Python 基础教程

杨志宏
西北民族大学

目　　录

1 Python 简介 **1**

1.1 Python 发展历史 1
1.2 Python 特点 1
1.3 使用 Python 的知名项目 2
1.4 搭建 Python 开发环境 2

2 Python 核心语法 **4**

2.1 Python 中的变量 4
2.2 数字类型 6
2.3 序列 9
2.4 字符串 11
2.5 列表 15
2.6 字典 18
2.7 元组 21
2.8 控制声明 21
2.9 函数 23
2.10 循环 27

3 Python 中的面向对象编程 **30**

3.1 Python 对象和类 30
3.2 操作符重载 32
3.3 继承和多态 33

4 异常处理 **36**

4.1 捕获异常 36

5 模块 **39**

5.1 创建模块 39
5.2 使用模块中的指定内容 39
5.3 dir 函数 39
5.4 包 40

A Python 关键字 **41**

第 1 章 Python 简介

1.1 Python 发展历史

Python 的创始人为 Guido van Rossum。1989 年圣诞节期间，在阿姆斯特丹，Guido 为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，做为 ABC 语言的一种继承。之所以选中 Python（大蟒蛇的意思）作为程序的名字，是因为他是一个叫 Monty Python 的喜剧团体的爱好者。ABC 是由 Guido 参加设计的一种教学语言。就 Guido 本人看来，ABC 这种语言非常优美和强大，是专门为非专业程序员设计的。但是 ABC 语言并没有成功，究其原因，Guido 认为是非开放造成的。Guido 决心在 Python 中避免这一错误。同时，他还想实现在 ABC 中闪现过但未曾实现的东西。

截至目前，Python 的版本为 3.5.1，2015 年 9 月 13 日发布。[**?**]

1.2 Python 特点

简单 Python 是一种代表简单主义思想的语言。阅读一个良好的 Python 程序就感觉像是在读英语一样。它使你能够专注于解决问题而不是去搞明白语言本身。

易学 Python 很容易上手，一方面是由于 Python 有完善的说明文档，另一方面网络中有大量的教程，学习资源可谓丰富。本书的写作就参考了诸多网络教程。[**?**]

开源 开源意味着人们可以自由地发布这个软件的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。Python 有非常活跃的开源社区，来自世界各地的程序员不断完善着 Python，如今 Python 拥有功能强大且门类齐全的扩展库。它可以帮助处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统、GUI（图形用户界面）、Tk 和其他与系统有关的操作。Python 语言及其众多的扩展库所构成的开发环境十分适合工程技术、科研人员处理实验数据、制作图表，甚至开发科学计算应用程序。

解释性 在计算机内部，Python 解释器把源代码转换成称为字节码的中间形式，然后再把它翻译成计算机使用的机器语言并运行。这使得使用 Python 更加简单。也使得 Python 程序更加易于移植。

可移植 Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。这些平台包括 Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acom RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE、PocketPC、Symbian 以及 Google 基于 linux 开发的 android 平台。

面向对象 Python 既支持面向过程的编程也支持面向对象的编程。在“面向过程”的语言中，程序是由过程或仅仅是可重用代码的函数构建起来的。在“面向对象”的语言中，程序是由数据和功能组合而成的对象构建起来的。

可扩展 如果需要一段关键代码运行得更快或者希望某些算法不公开，可以部分程序用 C 或 C++ 编写，然后在 Python 程序中使用它们。

可嵌入 可以把 Python 嵌入 C/C++ 程序，从而向程序用户提供脚本功能。

1.3 使用 Python 的知名项目

以下是使用 Python 作为主力开发语言的知名项目，其中有一些是用 python 进行开发，有一些在部分业务或功能上使用到了 python，还有的是支持 python 作为扩展脚本语言。

Reddit 社交分享网站，最早用 Lisp 开发，在 2005 年转为 python。

Dropbox 文件分享服务。

豆瓣网 图书、唱片、电影等文化产品的资料数据库网站。

Django 鼓励快速开发的 Web 应用框架。

EVE 网络游戏 EVE 大量使用 Python 进行开发。

Fabric 用于管理成百上千台 Linux 主机的程序库。

Blender 以 C 与 Python 开发的开源 3D 绘图软件。

BitTorrent bt 下载软件客户端。

Ubuntu Software Center Ubuntu 9.10 版本后自带的图形化包管理器。

YUM 用于 RPM 兼容的 Linux 系统上的包管理器。

Civilization IV 游戏《文明 4》。

Battlefield 2 游戏《战地 2》。

Google 谷歌在很多项目中用 python 作为网络应用的后端，如 Google Groups、Gmail、Google Maps。

NASA 美国宇航局，从 1994 年起把 python 作为主要开发语言。

Industrial Light & Magic 工业光魔，乔治 · 卢卡斯创立的电影特效公司。

Yahoo Groups 雅虎推出的群组交流平台。

YouTube 视频分享网站，在某些功能上使用到 python。

Cinema 4D 一套整合 3D 模型、动画与绘图的高级三维绘图软件，以其高速的运算和强大的渲染插件著称。

Autodesk Maya 3D 建模软件，支持 python 作为脚本语言。

gedit Linux 平台的文本编辑器。

GIMP Linux 平台的图像处理软件。

Minecraft: Pi Edition 游戏《Minecraft》的树莓派版本。

MySQL Workbench 可视化数据库管理工具。

Digg 社交新闻分享网站。

Mozilla 为支持和领导开源的 Mozilla 项目而设立的一个非营利组织。

Quora 社交问答网站。

Path 私密社交应用。

Pinterest 图片社交分享网站。

SlideShare 幻灯片存储、展示、分享的网站。

Yelp 美国商户点评网站。

Slide 社交游戏/应用开发公司，被谷歌收购。

1.4 搭建 Python 开发环境

Python 支持多个平台，其中在 Mac、类 UNIX 平台中已默认安装，Windows 平台中的安装也非常简单，从官方网站下载安装包安装即可，注意安装时将 Python 所在目录添加到系统路径中即可。

