第3章　编程基础

本章学习目标

- 掌握 Python 的基本语法，以及对应的编程环境；
- 了解 TensorFlow/PyTorch 基础知识。

3.1　Python 语 法

通常，学习 Python 的准备工作有以下两项：

(1) 从官网下载对应版本的 Python，并且配置好环境变量；

(2) 安装一个代码编辑器。

3.1.1　Python 基本概述

1. Python 简介

Python 是目前最受欢迎的编程语言之一，在设计上坚持了清晰划一的风格，这使得 Python 成为一门易读、易维护，并被大量用户欢迎的用途广泛的语言。Python 几乎可以完成计算机各个领域的任务，包括系统运维、图形处理、数学处理、文本处理、数据库编程、网络编程、Web 编程、多媒体应用、pymo 引擎、爬虫编写、机器学习和无人驾驶等。

Python 的名字由 Python 之父 Guido van Rossum 取自一部电视剧《蒙提·派森的飞行马戏团》(*Monty Python's Flying Circus*)。Python 支持将程序和与之对应的库进行打包，成为各个操作系统能执行的文件，其中包括常用的微软 Windows 平台下的.exe 文件，从而能够脱离 Python 的解释器环境和库，同时也能够制作微软格式的安装包(.msi 安装包)。Python 语言的语法简单灵活，易于掌握且功能强大。Python 可以支持面向对象设计和面向过程设计，作为传统的命令式语言，也可以支持函数式编程技术。更为重要的是，Python 拥有强大的社区以及对人工智能中各个领域的优秀扩展库。Python 目前被称为“胶水语言”，这是因为 Python 可以调用不同语言编写的模块，进而组合系统中这些封装好的包，发挥出不同语言各自的优势，满足各个领域中不同用户的差异化需求。

Python 的设计哲学是“优雅”“明确”“简单”。在设计 Python 语言时，如果面临多种选择，Python 开发者一般会拒绝花哨的语法，而选择没有或很少有歧义的语法。Python 开发人员应尽量避开不成熟或者不重要的优化，一些针对非重要部位的加快运行速度的补丁通常不会被合并到 Python 内。Python 本身被设计为可扩充的，并非所有的特性和功能都集

成到语言核心。Python 提供了丰富的 API 和工具，以便程序员能够轻松地使用 C 语言、C++ 、Cython 来编写扩充模块。Python 编译器本身也可以被集成到其他需要脚本语言的程序内。

2. Python 发展历程和版本说明

自 2004 年之后，Python 在编程语言市场占用率方面呈线性增长之势。目前，Python 的主流版本为 Python 2.x 和 Python 3.x 两个系列。这两个系列版本的用法大多并不相互兼容，主要区别有两点：①基本的输入输出方式不同；②使用的库和函数的用法不同。Python 3.x 系列对 Python 2.x 系列的标准库进行了增加、删除、合并和拆分修改，相对于标准库，其扩展库的相似度更低。因此，在进行工作和学习之前，应根据需求选择合适的版本。

市面上有能够有效保持 Python 3.x 和 Python 2.x 版本兼容性的工具和技术，但在许多代码的开发过程中，仍然会有冲突发生，开发者和维护者也不会一直保持两个版本同步更新的策略。并且，Python 官方于 2020 年 1 月 1 日起宣布停止对 Python 2.x 的更新支持。随着越来越多科研机构和公司使用和推荐 Python 3.x，这些 Python 包的作者自然也更倾向于放弃 Python 2.x，从而简化代码并更好地利用 Python 3.x 的许多新功能。Python 3.x 取代 Python 2.x 是必然趋势。

3. Python 编码规范

关于 Python 的简介或者学习资料中，多次提到了“优雅”这个词汇。“优雅”是因为 Python 具有良好的书写规范，只有开发者遵守规范，才能写出优秀的代码。本章介绍 Python 3.x 系列中的代码书写规则的要求以及一些良好的优化建议。

(1) 一般情况下，Python 3.x 编码的方式是 Unicode(可以不加)。

(2) Python 中对缩进是有严格要求的，因为 Python 不像 C 语言那样依靠括号，而是用缩进来标记代码的逻辑从属关系，若书写过程中代码块的缩进不正确，那么程序在逻辑上就会出现问题，导致程序结果出现偏差甚至报错。

(3) import 语句应该放在文件头部。每个 import 只能够导入一个模块，置于模块说明及 DocStrings 之后，全局变量之前。import 语句应该按照顺序排列，每组之间用一个空行分隔，如果发生命名冲突，则可以使用命名空间。

(4) 空格、空行和换行的使用。在运算符=，－，+=，==，＞，in，is not，and 等的两侧各增加一个空格，在列表(list)、元组(tuple)、字典(dict)、函数等的元素或参数列表中，逗号和冒号后面需要添加一个空格。在函数的参数列表和 print()中的 end 用法中，默认值等号两边不需要添加空格。不要为美观而在对齐赋值语句时使用额外的空格，尽量在完成一段功能完整的代码的情况下以及类定义和函数定义之后添加一个空行。Python 支持两种括号内的换行：①第二行缩进到括号中(一行显示，内容太多时)；②第二行缩进 4 个空格(内容较复杂时)。

(5) 一般情况下，一个语句占一行。if、for、while 等关键语句一定要换行。长字符串可以使用反斜杠“\”进行换行。Python 为了使得代码具有高可读性，在书写时应尽量避免使用长句式，句式过长时可使用“\”换行。DocStrings 文档字符串是一个重要工具，用于解释程序，使得程序文档更加简单易懂，所有的公共模块、函数、类、方法，都应该写 DocStrings；私有方法不一定需要 DocStrings，但应该在 def 后提供一个块注释来说明，DocStrings 的结

束"""应该独占一行，除非此 DocStrings 只有一行。

(6) 为了使得代码执行更加高效，应使用括号对运算的优先级和业务逻辑进行明确说明。关键部分需要添加代码注释，这也是所有编程语言的共性问题，Python 中使用 # 和三引号分别对单行和多行代码进行注释。

以上是 Python 的一些基本代码书写规范，但这些并不是全部规范，其他的需要读者在学习 Python 的过程中自行体会。

4. Python 命名规范

Python 中的变量需要使用标识符进行命名，所谓标识符其实就是用来给程序中的变量、类、方法命名的符号。简单来说，标识符就是合法的名字。Python 语言中的标识符必须以字母、下画线(_)开头，后面可以加任意数目的字母、数字和下画线。这里的字母并不局限于 26 个英文字母，也可以包含中文字符、日文字符等。

由于 Python 3.x 支持 UTF-8 字符集，因此 Python 3.x 的标识符可以使用 UTF-8 能够表示的多种语言的字符。而 Python 2.x 版本对中文的支持较差，所以如果想要在 Python 2.x 程序中使用中文字符或者中文变量，需要在 Python 源程序的第一行加上“# coding：utf-8”，然后将源文件保存为 UTF-8 字符集。

