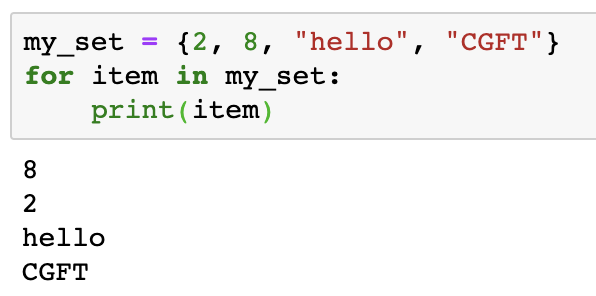
集合(set)是一个无序的不重复元素序列。

可以使用大括号{}或set()函数创建集合，注意：创建一个空集合必须用set()而不是{}，因为{}是用来创建空字典的。

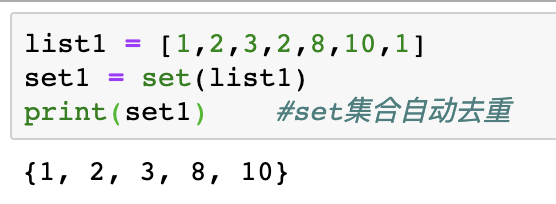
没有重复元素，set集合不支持通过索引返回指定位置元素：



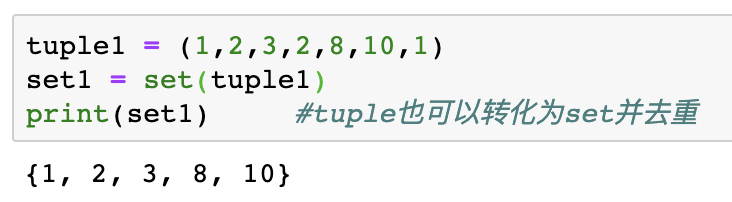
我们通过for循环遍历获取数据：



集合会对重复数据去重，下面的代码首先定义了一个list集合(包含重复数据)，当我们把列表(list)转换成集合set时，数据会被去重：



元组tuple也可以转换成set。list, tuple, set三者可以互相转化：



# 变量，对象，深拷贝与浅拷贝

该章节属于扩展理解章节，帮助我们深入理解变量及对象的关系，以及了解什么是深拷贝和浅拷贝。让我们先来定义三个概念：

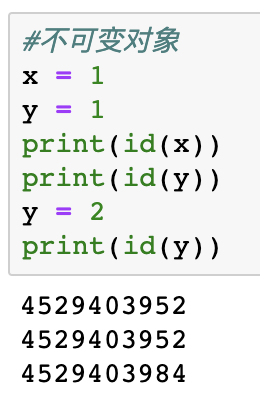
* 变量：是程序中我们定义的一个变量(可以想象成人的名字)
* 对象：是计算机分配的一块内存，用来存储真正的这个对象(可以认为真正的人)
* 引用：是自动形成的从变量到对象的指针，将内存中的对象用变量引用，进而可供程序的代码使用。

像a = 3，这段代码的执行过程是这样：首先内存中开辟一块内存，内存中存储的值为3；然后创建一个变量，最后将变量a指向内存中的对象3，将变量与新建的对象连接起来，程序里通过变量a来操作使用对象3。

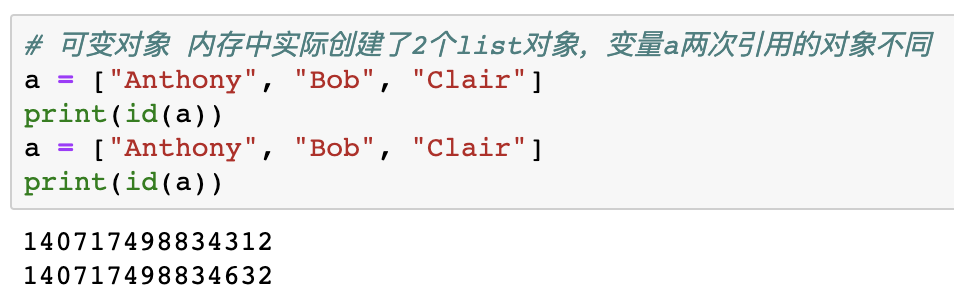
## 不可变对象和可变对象

在Python中，我们将对象分为可变(mutable)对象和不可变(immutable)对象：可变对象指可以在原处修改，而不用创建新的对象(包括列表，字典，集合以及我们自己定义的类)；不可变对象指不支持在原处修改，只能通过表达式创建新的对象，然后把结果分配给变量(包括数字，字符串，元组)。