虽然 Python 自带编辑器，但其不够方便，推荐使用轻量级的编辑器 Sublime Text。使用编辑器将文件保存成.py 后缀，然后通过命令行调用即可执行，也可以在 Sublime Text 编辑器中使用编译命令（ctrl+b）查看运行结果。

如在编辑器中键入如下内容：

```
print ('Hello world!')
```

保存为 hello.py，注意设置文件编码方式为 UTF-8，然后使用 ctrl+b 即可在编辑器内部查看运行结果。

或者通过命令行进入到脚本所在路径，键入脚本名称（后缀名可省略）也可运行脚本。

```
cd c:/wamp/www/python/example
hello
```

第 2 章 Python 核心语法

2.1 Python 中的变量

2.1.1 注释

在 Python 中，使用“#”标记注释。注释不会被 Python 解释器执行。注释是开发人员用来提醒自己或他人程序如何工作的重要手段，注释还会用在文档的写作中。

```
#display hello world
print("hello␣world")
```

上述代码将会打印出 hello world 字符串。

注脚 1 *物理行与逻辑行*

所谓物理行（Physical Line）是你在编写程序时你所看到的内容。所谓逻辑行（Logical Line）是 Python 所看到的单个语句。Python 会假定每一物理行会对应一个逻辑行。有关逻辑行的一个例子是诸如 print('hello world') 这样一句语句——如果其本身是一行（正如你在编辑器里所看到的那样），那么它也对应着一行物理行。

Python 之中暗含这样一种期望：Python 鼓励每一行使用一句独立语句从而使得代码更加可读。

2.1.2 变量命名

变量（Vaiable）实质上是对内存中地址的命名，在内存中存储着诸多对象，为了方便使用这些对象，便有了变量。把变量和函数的名称我们叫作标识符（Identifier）。在 Python 中，标识符必须遵守以下规则：

1. 所有标识符必须以字母或者下划线 (_) 开头，不能以数字开头。如 my_var 就是一个有效的标识符，而 1digit 就不是。
2. 标识符可以包含字母、数字和下划线。标识符不限长度。
3. 标识符不能是关键字（所谓关键字，就是 Python 中已经使用并有特定含义的单词）。Python 的关键字参见附录A。

2.1.3 变量赋值

值（Value）是程序运行过程中的基本元素之一，例如 1，3.14，"hello" 等等都是值。在编程属于中，它们又被叫作字面量（literals）。字面量拥有不同的类型，如 1 是整型（int），3.14 是浮点型（float），"hello" 是字符串（string）。在之后的章节中，我们将详细学习数据类型。

在 Python 中，无需声明变量类型，解释器会根据变量的值自动判断变量类型。使用等于号为变量赋值，等于号也被认为赋值操作符（operator）。以下是变量声明的一些例子：

```
x = 100                    # x 是整型
pi = 3.14                  # pi 是浮点类型
empname = "python␣is␣great" # empname 是字符串
a = b = c = 100            # 将100赋值给a、b、c
```

注意，变量 x 中并不储存 100 自身，它存储的是 100（它是一个整型对象）的引用（reference）地址。

2.1.4 同步赋值

Python 可以使用以下语法对多个变量同步赋值：

```
var1, var2, ..., varn = exp1, exp2, ..., expn
```

上述声明告诉 Python，将表达式右边的值依次赋值给表达式左侧的变量。同步赋值在要交换两个变量的值时非常有用。例如：

```
>>> x = 1
>>> y = 2

>>> y, x = x, y # 交换x、y的值

>>> print(x)
2
>>> print(y)
1
```

注脚 2 *>>> 是 Python 交互模式中的提示符。*

2.1.5 数据类型

Python 拥有 6 种标准数据类型。

1. Numbers，数字
2. String，字符串
3. List，列表
4. Tuple，元组
5. Dictionary，字典
6. Boolean，布尔值

2.1.6 从控制台中接受输入值

input() 函数用来接受从控制台输入的值。它的用法如下：

```
input([prompt]) -> string
```

注脚 3 *使用 input() 函数和用户进行交互 input() 函数接受一个可选的字符串变量，用以提示输入内容，该函数返回值为字符串。*

例如：

```
>>> name = input("Enter␣your␣name:␣")
>>> Enter your name: tim
>>> name
'tim'
```

2.1.7 引入模块

Python 使用模块（module）组织代码。Python 内置了许多常用的模块，比如 math 模块用来处理数学运算，re 模块用来处理正则表达式等等。但是在使用这些模块之前，你需要使用以下语法引入这些模块：

```
import module_name
```

你还可以使用以下语法导入多个模块：

```
import module_name_1, module_name_2
```

例如：

```
>>> import math
>>> math.pi
3.141592653589793
```

上述代码中的第一行引入了 math 模块，这样我们就可以使用 math 模块中的所有函数、类、变量和常量。为了使用 math 模块中的这些内容，我们需要在模块名称后使用 (.)，然后就可以使用模块中定义的类、函数、常量或者变量了。在上面的例子中，math.pi 表示 math 模块中的 pi 常量。