需要注意的是，Python 语言是区分大小写字母的，因此 abc 和 Abc 是两个完全不同的标识符。

在使用标识符时，需要遵循以下规则：

(1) 标识符可以由字母、数字、下画线组成，但是标识符的第一位不可以是数字。

(2) 标识符不可以是 Python 关键字，但可以包含关键字。

(3) 标识符中不可以含有空格。

在对 Python 语言中的文件名、包、模块等进行命名时，同样也有一些规则需要注意，从而保证命名的规范性。

(1) 文件名：全小写，可以使用下画线(_)。

(2) 模块和包：应该使用简短的、小写的名字。如果下画线可以改善可读性，则可以适当添加。

(3) 类：总是使用驼峰命名法，首字母大写。私有类可以使用一个下画线开头。

(4) 函数和方法：函数名应该小写，函数名如果含有多个单词，要用下画线隔开。私有函数可以在函数名前加一个下画线。

(5) 函数和方法的参数：总是使用 self 作为实例方法的第一个参数。总是使用 cls 作为类方法的第一个参数。

(6) 常量：常量名的所有字母都需要大写。

5. Python 开发环境的安装和使用

1) Python 的安装

(1) 前往 https://www.python.org/downloads/windows/ 下载最新版本的 Python 3.x，本文以 Python 3.6 为例说明。

(2) 双击下载包，进入 Python 安装向导，如图 3.1 所示。

需要注意的是，Python 3.6 已经可以自动添加环境变量，如果安装的是 Python 2.x 版

本,则需要手动配置环境变量,这里不做赘述。

图 3.1　Python 安装向导

(3) 选择安装路径为 C:\Program Files\Python36,如图 3.2 所示。安装路径可自由选择。

图 3.2　添加环境变量

(4) 安装完成之后需要检查 Python 是否安装成功。在 Windows 系统中检测 Python 是否成功安装,可以按键盘上的 Win 键+R 打开运行窗口,然后在运行窗口中输入 cmd,单击"确定"按钮进入命令行窗口。输入 python,按 Enter 键,如果显示类似图 3.3,则说明 Python 环境搭建成功。

(5) 安装完 Python 环境之后,需要选择适合自己的开发工具,虽然现在市面上有很多可以用来编写 Python 代码的编辑器,这些编辑器都各有优缺点。本书推荐新手使用 PyCharm。

2) PyCharm 下载

(1) Windows 下 PyCharm 的下载地址为 http://www.jetbrains.com/pycharm/download/

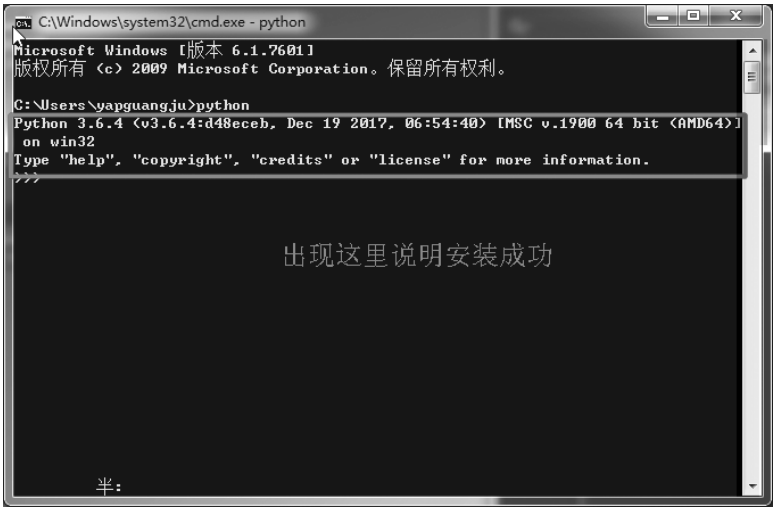

图 3.3　Python 环境成功搭建

＃section＝windows。

PyCharm 有两个版本：Professional 专业版，Community 社区版，推荐安装免费的社区版，如图 3.4 所示。

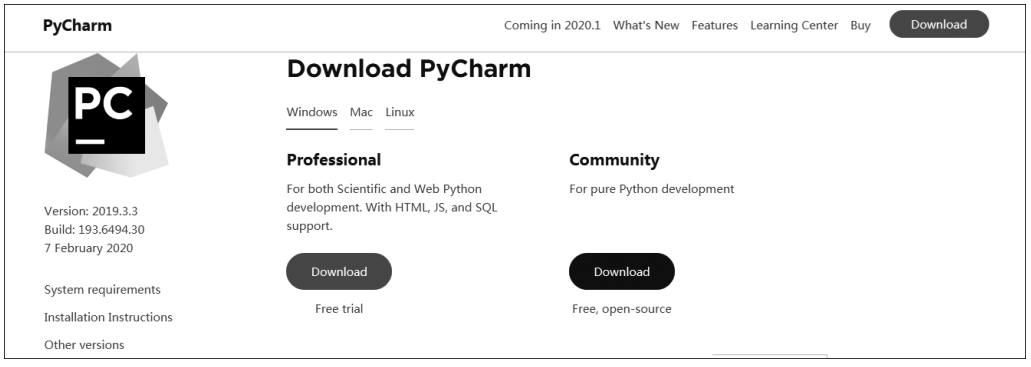

图 3.4　PyCharm 下载

（2）下载完成后，双击下载好的文件以安装，选择安装路径，单击 Next 按钮继续，如图 3.5 所示。

（3）根据计算机操作系统是 32 位或 64 位，选择对应需要安装的版本，单击 Next 按钮继续，如图 3.6 所示。

（4）单击 Install 按钮进行 PyCharm 安装，如图 3.7 所示。

3.1.2　Python 内置对象

1. 对象的概念

任何刚接触 Python 语言的人都会听过一句话："Python 中一切都是对象"。但是为什么这么说是不少刚接触 Python 的人心里都会有的疑问。编程语言有的支持面向对象，有的