在Python中，数值（整型，浮点型），布尔型，字符串，元组属于值类型，本身不允许被修改（不可变类型），数值的修改实际上是让变量指向了一个新的对象（新创建的对象），所以不会发生共享内存问题。下面的示例中，内置函数id()查看的是当前变量的指向对象的地址值。x = 1和y = 1两个操作的结果，从上面的输出可以看到x和y在此时的地址值是一样的，也就是说x和y其实是引用了同一个对象，即1，也就是说内存中对于1只占用了一个地址，而不管有多少个引用指向了它，都只有一个不可变对象1。不可变数据类型的优点就是内存中不管有多少个引用，相同的对象只占用了一块内存。



再来理解可变对象，下面的示例告诉我们，进行两次a = [1, 2, 3]操作，两次a引用的地址值是不同的，也就是说其实在内存汇总创建了两个不同的list对象，这一点明显不同于不可变数据类型，所以对于可变数据类型来说，具有同样值的对象是不同的对象，即在内存中保存了多个同样值的对象，地址值不同。可变数据类型是变量指向的可变对象的内容可以发生变化，即值可以变化，但是地址是不会变化的。

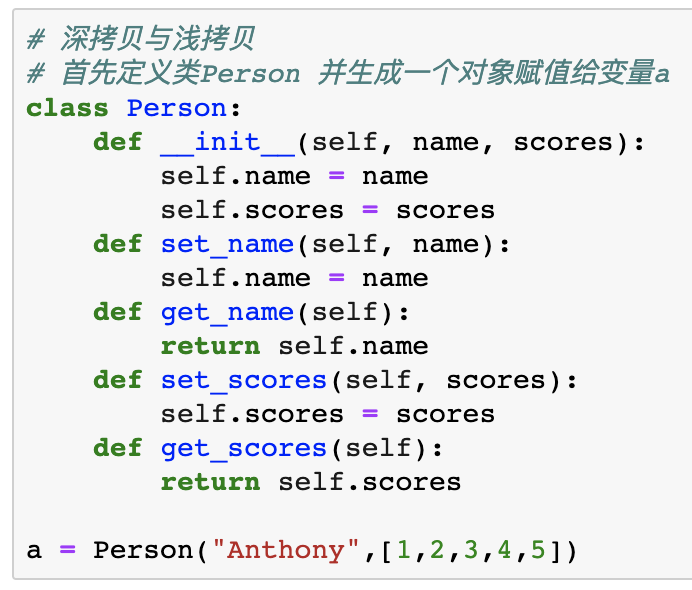


## 深拷贝与浅拷贝

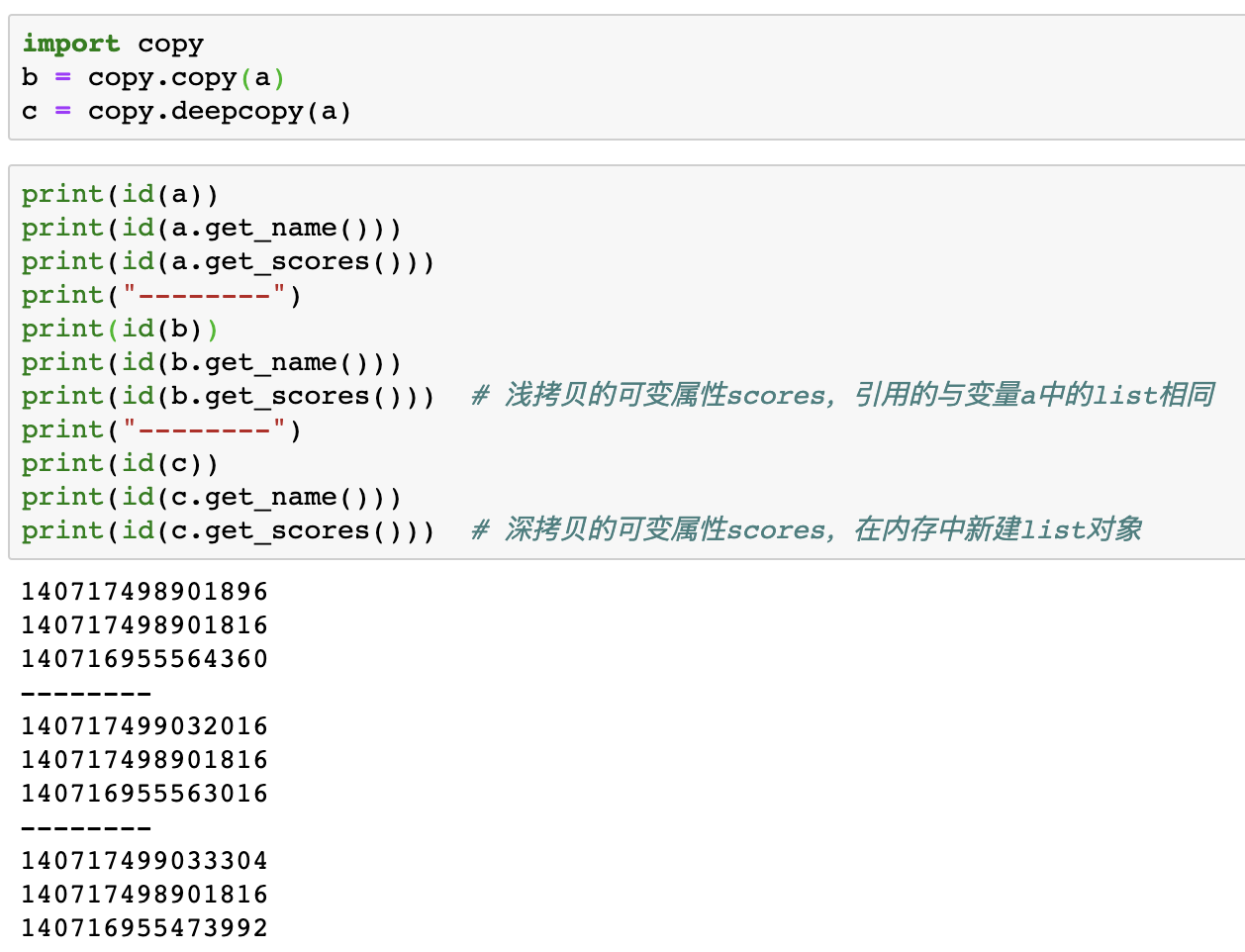
在程序中我们有时需要复制一个对象操作，但是又不希望对原有对象产生副作用，那就不能通过赋值给新变量来解决了(赋值只是将多个变量指向一个对象，不是克隆一个对象)。Python专门提供了一种拷贝机制，基于原对象创建一个含有相同值的对象。拷贝函数有copy模块提供。