2.2 数字类型

Python 3 支持 3 种不同类型的数字类型。

int 整型数字，比如 2015。

float 浮点型数字，比如 3.14。

complex 复数，比如 3+2j。

2.2.1 查看变量类型

Python 使用内置函数 type() 来查看变量的类型。在 Python 中，内置了一些高效强大的对象类型，使得开发人员不用从零开始进行编程。实际上，Python 中的每样东西都是对象。虽然 Python 中没有类型声明，但表达式的语法决定了创建和使用的对象的类型。一旦创建了一个对象，它就和操作集合绑定了，这就是所谓的动态类型和强类型语言。即 Python 自动根据表达式创建类型，一旦创建成功，只能对一个对象进行适合该类型的有效操作。[**?**]

```
>>> x = 12
>>> type(x)
 <class 'int'>
```

2.2.2 整型

整型（int）字面量在 Python 中属于 int 类。

```
>>> i = 100
>>> i
100
```

数字可以进行各种运算，如：

```
123 + 345
```

还可以使用数学模块进行更高级的运算，如产生随机数等等：

```
import random
print(random.random())
```

import 表示引入模块，import random 就是引入随机数模块。

2.2.3 浮点类型

浮点数（float）是指有小数点的数字。

```
>>> f = 12.3
>>> type(f)
<class 'float'>
```

2.2.4 复数

复数（Complex number）由实数和虚数两部分构成，虚数用 j 表示。我们可以这样定义一个复数：

```
>>> x = 2+3j
>>> type(x)
<class 'complex'>
```

2.2.5 运算符

Python 有各种运算符，我们可以使用这些运算符完成计算。运算符见下表2.1：

表 2.1: Python 常用数字运算符

名称	含义	例子	运行结果
+	加	3+1	4
-	减	40-2	38
*	乘	3*2	6
/	除	6/3	2
//	取整除	3//2	1
**	幂	2**3	8
%	求余数	7%2	1

2.2.6 运算符的优先级别

Python 按照运算符的有限级别计算表达式的值，比如：

```
>>> 3 * 4 + 1
```

在上面的表达式中，应该先进行加运算还是乘运算？为了搞清楚这个问题，我们需要明白 Python 中运算符的优先级别，表2.2显示了运算符的优先级别，依次从高到底排列如下：

在上表中我们可以看到，乘法运算的级别高于加法，因此，先进行乘法运算，再进行加法运算，最后的计算结果为 13。

表 2.2: 运算符的优先级别

运算符	描述
'expression,...'	字符串转换
{key:datum,...}	字典显示
[expression,...]	列表显示
()	分组
f(args...)	函数调用
x[index:index]	列表切分
x[index]	元素下标
x.attr	调用对象属性
**	指数运算
^x	按位取反
+x,-x	正负号
*，/，%	乘、除、取余数
+，-	加，减
<<，»	逐位左移，逐位右移
&	逐位求和
^	逐位异或
|	逐位或
<,<=,>,>=,<>,!=,==	比较
is,not is	同一性测试
in,not in	成员资格判断
not x	布尔“非”
and	布尔“并”
or	布尔“或”
lambda	Lamada 表达式

```
>>> 3 * 4 + 1
>>> 13
```

让我们再看下面的例子，以便演示优先顺序的另一个问题：

```
>>> 3 + 4 - 2
```

上述表达式到底先进行加法运算还是减法呢？因为在表2.2中我们看到加减运算的优先级别相同。当优先级别相同时，表达式从左向右计算，也就是说，上述的例子将先进行加法运算，再进行减法运算。

```
>>> 3 + 4 - 2
>>> 5
```

同级别运算符从左到右运算，这条规则有个例外，那就是赋值运算 (=)，赋值运算是从右向左计算的。例如：

```
a = b = c
```

先将 c 的值，赋给 b，再将 b 的值赋给 a。

2.2.7 增强赋值运算符

增强赋值运算符能简化赋值声明语句，例如：

```
>>> count = 1
>>> count = count + 1
```

```
>>> count
2
```

使用增强赋值运算符，我们可以将上述代码变为：

```
>>> count = 1
>>> count += 1
>>> count
2
```

类似的增强赋值运算符，除了 += 外，还有-=，%=，//= , /= , *= , **=。

2.3 序列

序列（Sequence）是一个包含其他对象的有序集合，序列中的元素包含了一个从左到右的顺序，可以根据元素所在的位置进行存储和读取。Python 中内建了 6 种序列，分别是列表、元组、字符串、unicode 字符串、buffer 对象和 xrange 对象。

序列作为 Python 的数据结构，有一些操作是通用的，如：索引、分片、加、乘以及检查某个成员是否属于序列的成员（成员资格），另外，还有一些计算长度、找到最大元素等等的内建函数。

2.3.1 索引

序列中的所有元素都有编号，从 0 开始，可以按照编号来访问序列中的元素，这个标号就是索引 (indexing)。

```
se = 'Hello'
print(se[0])
print(se[-1])
```

se[0] 表示序列 se 中的第一个元素，se[-1] 表示序列中的最后一个元素。

2.3.2 分片

分片（Slicing）操作指的是访问序列中一定范围之内的元素。分片通过冒号相隔的两个索引来实现，第一个索引是需要提取部分的第 1 个元素的编号，而第二个索引是分片之后剩下部分的第 1 个元素的编号，第二个索引不包含在分片之中：

```
se = 'Hello␣Pythoner! '
print(se[0:5])
```