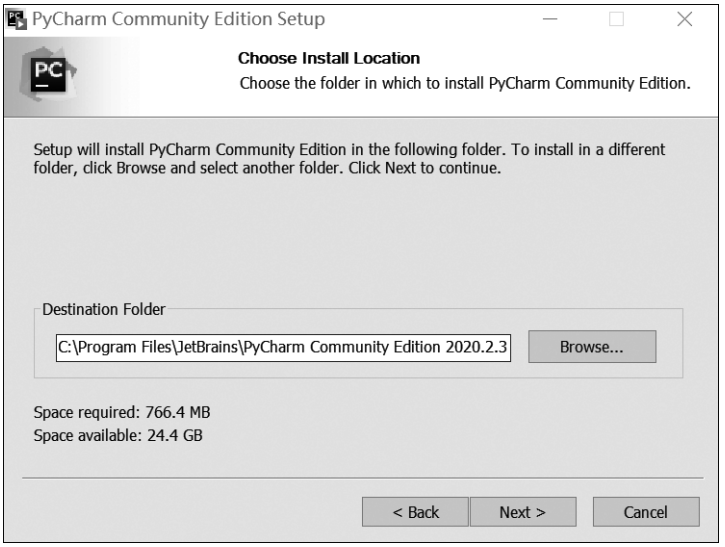

图 3.5　选择安装路径

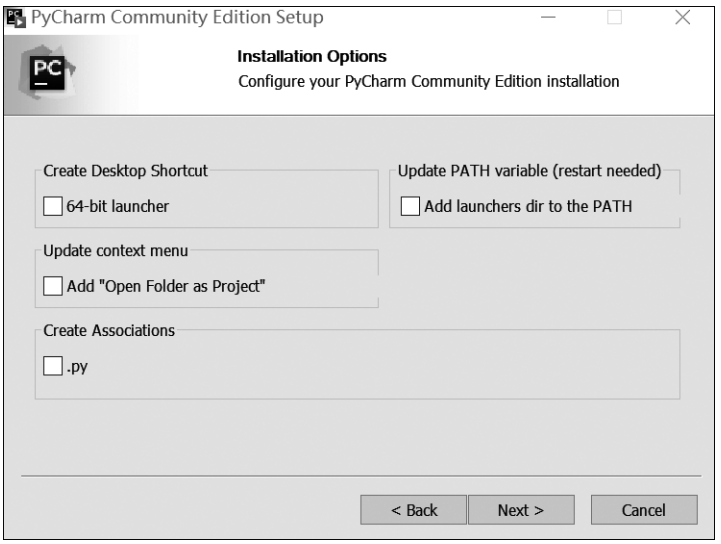

图 3.6　根据系统版本进行选择

支持面向过程,有的既支持面向对象也支持面向过程。虽然对象的定义语法和使用方法在各个编程语言中各有不同,但是“对象”这个概念在这些编程语言中是相对统一的。

所谓对象,就是指编程语言中相对独立的实体,它可以被用来调用、赋值或者作为参数供其他函数使用。那么为什么 Python 格外强调一切都是对象这个概念呢? 原因就是 Python 在支持对象的这条路上走得更远,因为在 Python 中所有的程序都由方法和数据组成。Python 中不仅每一项的数据都是对象,而且用来定义方法的函数、类等也都可以作为对象来存储和处理。总体来说,一切都可以赋值给变量。

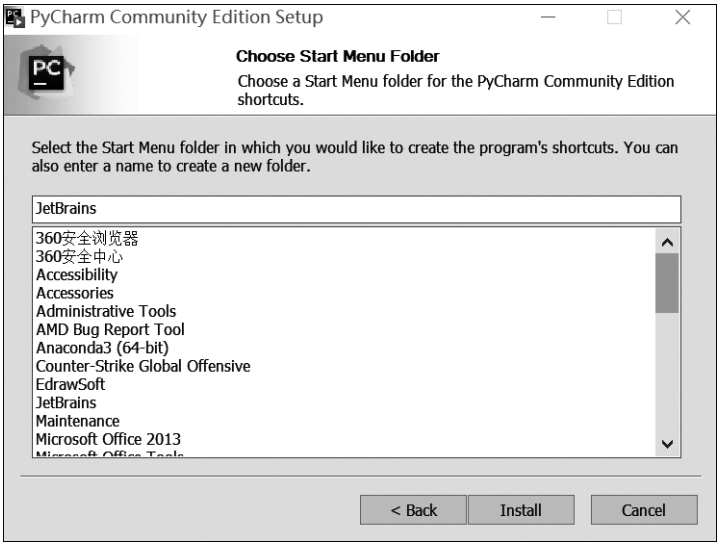

图 3.7　系统安装

2. 常量和变量

常量，顾名思义就是指不需要改变也不能改变的字面值。例如一个整数 6，一个列表[1,3,5]等。变量与常量相反，其值是可以变化的。在其他大多数编程语言中，变量都需要遵循先声明后使用的原则，然而在 Python 中，变量的名称和类型不需要经过事先声明，就可以直接被用户进行赋值操作。不仅变量的值可以随时改变，变量的类型也可以根据用户需求进行改变。

变量被进行赋值操作时的执行过程是这样的：首先计算出等号右边的表达式的值，然后在内存中寻找一个可以存储该值的位置，最后才会创建变量，然后把创建好的这个变量指向上一步中找到的内存地址。这就是 Python 中的变量类型也能够随时改变的原因。Python 的内存管理采用了基于值的内存管理模式，变量都是存储或引用值的内存地址，而不是直接存储变量的值。

3. Python 中常用的内置对象

Python 中主要的内置对象有：①数字(整型 int、浮点型 float)；②字符串(不可变性)；③列表；④字典；⑤元组(不可变性)；⑥文件；⑦集合(不重复性)。

在 Python 中有整数、实数和复数这三种内置数字类型。分数的实现和相关运算可以借助标准库 fractions 中的 Fraction 对象，如果想要实现精度更高的计算，则可以使用 fractions 库中的 Decimal 类。

Python 对数字的大小没有限制，但是会受到内存大小的影响。内存越大，Python 支持的数字就越大。因为实数之间的运算有时可能会存在一定的误差，所以不建议直接将两个实数进行相等性测试。出现误差是因为精度的问题，如果要把两个实数进行相等性比较，建议将二者进行相减运算，然后观察得到的差的绝对值是不是足够小。

在 Python 3.6.x 版本中，为了提高数字的可读性，用户可以使用单个的下画线来分隔数字，但是下画线不能出现在数字的首尾位置，另外使用连续的下画线也是不被允许的。

4. 字符串和字节串

字符串就是一串字符，是 Python 中一种表示文本的数据类型，可以使用单引号、双引号、三单引号、三双引号来定义一个字符串，单引号、双引号、三单引号、三双引号被称为定界符，不同的定界符可以相互嵌套。连接字符串的运算符为加号，除此之外，Python 还提供了很多的方法来支持字符串的查找、替换、排版等操作。

Python 3.x 版本除了支持 Unicode 编码的 str 类型字符串之外，还支持字节串类型 bytes。对 str 类型的字符串调用其 encode()方法进行编码，可得到 bytes 字节串，对 byte 字节串调用其 decode()方法并制定正确的编码格式则会得到 str 字符串。

需要注意的是，Python 字符串属于不可变对象，所以，所有实现字符串修改和生成操作的方法都是在另一个内存片段中新生成一个字符串对象。例如调用'abc'.upper()方法将会划分另一个内存片段，同时将返回的字符串 ABC 保存在这个内存片段中。