拷贝分为深拷贝和浅拷贝，深拷贝和浅拷贝最大的区别是可变元素的拷贝：在浅拷贝时，拷贝出来的新对象的地址和原对象是不一样的，这一点和深拷贝相同，但是新对象里面的可变元素（如列表）的地址和原对象里的可变元素的地址是相同的，也就是说浅拷贝它拷贝的是浅层次的数据结构（不可变元素），对象里的可变元素作为深层次的数据结构并没有被拷贝到新地址里面去，而是和原对象里的可变元素指向同一个地址，所以在新对象或原对象里对这个可变元素做修改时，两个对象是同时改变的；但是深拷贝不会这样，深拷贝完全复制了一个新的对象来供程序使用，深拷贝的对象任何操作都不会影响原来对象的元素。这个是浅拷贝相对于深拷贝最根本的区别。

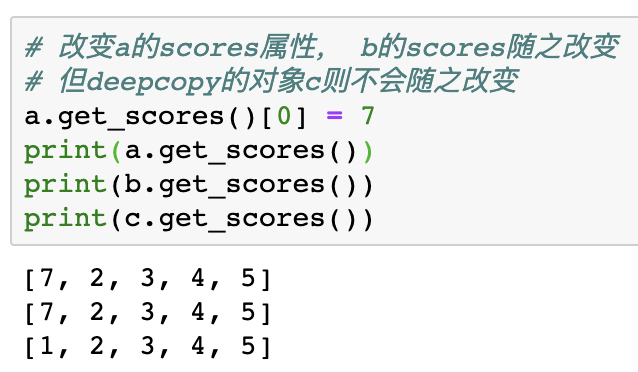
下面这个示例，我们首先定义类Person，并新建对象赋值给变量a:



浅拷贝(copy)的对象(变量b)的可变属性scores和原对象(变量a)相同，深拷贝(deepcopy)的对象(变量c)的的可变属性scores则单独在内存中保存一份，跟原对象(变量a)没有关联。



我们对a操作其可变属性scores，b的scores随之改变，但deepcopy的对象c则不会随之改变：



## 小结

* Python的变量创建过程是在代码第一次给他赋值就创建了变量，程序首先创建一个对象(包括可变对象或不可变对象，然后将变量指向内存中的对象)，之后我们再给这个变量赋值，实际上改变这个变量的指向，让它引用其它对象。
* Python的变量是动态类型，也就是没有明确类型的。变量是通用的，它可以引用任何对象，只是在一个特定的时间点，引用了一个特定的对象。
* Python中使用变量的时候，当变量出现在表达式中时，它会马上被所引用的对象所替代。使用没赋值的变量会产生错误。
* 浅拷贝拷贝出来的对象其可变属性和原对象共享；深拷贝会把里面的元素也重新拷贝一份，修改任何属性的值，不会对原对象产生影响。