上述代码将打印出‘Hello’字符串。但有时，我们需要获取序列的后面几个元素，同时，序列的大小是未知的，我们可以这样写：

```
se = 'Hello␣Pythoner! '
print(se[-9:])
```

se[-9:] 中空了第 2 个索引，表示一直到最后一个元素。上述代码将打印出‘Pythoner!’字符串。

进行分片时，分片的开始和结束点需要指定。而另外一个参数步长（step length）通常默认为 1，当有必要时，可是指定切片的步长，如每隔 1 个元素就取出元素：

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(numbers[0:10:2])
print(numbers[1::2])
```

上述代码将打印出‘[1, 3, 5, 7, 9]’和‘[2, 4, 6, 8, 10]’，其中的步长都是 2。当然步长也可以设置为负值，这样分片会从后往前进行。

2.3.3 序列相加

可以通过加号能对两个相同类型的序列进行连接运算，如字符串：

```
hello = '你好'
name = 'yangjh'
print(hello + name)
```

上述代码将打印出‘你好 yangjh’字符串。

2.3.4 序列相乘

序列乘以数字，表示将原有序列重复若干次：

```
hello = '你好'
print(hello * 3)
```

上述代码将打印出‘你好你好你好’字符串。

空列表可以使用‘[]’来表示，但是，如果想创建有 10 个空元素组成的列表，就需要使用 None，None 是 Python 内建的一个值，表示什么都没有，因此，要创建含有 10 个空元素的列表，就可以这样：

```
print([None] * 10)
```

2.3.5 成员资格

使用 in 运算符，可以检查某个元素是否存在与指定的序列中。如果元素存在于序列中，则返回 True，否则返回 False。

```
print('张三' in ['张三', '李四', '王二'])
```

上述代码将打印出布尔值 True。

2.3.6 长度、最小值、最大值

注脚 4 使用 *dir()* 函数输出对象的内置方法 *dir()* 函数可以输出对象的内置方法。如：*dir('str')* 就可以打印出所有字符串对象的内置方法。

内建函数 len() 可以返回序列的大小，如：

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(len(numbers))
print(max(numbers))
print(min(numbers))
```

上述代码将打印出 numbers 序列的长度‘10’和最大值以及最小值。

2.4 字符串

Python 中的字符串（Strings）是用单引号或双引号标记的一系列连续字符（characters），换句话说，字符串是由单个字符组成的序列（list）。即便只有一个字符，也是字符串，Python 中没有字符数据类型。记住单引号括起的字符串和双引号括起的字符串是一样的——它们不存在任何区别。

2.4.1 创建字符串

```
>>> name = "tom"
>>> mychar = 'a'
```

我们还可以使用下面的语法创建字符串：

```
>>> name1 = str() # 创建一个空字符串
>>> name2 = str("newstring") # 创建一个内容为newstring的字符串
```

2.4.2 字符串的不可变性

在 Python 中，每一个对象都可以分为不可变性或者可变性。在核心类型中，数字、字符串和元组是不可变的。

字符串在 Python 中一旦创建就不能就地改变，例如不能通过对其某一位置进行赋值而改变字符串。下面的语句就会出现如下语法错误：“TypeError: 'str' object does not support item assignment”。

```
s = 'string'
print(len(s))
print(s[0])       # 输出序列的第一个元素
s[0] = 'another␣s'  # 试图修改字符串的内容
print(s)
```

关于不可变性，我们再看一个例子：

```
>>> str1 = "welcome"
>>> str2 = "welcome"
```

上述代码中，str1 和 str2 都指向存储在内存中某个地方的字符串对象"welcome"。我们可以通过 id() 函数来测试 str1 和 str2 是否真的指向同一个对象。

注脚 5 *id() 函数 id() 函数可以得到对象在内存中的存储地址。*

如下：

```
>>> str1 = 'welcome'
>>> str2 = 'welcome'
>>> id(str1)
35462112
>>> id(str2)
35462112
```

我们可以看到，str1 和 str2 都指向同一个内存地址，因此，他们都指向同样的对象 “welcome”。下面让我们再编辑 str1 的内容看看：

```
>>> str1 += "␣yangjh"
>>> str1
'welcome␣yangjh'
>>> id(str1)
35487600
```

我们可以看到，现在变量 str1 指向了一个完全不同的内容地址，这也说明，我们对 str1 的内容操作实际上是新建了一个新的字符串对象。

2.4.3 字符串操作

字符串索引开始于 0，因此，我们可以这样获取字符串的第一个字符：

```
>>> name = 'yangjh'
>>> name[0]
'y'
```

在对字符串操作时，还可以从后往前取元素：

```
>>> name[-1]
'h'
```

运算符 “+” 用来连接字符串，运算符 “*” 用来重复字符串，例如：

```
>>> s = "tom␣and␣" + "jerry"
>>> print(s)
tom and jerry
>>> s = "love␣" * 3
>>> print(s)
love love love
```

2.4.4 字符串分片

我们还可以通过 “[]” 操作符来获取原始字符串的子集，这就是所谓的分片。语法规则如下：

```
s[start:end]
```

切分操作将返回字符串的部分内容，起始于 index，结束于 end-1。例如：

```
>>> s = 'yangjh'
>>> s[1:3]
'an'
>>> s = "Welcome"
>>> s[ : 6]
'Welcom'
>>> s[4 : ]
'ome'
>>> s[1 : -1]
'elcom'
```