5. 列表、元组、字典、集合

Python 中比较常见的序列类型有列表、元组、字典和集合这四种，这些序列根据定义可以划分为有序序列和无序序列。列表、元组属于有序序列，都支持双向索引，而且 Python 中有序序列的一大特色就是能够使用负数来作为索引，这样能够很大程度地提高开发效率。接下来详细介绍四种序列结构各自的特点。

1) 列表

列表是一种有序的、可变的序列，列表中的元素可以重复，列表的定义符号为[]。可以使用序号作为下标，用逗号来分隔列表中的元素，一个列表中可以含有多个不同的数据类型，空列表的表示为一对方括号。列表之中的元素查找速度非常慢，可以说是四种序列类型之中最慢的。如果对列表进行新增和删除元素操作，只有尾部的操作会很快，其他位置的操作都很慢。

(1) 列表的创建和删除。

一般来说，创建列表有三种方法。

第一种方法是用[]直接创建，例如，a=[1,2,3]。

第二种方法是使用 list()来进行创建，list()函数的作用是把其他的序列转换成列表，例如，list((1,3,5))，这就是把元组转化成列表。除了元组外，range 对象、字符串、集合等也可以通过 list()被转化为列表。

第三种方法是使用列表生成式来进行创建。列表生成式是生成新列表的最快最高效的方法，总体来说就是对列表里面的数据进行运算和操作。例如：[x * x for x in range(1, 11)]。

当用户不再想使用某一列表时，可以用 del 删除整个列表，del 也可以用来删除元素，例如：

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

除了使用 del 可以起到删除的作用外，还有两种常用的方法可以用来删除列表中的元素。一种是 list 中的 remove()方法。通常来说，remove()可以删除列表中某一个值的第一个匹配项，用法为 list.remove(obj)。圆括号里的 obj 就是列表中要删除的对象。例如：

```
>>>a_list=[1,2,3,4, 'abc']
>>>a_list.remove(1);
>>>print  "List:  ",a_list
```

输出结果为：

```
List: [2,3,4,'abc']
```

还有一种是 list 的 pop()方法。一般来说，pop()可以用来删除列表中的一个元素，但默认为最后一个元素，然后返回被删除元素的值。用法为 list.pop([index=-1])，默认 index=-1。例如：

```
>>>list1 = ['baidu', 'Firefox', 'Taobao']
>>>list_pop=list1.pop(1)
>>>print("删除的项是 :", list_pop)
>>>print("列表现在是 : ", list1)
```

输出结果为：

```
删除的项是 : baidu
列表现在是 :  ['Firefox', 'Taobao']
```

很多时候，我们希望得到列表之中最早放入的那个元素，这个时候就可以使用 pop(0)。

(2) 列表的插入操作。

简单的创建和删除满足不了人们对于列表操作的需求，下面介绍如何在表之中插入元素。

有三种方法可以用来实现向列表对象中增加元素的操作。这三种方法分别是 append()、insert()和 extend()。append()方法一般用来在列表的尾部追加元素，insert()方法则可以在列表的任意位置插入元素，而 extend()方法是两个列表之间的操作，在列表的尾部位置追加另一个列表的全部元素。这三种方法都不会对列表对象在内存中的起始地址有任何的影响，属于原地操作。如果要向长列表中增加元素，不建议使用 insert()方法，因为列表拥有内存自动收缩和扩张的功能，所以在列表的中间位置插入元素或删除元素的效率很低。

(3) 列表的计数操作。

一般来说，编程中常常会遇到想要得到一个列表中某个元素出现次数的情况。遇到这种情况时，就可以使用 count()方法，count()方法的作用是返回列表中指定元素出现的次数。例如：

```
>>>x=[1,1,1,3,5,6]
>>>x.count(1)
>>>x.count(3)
```

输出结果为 3 和 1。

当然，有时可能也需要找到某一个元素在列表中第一次出现的位置，这时就可以使用 index()方法。index()可以返回指定的元素在列表中第一次出现的位置，如果指定的这个元素不存在，则会抛出异常。

(4) 列表的排序操作。

排序是一个在编程中使用较为频繁的操作，Python 中的列表对象有两种排序方法。一种是列表对象中的 sort()方法，sort()方法可以把列表中的所有元素按照一定的规则进行排序，默认的规则是从小到大进行排序。还有一种是 reverse()方法，reverse()方法可以把列表中的所有元素进行翻转。所谓的翻转就是指把列表中的第一个元素和最后一个元素交换位置，第二个元素跟倒数第二个元素互相交换位置，以此类推。这两种排序方法也属于原地操作。即用处理后的数据来替代原数据，从而使得列表的首尾位置不发生改变，另外这两种排序方法也没有返回值。

(5) 列表的复制操作。

在讲列表的复制操作之前，首先需要了解一下浅复制和深复制。浅复制指引用原列表中的所有元素的引用，复制到一个新生成的列表里。如果原来的列表中含有字典等可变数据类型，无论是修改原来的列表还是新的列表，它们都会互相影响。因为被复制对象的所有变量都含有与原来对象相同的值，而所有其他对象的应用仍然指向原来的对象。简单来说，浅复制仅仅复制所考虑的对象，而不是复制它所应用的对象。下面用一个例子来加深理解。

```
>>>x=[1,2,3,4,[5,6]]
>>>y=x.copy()
>>>print(y)
```

此时 y 复制了 x 列表中的所有元素，y=[1,2,3,4[5,6]]。然后我们分别改变一下 x 和 y 中的元素。

```
>>>y[2].append(7)
>>>y.append(8)
>>>x[0]=7
```

此时 y=[1,2,3,4,[5,6,7],8]，x=[7,2,3,4,[5,6,7]]。从这个例子中可以很明显地看出，在浅复制时，整数、实数等不可变的数据类型不会因为一个列表中对应元素的改变而改变。只有像列表、字典这种可变的数据类型才会受到这种影响。

如果想要避免这种影响，那么就可以采用深复制的方式。所谓的深复制就是把原来列表中所有元素的值进行复制，然后填充到新生成的列表中。采用深复制方法，进行操作的两个列表都是相互独立的，无论修改哪一个列表，另一个都不会受到影响。深复制使用的函数是 deepcopy()。

需要注意的是，不管是浅复制还是深复制，都不同于列表对象的直接赋值。

(6) 列表的切片操作。

通过前面的介绍可以知道，如果想要查找序列类型中的单个元素，可以使用索引值来实现。但是在很多时候，我们需要得到一个列表或者是元组中的一部分元素，那么这个时候就需要借助切片操作来灵活地处理序列类型的对象。

切片是 Python 序列中很重要的操作之一，是一种用来获取索引片段的方式，但是只有列表的切片操作功能是最为强大的。对列表进行切片操作不仅可以获得列表中的任意部分元素并返回得到一个新列表，而且还可以修改或删除列表中的一部分元素，甚至还可以为列表添加元素。下面介绍切片的一些比较常见的用法。

切片的使用形式为[start:end:step]，一般使用两个冒号分隔的三个数字来完成。start 是切片的起始索引值，当它是列表中的第一位时可以省略；end 表示切片的结束(但不包含)位置，当它为列表中最后一位时可以省略；step 表示步长，默认值为 1，且当 step 为 1 时可以省略，同时省略最后一个冒号，step 不允许为 0，当 step 是负数时，则表示反向切片。需要注意的是，start、end 和 step 这三个值都可以比列表的长度大，并且不会显示越界的报错信息。

总体来说，切片的基本含义就是从序列的第 start 位索引起，向右取到 end 位元素位置(不包括第 end 位)，以 step 为步长进行过滤。下面简单介绍如何使用切片为列表增加、替换、修改和删除元素。

使用切片操作为列表增加元素属于原地操作，因为虽然把元素插入到了列表的任意位置，但并不会对列表对象的内存地址有任何影响。例如：

```
>>>x=[3,5,6]
>>>x[len(x):]=7
>>>x[:0]=[1,2]
>>>x[3:3]=[4]
```

输出结果为 x=[1,2,3,4,5,6,7]。x[len(x):]=7 表示在列表的尾部插入一个元素 7，x[:0]=[1,2]表示在列表的头部插入两个元素 1 和 2，x[3:3]=[4]则表示在列表的中间位置即第三位之前插入一个元素 4。

同样用一个例子来了解切片是如何替换和修改列表中的元素的。例如：

```
>>>x=[2,4,6,8]
>>>x[:2]=[0,1]
```

此时的输出结果为 x=[0,1,6,8]。程序相当于用一个[0,1]的列表来替换原来列表的前两个元素。需要注意的是，等号两边的列表长度需要相等。

最后再来介绍如何使用切片操作来删除列表中的元素。例如：

```
>>>x=[2,4,6,8]
>>>x[:3]=[]
```

此时的输出结果为 x=[8]。x[:3]=[]语句的意思就是用一个空列表代替原列表中第三位之前的所有元素，从而达到删除效果。除了这种方法以外，还可以使用 del 命令跟切片结果来删除列表中的部分元素，并且切片元素可以是不连续的。

关于切片操作，需要格外注意的是切片得到的列表是浅复制。

2) 元组

通过前面的知识，读者应对列表有了大概的了解。下面介绍另一种序列类型，元组。元组被称为轻量级的列表，它拥有列表的部分功能，并且运行效率也很高。可能有人会想，既然元组只拥有列表的部分功能，为什么还要推出使用呢。列表的功能虽然很强大，但是正因为如此，列表的负担也很重，从而导致运行效率上并没有特别高效。并且在很多情况中，我们也并不需要使用列表的全部功能，大多的时候只需要使用其中的部分功能，所以元组便有了用武之地。

元组是一种有序的、不可变的序列，元组之中的元素可以重复，定义符号为()。可以使用序号作为下标，用逗号来分隔元组中的元素，如果元组之中只有一个元素，那么需要在这个元素后面加一个逗号。元组之中的元素查找速度很慢，并且不允许对元组之中的元素进行新增和删除操作。

元组的创建有两种较为常见的方法。一种是直接创建，如 x=(1,2,3)；还有一种是利用 tuple()方法来进行创建，如 x=tuple(range(5))。

元组的删除跟列表一样，都可以使用 del 进行删除操作。

由于元组是不可变序列，所以不能进行修改操作。元组跟列表一样都支持双向索引以访问其中的元素。元组也支持切片操作，不过因为元组是不可变序列，所以只能通过切片操作来访问元组中的元素，而不能够通过切片操作来对元组中的元素进行修改、增加和删除。

3）字典

Python 中还有一种序列类型称作字典，它一般用来存放具有映射关系的数据。字典中可以存储任意类型的对象，例如字符串、数字、元组等。

字典是一种无序的、可变的序列。字典中元素的形式为“键：值”，“键”必须是可哈希的，且不允许重复。因为字典中的“键”是非常关键的数据，而且一般程序需要通过“键”来访问“值”，所以字典中的“键”不允许重复。“值”是可以重复的，定义符号为{}，使用“键”作为下标，每个元素的“键”和“值”之间使用冒号分隔，字典中的不同元素使用逗号分隔。字典之中的元素查找速度非常快，元素的新增和删除操作也有很快的速度。

（1）字典的创建与删除。

一般来说，有三种方法可以用来创建字典。第一种是直接使用运算符“=”将一个字典赋值给一个变量，如：x={'Name': 'Zara', 'Age': 7, 'Class': 'First'}；第二种方法是用 dict()方法进行创建，如：x = dict(spinach = 1.39, cabbage = 2.59)；还有一种是使用 dict(zip(list1,list2))来根据已有的数据创建字典。

和前面已经介绍过的序列对象一样，当用户不再需要这个字典时，可以直接删除字典，具体使用的方法是相同的。

（2）字典元素的访问。

访问字典元素比较常用的方法有三种。一种是通过“键”来访问。由于字典中存放的数据元素都具有一定的映射关系，所以可以根据提供的“键”作为下标来访问字典中对应的“值”，如果字典中没有这个“键”，则会抛出异常。所以当使用“键”来对字典中的元素进行访问时，最好可以配合条件判断或异常处理结果，这样能可以有效地避免程序运行时因引发异常而导致崩溃。

第二种方法是调用 get()方法对字典里的元素进行访问。

最后一种方法就是对字典里的元素直接进行迭代操作或是遍历操作。无论是迭代操作还是遍历操作，默认的都是遍历字典的“键”。

（3）字典中元素的添加、修改和删除。

当用“键”来作为下标给字典中的元素进行赋值操作时，如果指定的这个“键”存在，那么就表示修改“键”在字典中相对应的“值”；如果字典中没有指定的这个“键”，就代表着在字典中添加一个新的“键”和一个新的“值”，也就是向字典中添加一个元素。

除了这种方法可以向字典中添加元素外，还有两种比较常见的方法可以实现这一操作。

第一种是使用字典对象的 update()方法，这个方法的作用就是把一个字典里的所有元素(键值对)一次性地全部添加到另一个字典中。如果进行操作的这两个字典中含有相同的“键”，那么就保留原始字典中这个“键”对应的“值”，对另一个字典进行更新。

第二种方法是使用字典对象的 setdefault()方法把新的元素添加到字典里。

和之前介绍过的列表、元组一样，当不需要使用字典的时候，就可以使用 del 命令来进行删除。

倘若想要删除字典中的元素并且得到它的返回值，则可以使用字典对象的 pop()方法和 popitem()方法。

4) 集合

最后一种介绍的序列类型就是集合。集合是一种无序的、可变的序列，集合之中的每一个元素都是唯一的，不可以重复，定义符号为{}。不可使用序号作为下标，集合中的元素由逗号分隔。集合中的元素查找速度非常快，如果对集合进行新增和删除元素操作的话，速度也很快。