注意：开始索引和结束索引都是可选的，如果忽略，开始索引就是 0，而结束索引就是字符串的最后一个字符对应的索引值。

2.4.5 in 和 not in 操作符

我们可以使用 in 和 not in 操作符检查一个字符串是否存在于另一个字符串，in 和 not in 就是所谓的成员资格操作符（membership operator）。

```
>>> s1 = "Welcome"
>>> "come" in s1
True
>>> "come" not in s1
False
```

2.4.6 String 对象的方法

下表2.3是三个常用的字符串方法：

表 2.3: 常用的字符串方法

方法名称	功能描述
len()	返回字符串长度
max()	返回字符串中 ASCII 编码值最大的字符
min()	返回字符串中 ASCII 编码值最小的字符

```
>>> len("hello")
5
>>> max("abc")
'c'
>>> min("abc")
'a'
```

2.4.7 比较字符串

我们可以使用 (> , < , <= , <= , == , !=) 比较两个字符串。Python 比较字符串是按照编纂字典的方式进行的，也就是使用ASCII 编码值[1]比较字符。

假设 str1 的值为"Jane"，str2 的值为"Jake"，首先比较这两个字符串的第一个字符 “J”，如果相等，就继续比较第二个字符（a 和 a），因为相同，所以继续比较第三个字符（n 和 k），因为 n 的 ASCII 编码值大于 k，因此 str1 大于 str2。更多例子参见下面的代码：

```
>>> "tim" == "tie"
False
>>> "free" != "freedom"
True
>>> "arrow" > "aron"
True
```

[1]美国信息交换标准码 (American Standard Code for Information Interchange) 是由美国国家标准学会 (American National Standard Institute,ANSI) 制定的单字节字符编码方案，供不同计算机在相互通信时用作共同遵守的西文字符编码标准，它已被国际标准化组织（ISO）定为国际标准，称为 ISO646 标准。

```
>>> "green" >= "glow"
True
>>> "green" < "glow"
False
>>> "green" <= "glow"
False
>>> "ab" <= "abc"
True
```

2.4.8 遍历字符串

字符串是序列，因此也可以使用循环遍历成员。

```
>>> s = "yangjh"
>>> for i in s:
...     print(i, end="")
...
yangjh
```

注脚 6 *改变 print() 函数的输出格式 print() 函数在默认状态下，会另起一行打印字符串，我们可以使用第二个参数修改结束标记。如 print("my string", end="") 就表示打印字符串，但不另起一行。*

2.4.9 字符串内容检验

Python 字符串类内置了丰富的方法，使用这些方法（见表2.4），我们可以检查字符串内容的类型。

表 2.4: 字符串内容类型检验方法

方法名称	方法说明
isalnum()	如果 string 包含字符都是字母或数字则返回 True
isalpha()	如果 string 包含字符都是字母则返回 True
isdigit()	如果 string 包含字符都是数字则返回 True
isidentifier()	判断字符串是否是合格的标识名
islower()	判断字符串中是否都是小写字母
isupper()	判断字符串中是否都是大写字母
isspace()	判断字符串是否由空格组成

这些判断方法的实例如下：

```
>>> s = "welcome␣to␣python"
>>> s.isalnum()
False
>>> "Welcome".isalpha()
True
>>> "2012".isdigit()
True
>>> "first␣Number".isidentifier()
False
```

```
>>> s.islower()
True
>>> "WELCOME".isupper()
True
>>> "␣␣\t".isspace()
True
```

2.4.10 在字符串内查找和替换

除了一般的序列操作，字符串还有独有的一些方法。如查找和替换：

```
print(s.find('in'))
print(s.replace('g', 'gs')) # 虽然显示字符串已被替换，但实际上是一个新的字符串。
```

相关的方法见下表2.5：

表 2.5: 查找子字符串

方法名称	方法说明
endswith(s1: str): bool	如果字符串以指定的字符串结尾，则返回真
startswith(s1: str): bool	如果字符串以指定的字符串开始，则返回真
count(substring): int	返回子字符串在字符串中出现的次数
find(s1): int	返回子字符串在字符串中第一次出现的索引，如果没有，则返回-1
rfind(s1): int	返回子字符串在字符串中最后一次出现的索引，如果没有，则返回-1

示例如下：

```
>>> s = "welcome␣to␣python"
>>> s.endswith("thon")
True
>>> s.startswith("good")
False
>>> s.find("come")
3
>>> s.find("become")
-1
>>> s.rfind("o")
15
>>> s.count("o")
3
```

2.5 列表

Python 的列表（list）对象是最常用的序列 (Sequence)。与字符串是不可变序列不同，列表是可变的。可通过对偏移量进行修改和读取。

2.5.1 列表赋值

列表可通过索引对其对应的元素进行赋值，从而改变列表的内容，如：

```
>>> a = [2, 2, 2]
>>> a[1] = 1
>>> print(a)
[2, 1, 2]
```

通过上述代码的运行，我们可以看到列表确实是可以改变的。

2.5.2 删除元素

使用 del 语句可以删除列表中的元素，如：

```
>>> a = [2, 2, 2]
>>> del a[1]
>>> print(a)
[2, 2]
```