(1) 集合的创建与删除。

创建集合有两种方式：一种是直接利用赋值语句将集合赋值给一个变量，从而创建一个新的集合对象；还有一种是利用 set()函数创建集合，set()函数可以把列表、元组、字符串等其他可以用来迭代的对象转化为集合。需要注意的是，如果待转换的可迭代对象中含有重复的元素，那么在转化为新的集合时系统只会保留一个，如果可迭代的对象中包含不可哈希的值，则会抛出异常，无法将其转化为集合。

与前面介绍过的几种序列类型一样，当用户不需要再使用某个集合时，就可以用 del 命令删除整个集合。除了这种方法之外，还有三种方法可以用来实现对集合中元素的删除。

第一种方法是集合对象中的 pop()方法。使用 pop()方法可以随机删除集合中的某一个元素，如果进行操作的集合是空集合，程序就会抛出异常。

第二种方法是集合对象中的 remove()方法。remove()方法可以用来删除集合中的元素，如果集合中没有指定要删除的元素，那么就会抛出异常。与 remove()方法类似的还有 discard()方法，该方法与 remove()方法的区别是，如果集合中没有要删除的指定元素，discard()方法并不会抛出异常，只会忽略这一操作。

最后一种方法就是 clear()方法，用于清空整个集合。

(2) 集合的操作和运算。

如果想要向集合中添加元素，那么可以借助两种方法实现：第一种是集合对象的 add()方法，如果集合中已经存在将要增加的元素，那么程序会自动忽略这一操作；第二种方法是集合对象的 update()方法，其作用是把一个集合中的所有元素合并到另一个集合之中，并且会主动过滤掉重复的元素。

内置函数 len()、main()、min()、sum()、sorted()、map()、filter()、enumerate()等也都是在集合中适用的。

3.1.3　Python 运算符和表达式

在 Python 中，最简单的表达式就是单个常量或者是变量。使用赋值运算符等任意运算

符或函数连接的式子都属于表达式。

Python 除了有算术运算符、关系运算符、逻辑运算符和位运算符等比较常见的运算符之外，还有一些 Python 特有的运算符，如成员测试运算符、集合运算符、统一性测试运算符等。Python 的很多运算符在不同语句中的含义是不一样的，使用起来十分灵活。

在运算符优先级规则中，优先级最高的是算术运算符（主要用于两个对象的加、减、乘、除等算术运算），其次是位运算符（对 Python 对象进行按照存储的位操作）、成员测试运算符（判断一个对象是否包含另一个对象）、关系运算符（用于判断两个对象的相等、大于等运算）、逻辑运算符（用于与或非等逻辑运算）等。在算术运算符中，数学中的先乘除后加减这一运算法则也是适用的。当编写比较复杂的表达式时，可以适当地使用圆括号来让表达式中的逻辑更加明确，从而提高代码的可读性。

下面详细了解各种运算符在 Python 中的应用。

1. 算术运算符

算术运算符是 Python 运算符中优先等级最高的运算符，主要有加、减、乘、除等运算。

“+”运算符在不同的语句中代表着不同的含义，在数字运算中表示加法运算，但是在列表、元组、字符串等对象之中就表示对同类型的对象进行相加或者是连接操作。需要注意的是，使用“+”运算符进行运算的对象必须是同一种类型的。

“-”运算符只能用于数字类型的对象间的运算操作。

“*”运算符不仅可以在数字运算中表示乘法，还可以用于列表、元组、字符串等序列和整数之间的乘法。序列和整数之间的乘法代表着序列元素的重读，然后生成新的序列对象。因此，字典和集合这两个不允许元素重复的序列类型是不支持与整数的相乘运算的。

“/”运算符在 Python 中表示算术除法，“//”运算符在 Python 中表示算术求整商。需要注意的是，当“//”运算中两个数都是整数的时候，结果就为整数。如果这两个被操作数中含有实数，那么结果就会显示为实数形式的整数值。

“%”运算符一般用于求余数的运算，在有些情况中，也用%运算符来对字符进行格式化。

“**”运算符和内置函数 pow()的功能一样，都是幂乘运算。

2. 关系运算符

当使用关系运算符的时候，必须留意进行操作的数据是不是可以比较大小。如果这些操作数不可以比较大小，那么就不可以使用关系运算符。关系运算符的含义和人们日常生活中的理解是完全一致的。此外，关系运算符是可以连用的。

3. 成员测试符和同一性测试符

in 在 Python 中是成员运算符，它的功能就是判断一个对象是不是被包含在另一个对象中。同一性运算符是 is，一般用来测试两个对象是不是同一个，如果这两个对象是同一个，那么返回值为 True；如果这两个对象不是同一个，那么返回值为 False。如果这两个对象是同一个，那么这两个对象的内存地址是完全相同的。

4. 位运算符和集合运算符

可以使用位运算符的数据类型只有整数。具体的执行过程如下：第一步先把整数转化为对应的二进制数；第二步是对这个二进制数进行位运算，在进行位运算之前，必须右对齐，有的时候需要在左侧补上 0；最后一步就是将计算得到的结果再转化为十进制数返回。位运

算符有按位与运算符、按位或运算符、按位异或运算符等。

按位与运算符是 &，代表的意思是，参与运算的两个值如果两个相应位都为 1，则该位的结果为 1，否则为 0。例如，设 a 为 60，b 为 13(对下面所有位运算皆相同)，则(a & b) 输出结果为 12，二进制为 0000 1100。

按位或运算符是|，代表的意思是，只要对应的二个二进位有一个为 1 时，结果位就为 1。例如，(a | b) 输出结果为 61，二进制为 0011 1101。

按位异或运算符是^，所代表的意思是，当两个对应的二进位相异时，结果位就为 1。例如，(a ^ b) 输出结果为 49，二进制为 0011 0001。

按位取反运算符是～，所代表的意思是，对数据的每个二进制位取反，即把 1 变为 0，把 0 变为 1。～x 类似于 －x－1。例如，(～a) 输出结果 －61，二进制为 1100 0011，一个有符号二进制数的补码形式。

左移动运算符是＜＜，所代表的意思是，运算数的各二进位全部左移若干位，由 ＜＜ 右边的数字指定移动的位数，高位丢弃，低位补 0。例如，a ＜＜ 2 输出结果为 240，二进制为 1111 0000。

右移动运算符是＞＞，所代表的意思是，将＞＞左边的运算数的各二进位全部右移若干位，＞＞ 右边的数字指定了移动的位数。例如，a ＞＞ 2 输出结果为 15，二进制为 0000 1111。

5. 逻辑运算符

逻辑运算符有 and、or 和 not 三种，通常用于将条件语句进行连接，从而组成更为复杂的条件表达式。当使用 not 这个逻辑运算符的时候，返回值只能是 True 或 False，但是在使用 and 和 or 这两个逻辑运算符的时候，就不一定会返回 True 或者 False，因为这两个逻辑运算符最终返回的值是通过最后一个表达式计算出来的。需要格外注意的是，因为 and 和 or 这两个逻辑运算符具有惰性求值或者逻辑短路的缺点，所以当它们连接比较多的表达式时，只会计算必须要计算的值。

6. 乘法矩阵运算符

乘法矩阵符@，顾名思义就是用来实现矩阵中的乘法运算的运算符。这个运算符是在 Python 3.5 版本中新增的，并且 Python 3.5 以后的版本都支持这个运算符。基本来说，乘法矩阵运算符@都是和 NumPy 扩展库一起使用的，原因是 Python 中并没有内置的矩阵类型。

Python 中的运算符除了上面介绍到的这些，还有大量的复合赋值运算符，这里不作过多的描述。关于 Python 中的运算符，还有一个需要注意的地方。像＋＋和－－运算符在有些其他语言中是可以使用的，但是实际上 Python 并不支持＋＋和－－这两个运算符，要把 Python 和其他的语言区别开。

3.1.4　程序控制结构

前文已经介绍了 Python 中的数据类型、数据结构、运算符等，那是不是说我们就可以来实现一些简单的业务逻辑了呢？其实并不是这样，正所谓“路漫漫其修远兮”，这些远远不是 Python 的全貌。接下来介绍 Python 中比较重要的一个部分——程序控制结构。学习这部分内容后，就可以结合读者前面已经掌握的内容来实现一些特定的业务逻辑，把学到的东西

都串联起来。

程序控制结构的三种基本结构分别是顺序结构、分支结构和循环结构，无论是哪一个结构，其执行流程都是要借助条件表达式的值来确定的。

(1) 顺序结构：根据字面上的意思，可以知道这是一种按照顺序来依次执行的控制结构。例如，语句块 1＝＞语句块 2＝＞……＝＞语句块 n。

(2) 分支结构：在顺序结构的基础上多了选择，根据不同的判断条件来执行不同的语句。二分支结构就是最基础的分支结构，它一般用来根据给出的指定条件来进行判断，得到 True 或者 False 结果，然后根据判断得到的结构选择要执行的语句。多分支结构就是由多个二分支结构组成的。

(3) 循环结构：按照判断分支结构向后执行的一种控制结构。它主要是根据结果来判断循环体中的语句需不需要再一次执行。下面详细介绍分支结构和循环结构。

1. 分支结构

分支结构中最简单的是单分支结构，一般使用 if 对给出的指定条件进行判断。语法形式如下：

```
if 判断条件:
        语句块
```

判断条件后面必须要加上冒号，只有加上冒号才表示一个语句块已经开始，而且，这些语句块必须要有相应的缩进，一般来说都是以 4 个空格为缩进单位。需要注意的是，Tab 键也可以用来缩进，但是不能跟空格混用。

当条件判断语句的值为 True 的时候，就表示在程序中这个条件是满足的，然后就会执行下面的语句块。如果条件判断语句的值为 False，那么接下来的语句块将会被跳过。

双分支结构在单分支结构上多了一个可供选择的语句，一般使用 if-else 来对给出的条件进行判断。语法形式如下：

```
if 判断条件:
    语句块 1
else:
    语句块 2
```

双分支结构的语句执行规则是这样的。当条件判断语句的值为 True 时，执行语句块 1 跳过语句块 2，当条件判断语句的值为 False 时，跳过语句块 1 执行语句块 2。

二分支结构除了这种表达形式之外，还有一种比较简洁的形式。举例如下：

```
s = eval(input('请输入一个整数:'))
t = '是'if s%3 == 0 and s%5 == 0 else'不'
print('{}{}能被 3 和 5 整除!'.format(s,t))
```

只有当语句块 1 和语句块 2 都是简单表达式时，才可以使用这种形式。如果这两个语句块比较复杂，那么还是推荐一般的表达式。

多分支结构通常可以看作由很多个二分支组成，使用 if-elif-else 可对给出的条件进行判断，该语句一般用来判断同一个条件或一类条件的多个执行路径。需要注意的是，Python

会按照分支结构的代码顺序依次判断，无论是哪一个判断条件成立，就会执行相应语句，然后跳出整个 if-elif-else 结构。若全部判断条件全都不成立，那么就执行 else 中的语句。语法形式如下：

```
if 判断条件:
        语句块 1
elif 判断条件
        语句块 2
elif 判断条件
        ⋮
else:
语句块 n
```

如果想要实现比较复杂的业务逻辑，那么就可以把分支结构进行嵌套。嵌套分支结构时，一定要注意控制好不同级别代码块的缩进量，因为缩进量不仅决定着不同代码之间的从属关系，而且还影响着程序能否正确地实现业务逻辑。

2. 循环结构

循环结构主要有遍历循环和无限循环两种，不仅可以把循环结构进行嵌套使用，还可以把循环结构和分支结构进行嵌套使用，从而实现比较复杂的业务逻辑。

(1) 遍历循环。

遍历循环使用保留字 for 对遍历结构中每一个元素进行一次提取，一般在循环次数已知的时候使用。语法形式如下：

```
for 循环变量 in 遍历结构:
        语句块
```

遍历循环可以认为是从遍历结构中逐一提取元素放在循环变量中，每提取一次元素则执行一次语句块。for 循环的执行次数取决于遍历结构的元素个数。遍历结构可以是字符串、文件、range()函数或者组合数据类型等。示例如下：

```
for c in 'Python':
    print(c)
```

还有一种遍历结构的扩展模式，具体形式如下：

```
for c in 'PY':
    print(c)
else:
    print('遍历结束')
```

(2) 无限循环。

无限循环使用保留字 while 根据给出的指定条件来判断执行程序。无限循环可以在循环次数不确定的时候使用。语法形式如下：

```
while 判断条件:
    语句块
```

当判断条件语句的值为 True 时，就执行循环体中的语句；如果判断条件语句的值为 False，那么就会结束整个循环，然后执行与 while 同一缩进级别的后续语句。具体示例如下：

```
n = 0
while n < 10:
    print(n)
    n += 3
```

与遍历循环相似，无限循环也有一种使用 else 的扩展模式，使用扩展模式修改上述代码如下：

```
n = 0
while n < 10:
    print(n)
    n += 3
else:
    print('循环正常结束')
```

需要注意的是，while 循环的扩展结构只有在循环正常结束的情况下才会执行 else 中的语句块。如果是因为执行了 break 语句而导致了循环提前结束，那么 else 中的语句块将不会被执行。

3. 循环控制

在循环结构中有两个可以辅助循环控制的保留字：一个是 break，还有一个是 continue。无论是在遍历循环还是无限循环中都可以使用这两个保留字，而且在大部分时候都会跟分支结构或者异常处理结构一起使用。当程序执行 break 语句时，就会从现在的这个循环结构体中跳出来，整个循环结束，然后执行与循环体同一缩进级别的其他语句。当程序执行 continue 语句时，仅跳出当前的一次循环，从而提前进入到这个循环体的下一次循环，不跳出循环结构体。