2.5.3 分片赋值

分片赋值可以一次为多个元素赋值，并且不用考虑原列表的长度是否和新的列表长度一直，如：

```
>>> name = list('Python')
>>> print(name)
['P', 'y', 't', 'h', 'o', 'n']
>>> name[2:] = list('data')
>>> print(name)
['P', 'y', 'd', 'a', 't', 'a']
```

上述代码中的 list 函数是 Python 内置函数，其作用是将字符串转换为列表。运行结果显示，通过分片赋值，将原有列表 ['P', 'y', 't', 'h', 'o', 'n']，修改为 ['P', 'y', 'd', 'a', 't', 'a']。

分片赋值还可以用来插入元素，如：

```
>>> name = list('Python')
>>> name[1:1] = list('--')
>>> print(name)
['P', '-', '-', 'y', 't', 'h', 'o', 'n']
```

结果显示将原有列表 ['P', 'y', 't', 'h', 'o', 'n']，修改为 ['P', '-', '-', 'y', 'd', 'a', 't', 'a']。

2.5.4 列表对象常用内置方法

2.5.4.1 追加列表元素

。列表提供了在列表尾部追加新对象的方法 append。

```
>>> code = [1, 2, 3]
>>> code.append(4)
```

```
>>> print(code)
[1, 2, 3, 4]
```

2.5.4.2 计数

count 方法统计指定元素在列表中出现的次数，如：

```
>>> code = ['to', 'be', 'or', 'not', 'to', 'be']
>>> print(code.count('to'))
2
```

以上代码将统计出列表中‘to’元素出现的次数，结果为 2。

2.5.4.3 合并列表

extend 方法在列表的末尾一次性追加另一个序列中的多个值，如：

```
a = [1, 2, 3]
b = [4, 5, 6]
a.extend(b)
print(a)
```

以上代码将把 b 列表追加到 a 列表中，打印出的 a 列表的值为 [1, 2, 3, 4, 5, 6]。和序列加运算不同，extend 方法将改变原有列表的内容，而加运算却不会。例如：

```
b = [4, 5, 6]
b + [7, 8, 9]
print(b)
```

上述代码结果显示为 [4, 5, 6]，b 列表的内容并没有改变。

2.5.4.4 元素索引

index 方法用于从列表中找出指定值第一次匹配的索引值。例如：

```
a = [1, 2, 3, 3, 2, 1]
print(a.index(1))
```

以上代码运行结果为 0，即第一个 1 出现的索引为 0。

2.5.4.5 插入元素

insert 方法用于将对象插入到列表中，例如：

```
a = [1, 2, 3]
a.insert(2, 2.5)
print(a)
```

运行结果为 [1, 2, 2.5, 3]，insert 方法的两个参数值很好理解，第一个参数为在哪个元素后插入，表示位置，第二个参数为插入的内容。

2.5.4.6 pop

pop 方法会移除列表中的一个元素，默认为最后一个，和 append 方法刚好相反，并且返回该元素的值。例如：

```
a = [1, 2, 3]
print(a.pop())
print(a)
```

运行结果为 3 和 [1, 2]，当然，pop 方法也可以指定移除某个索引的元素。

2.5.4.7 remove

remove 方法用于移除列表中某个值的第一个匹配项：

```
code = ['to', 'be', 'or', 'not', 'to', 'be']
print(code.remove('or'))
print(code)
```

运行结果为 None 和 ['to', 'be', 'not', 'to', 'be']。这说明 remove 方法并不返回匹配到的内容。

2.5.4.8 reverse

reverse 方法将倒序排列列表元素：

```
a = [1, 2, 3]
a.reverse()
print(a)
```

运行结果为 [3, 2, 1]。

2.5.4.9 sort

sort 方法用于对列表排序，如：

```
a = [1, 3, 4, 8, 6, 2]
a.sort()
print(a)
```

运行结果为：[1, 2, 3, 4, 6, 8]。需要注意的是，sort 方法没有返回值，并且改变列表的内容，如果你不但要排序，而且还要保持原有数据的内容，解决的方法之一是将原有内容赋值到另外一个变量中保存。

2.6 字典

字典（Dictionary）是 Python 中的一种数据类型，用来存储键（key）值（value）对。字典数据能够使用键名快速取回、添加、删除、编辑值。字典和其他语言中的数组（array）或者哈希表（hash）非常相似。字典是可变（mutable）序列。[**?**]

2.6.1 创建字典

使用花括弧 () 就可创建字典。字典中的每一个项目都由键名、冒号 (:) 和值组成，多个项目之间用逗号 (,) 分割。让我们看一个实例：

```
friends = {
'tom'  : '66666666',
'jerry' : '88888888'
}
```

上面的变量 friends 是一个含有两个项目的字典。需要注意的一点是，键名必须是可哈希 [?] 类型，而值可以是任意类型。字典中的键名必须是唯一的。

2.6.2 获取、修改和添加字典元素

获取字典中的项目，使用如下语法：

```
dictionary_name['key']
```

例如：

```
>>> friends['tom']
'66666666'
```

如果字典中存在指定的键名，则返回对应的值，否则抛出键名异常。

添加和编辑项目，使用如下语法：

```
dictionary_name['newkey'] = 'newvalue'
```

例如：

```
>>> friends['bob'] = '99999999'
>>> friends
{'jerry': '88888888', 'bob': '99999999', 'tom': '66666666'}
```

删除字典中的项目使用如下语法：

```
del dictionary_name['key']
```

例如：

```
>>> del friends['bob']
>>> friends
{'tom': '66666666', 'jerry': '88888888'}
```