3.1.5 函数

在编写程序的时候，经常会遇到这样的情况：需要用代码解决同样的或者是类似的问题，这些问题的不同之处仅在于需要处理的数据不一样。在还没有学习函数的内容时，大部分人可能会想要把之前的代码进行复制，然后粘贴到代码中需要的位置，并且做出适当的修改。如果按这种方法操作，就会让程序在不同的代码位置重复地执行这些相似的代码块，这样会显得很冗余，并且效率也不会很高。尤其是时间久了之后，对程序的一些功能有了新的需求，假如之前使用了复制代码的方法来实现代码的复用，此时就需要依次寻找之前粘贴的代码位置，然后逐一进行修改。这么做不仅耗时耗力，而且随着时间的增长，代码的数量也在扩大，代码的关系也不像之前那样单纯，在很大程度上来说，即使修改了那些复制的代码，也会存在漏洞。一般来说，不推荐使用直接复制代码的方式来实现代码的复用。那么如何解决这个问题呢？

解决这个问题最好的方法就是设计函数和类。在前面的学习中，已经接触了很多

Python 中的内置函数，除了可以使用这些内置函数以外，用户还可以自己设计编写函数，就是把一段有规律的、可重复使用的代码定义成函数，从而达到一次编写、多次调用的目的。把需要重复执行的代码封装成函数，然后在需要这个功能模块的地方调用封装好的函数，这样不仅可以很好地实现代码的复用，而且还能够使代码的一致性得到保证。

所谓封装，就是把对象的属性和具体实现的细节隐藏起来，仅仅对外公开接口，控制在程序中属性的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机结合，形成“类”，其中数据和函数都是类的成员。

在设计编写函数的时候也需要遵循以下原则：第一，一个函数中不要实现过多的功能，最好在一个函数中只实现一个大小合适的功能；第二，降低全局变量的使用频率，使得函数之间只通过调用和参数传递来显示体现它们之间的关系；第三，在实际项目开发中为了方便管理，会把一些通用的函数封装到一个模块里，并且把这个通用模块文件放到顶层文件中。

1. 函数的定义

在 Python 中定义函数通常使用 def 关键字作为开头，然后输入一个空格，再输入函数的名字和一对圆括号，最后还要加上一个冒号，然后换行编写函数体。圆括号中放入的是形式参数（形参），如果定义的这个函数拥有多个参数，那么就需要用逗号把它们隔开。具体语法形式如下：

```
def functionname(parameters):
    function_suite
    return [expression]
```

function_suite 代表的是具体的功能代码，return [expression]表示函数结束，然后选择性地返回一个值给调用方。如果 return 后面没有加表达式，那么就相当于返回一个空值。

定义函数时，需要注意以下 4 点：

（1）定义时不需要声明圆括号里的形参类型，也不需要给函数指定一个返回值的类型。

（2）如果定义的函数中没有参数，那么圆括号是不可以省略的，一定要保留。

（3）圆括号的后面一定要加上冒号。

（4）函数体相对于 def 关键字必须保持一定的缩进。

在编写函数时，为了增加可读性，一般会为其加上适当的注释。

2. 函数的使用

Python 中可以对函数进行嵌套定义。所谓嵌套，就是在一个函数中再对另一个函数进行定义。一般来说，函数的作用域和变量生存周期都不会因为嵌套发生任何改变。作用域指函数可以被看见的范围。对函数进行嵌套定义时，需要注意内部函数不可以被外部直接调用，如果直接调用会抛出 NameError 异常。

虽然函数嵌套定义有很多优点，使用起来也很方便，但是通常来说不建议在代码中使用过多的函数嵌套，原因是过多的函数嵌套会使得内部的函数被反复定义，从而导致代码的执行效率变得低下。

Python 中的可调用对象一共有 7 种，用户自己定义的函数也属于其中的一种。其他 6 种形式分别是内置函数、内置方法、方法（定义在类中的函数）、类、类实例（如果类定义了

__call__方法，那么它的实例就可以作为函数进行调用）和 Generator 函数（使用 yield 关键字的函数或方法）。

嵌套函数不仅使得编写代码更加方便，还有另外一个非常重要的应用，即修饰器。从本质上来说，修饰器也是一个函数，只是这个函数的功能有一点特殊。修饰器会把其他函数作为参数使用，然后在一定程度上改造这些函数，最后返回一个新的函数。总体来说，修饰器的作用在于可以让已经存在的对象用于其他额外的功能。在 Python 中，静态方法、类方法、属性等都可以通过修饰器实现。

Python 中有三个内置的修饰器：①staticmethod，作用是把类里面定义的实例方法变成静态方法；②classmethod，作用是把类里面的实例方法变成类方法；③property，作用是把类里面定义的实例方法变成类属性。一般来说，静态方法和类方法的使用频率都不是很高，因为用户可以自己在模块里定义函数。

3. 函数的参数

定义函数的时候会使用一对圆括号，圆括号中的内容就是函数将要使用到的形参列表。形参的意思就是形式上的参数，就跟数学上的 x 一样，它没有具体的值，一般都是人为地来给它赋值，在赋值之前它是没有任何意义的。相对应地，Python 中还有一个实际参数（实参），就是一个已经确定下来的数，可以是数字或者是字符串等。当调用函数的时候，就会把一个实际的参数传给圆括号中的形参，这样形参就变成了实参，执行函数体内容的时候就会执行相应的操作。

函数可以有很多个参数，这些参数在写进圆括号的时候需要用逗号分隔开。函数也可以没有参数，但是没有参数的时候，圆括号也是不可以省略的。定义函数时不需要再对参数进行声明，因为 Python 里的解释器会自动根据实参的类型推断出形参的类型。需要格外注意的是，圆括号里形参的数量要跟调用函数时传入实参的数量一样，而且圆括号里的形参的顺序也要和传入的实参的顺序一样。

一般来说，当在函数内部直接对形参的值进行一些修改时，并不会对实参产生任何影响，实参不会发生改变。例如：

```
def f(a):
    a+=1
```

这时，就会得到一个新的变量 a。

```
a=2
f(a)
```

这个时候输出结果为 a=2。

上面的例子在函数内部对形参 a 的值作出了修改，但是当这段代码运行结束时，实参 a 的值没有因为形参的改变而发生任何改变。当函数参数是列表、字典这些可变序列类型时，如果在函数内部通过列表或字典对象自身拥有的方法修改参数中的元素时，实参也会发生改变。

（1）位置参数

在定义函数时最常用的形式就是位置参数，调用函数时实参的顺序必须跟顺序参数的相同，而且这两个参数的数量也是必须相同的。