2.6.3 遍历字典

我们可以使用循环遍历字典中的所有项目。

```
>>> friends = {
...     'tom'  : '66666666',
...     'jerry': '88888888'
... }
```

```
>>> for key in friends:
...     print(key, ":", friends[key])
...
tom : 66666666
jerry : 88888888
```

2.6.4 字典比较

使用 == 和!= 操作符判断字典是否包含相同的项目。

```
>>> d1 = {"mike":41, "bob":3}
>>> d2 = {"bob":3, "mike":41}
>>> d1 == d2
True
>>> d1 != d2
False
>>>
```

注脚 7 不能使用其它的关系操作符 (<, >, >=, <=) 比较字典类型变量。

2.6.5 字典常用方法

Python 提供了多个内置的方法，用来操作字典，常用方法见下表2.6:

表 2.6: 字典常用方法

方法名	方法用途
popitem()	返回并移除字典中的任意项目
clear()	删除字典中的所有项目
keys()	以元组的形式获得字典的键名
values()	以元组的形式获得字典的值
get(key)	获得指定键名对应的值
pop(key)	移除指定键名的项目

```
>>> friends = {'tom': '111-222-333', 'bob': '888-999-666', 'jerry': '666-33-111'}

>>> friends.popitem()
('tom', '111-222-333')

>>> friends.clear()

>>>  friends
{}

>>> friends = {'tom': '111-222-333', 'bob': '888-999-666', 'jerry': '666-33-111'}

>>> friends.keys()
dict_keys(['tom', 'bob', 'jerry'])
```

```
>>> friends.values()
dict_values(['111-222-333', '888-999-666', '666-33-111'])

>>> friends.get('tom')
'111-222-333'

>>> friends.get('mike', 'Not Exists')
'Not Exists'

>>> friends.pop('bob')
'888-999-666'

>>> friends
{'tom': '111-222-333', 'jerry': '666-33-111'}
```

2.7 元组

在 Python 中，元组（Tuple）和列表非常相似，与列表不同的是，元组一旦创立，就不可改变，也就是说，元组是不可变的。

2.7.1 创建元组

```
>>> t1 = ()            # 创建一个空元组
>>> t2 = (11,22,33)    # 创建一个包含三个元素的元组
>>> t3 = tuple([1,2,3,4]) # 使用列表创建元组
>>> t4 = tuple("abc")   # 使用字符串创建元组
```

2.7.2 元组相关方法

元组也是序列，因此序列能使用的方法，如 max , min , len , sum 方法元组也能使用。

```
>>> t1 = (1, 12, 55, 12, 81)
>>> min(t1)
1
>>> max(t1)
81
>>> sum(t1)
161
>>> len(t1)
5
```

2.8 控制声明

在程序中，常常要根据一些条件执行相应的命令。

2.8.1 分支判断

Python 使用 if-else 进行控制声明。语法如下：

```
if boolean-expression:
   #statements
else:
   #statements
```

注脚 8 *在每一个 if 程序块中，必须使用相同数量的缩进，否则会产生语法错误。这是 Python 和其他语言非常不同的一点。*

现在我们看一个例子：

```
i = 11
if i % 2 == 0:
    print("偶数")
else:
    print("奇数")
```

运行结果将根据 i 的值发生变化。

如果需要判断多个条件，我们就可以使用 if-elif-else 控制声明，例如：

```
today = "monday"

if today == "monday":
   print("this␣is␣monday")
elif today == "tuesday":
   print("this␣is␣tuesday")
elif today == "wednesday":
   print("this␣is␣wednesday")
elif today == "thursday":
   print("this␣is␣thursday")
elif today == "friday":
   print("this␣is␣friday")
elif today == "saturday":
   print("this␣is␣saturday")
elif today == "sunday":
   print("this␣is␣sunday")
else:
   print("something␣else")
```

我们可以根据实际需求，添加对应的多个 elif 条件。

2.8.2 分支嵌套

我们可以在 if 声明语句块中嵌套使用 if 声明。例如：

```
today = "holiday"
bank_balance = 25000
```

```
if today == "holiday":
    if bank_balance > 20000:
        print("Go␣for␣shopping")
    else:
        print("Watch␣TV")
else:
    print("normal␣working␣day")
```

2.9 函数

函数是可重用的代码块，使用函数可以帮助我们组织代码的结构。我们创建函数的目的，是能在程序运行中多次使用一系列代码，而不用重复书写代码。

2.9.1 创建函数

Python 使用 def 关键词创建函数，语法如下：

```
def function_name(arg1, arg2, arg3, .... argN):
    #statement inside function
```

注脚 9 缩进 空白区在 *Python* 中十分重要。实际上，空白区在各行的开头非常重要。这被称作缩进（*Indentation*）。在逻辑行的开头留下空白区（使用空格或制表符）用以确定各逻辑行的缩进级别，而后者又可用于确定语句的分组。

这意味着放置在一起的语句必须拥有相同的缩进。每一组这样的语句被称为块（*block*）。有一件事你需要记住：错误的缩进可能会导致错误。

所有在函数内部的声明，都必须使用相等的缩进。函数可以没有参数，也可以有多个参数。多个参数之间用逗号隔开。还可以使用 *pass* 关键字忽略掉函数主题的声明。

我们看一个函数的例子，下面的函数将计算指定范围的整数之和：

```
def sum(start, end):
    result = 0
    for i in range(start, end + 1):
        result += i
    print(result)

sum(1, 10)
```

在上面的代码中，我们定义了一个叫作 sum() 的函数，该函数有两个参数（start 和 end），该函数将从 start 开始，累加到 end，最后打印出累积之和。代码运行的结果为 55。

2.9.2 函数返回值

上文定义的函数只是简单地在控制台打印出结果，如果我们想要将计算结果赋值给变量，以便做更深入的处理时应该怎么办？当我们遇到这种情况时，可使用 return 语句，将返回函数计算结果并且退出函数。例如：

```
def sumReturn(start, end):
    result = 0
    for i in range(start, end + 1):
        result += i
    return result

a = sumReturn(1, 5)
print(a)
```

在上面这段代码中，我们定义了有返回值的函数 sumReturn()，并将其结果赋值给变量 a。上面代码的运行结果为 15。

当然，return 语句也可以不返回值，而是用来退出函数（实际上会返回 None，为 NoneType 对象）。每一个函数都在其末尾隐含了一句 return None，除非你写了你自己的 return 语句。

```
def sum2(start, end):
    if(start > end):
        print("start␣should␣be␣less␣than␣end")
        return
    result = 0
    for i in range(start, end + 1):
        result += i
    return result

s = sum2(110, 50)
print(s, type(s))
```

上述代码的运行结果如下：

```
start should be less than end
None <class 'NoneType'>
```

在 Python 中，如果你不指定 return 的返回值，则会返回 None 值。

2.9.3 全局变量和局域变量

全局变量指的是不属于任何函数，但又可以在函数内外访问的变量。而局域变量指的是在函数内部声明的变量，局域变量只能在函数内部使用，无法在函数外访问（函数执行完后，会销毁内部定义的局部变量）。

下面我们通过例子来演示这两者的区别：

```
global_var = 12       # 定义全局变量

def func():
    local_var = 100   # 定义局部变量
    print(global_var) # 可以在函数内部访问全局变量

func()                # 调用函数func()

print(local_var)      # 无法访问变量local_var
```

上述代码将会出现错误：

```
NameError: name 'local_var' is not defined
```

我们再看一个例子：

```
xy = 100              # 定义全局变量xy

def func():
    xy = 200          # 定义局部变量xy
    print(xy)         # 此时访问的是局部变量xy

func()                # 调用函数func()
```

该代码显示的结果是 200，不是 100。

使用 global 关键字，可以将局部变量同全局变量绑定在一起。例如：

```
t = 1

def increment():
    global t   # 现在的变量t在函数内外都是一致的
    t = t + 1
    print(t)   # 输出 2

increment()
print(t)       # 输出 2
```

注脚 10 *使用 global 关键字声明全局变量时，无法直接赋值，比如“global t = 1”的写法存在语法错误。*

2.9.4 参数的默认值

为参数指定默认值，只需在定义函数时使用赋值语句即可。例如：

```
def func(i, j = 100):
    print(i, j)
```

上述定义的函数 func() 有两个参数 i 和 j。j 的默认值为 100，这意味着我们在调用这个函数的时候可以忽略掉 j 的值，比如 func(2)，运行结果为 2 100。

2.9.5 关键字参数

为函数传递参数值的方法有两种：位置参数和关键字参数。我们之前调用函数的时候都使用的是位置参数。下面我们看如何使用关键字参数：

```
def named_args(name, greeting):
    print(greeting + "␣" + name)

named_args(name='jim', greeting='Hello')
named_args(greeting='Hello', name='jim')
named_args('jim', greeting='hello')
```

上述代码运行结果都是“hello jim"。

关键字参数使用“name=value”的名称、值对传递数据，正如上面代码演示的那样，使用关键字参数的时候，参数的顺序是可以调换的，而且位置参数和关键字参数可以混合使用（只能先使用位置参数，后使用关键字参数）。

2.9.6 返回多个值

我们可以通过在 return 语句中使用逗号，将多个值返回，这种返回值的类型是元组。例如：

```
def bigger(a, b):
    if a > b:
        return a, b
    else:
        return b, a

s = bigger(12, 100)
print(s)
print(type(s))
```

运行结果为：

```
(100, 12)
<class 'tuple'>
```

2.9.7 函数文档字符串

Python 有一个甚是优美的功能称作文档字符串（Documentation Strings），在称呼它时通常会使用另一个短一些的名字 docstrings。DocStrings 是一款你应当使用的重要工具，它能够帮助你更好地记录程序并让其更加易于理解。令人惊叹的是，当程序实际运行时，我们甚至可以通过一个函数来获取文档！

```
def print_max(x, y):
    '''Prints the maximum of two numbers.

    The two values must be integers.'''
    # 如果可能，将其转换至整数类型
    x = int(x)
    y = int(y)

    if x > y:
        print(x, 'is␣maximum')
    else:
        print(y, 'is␣maximum')

print_max(3, 5)
print(print_max.__doc__)
输出：

$ python function_docstring.py
```

```
5 is maximum
Prints the maximum of two numbers.


    The two values must be integers.
```

该文档字符串所约定的是一串多行字符串，其中第一行以某一大写字母开始，以句号结束。第二行为空行，后跟的第三行开始是任何详细的解释说明。强烈建议你的文档字符串中都遵循这一约定。

我们可以通过使用函数的 __doc__（注意其中的双下划线）属性（属于函数的名称）来获取函数 print_max 的文档字符串属性。

2.10 循环

Python 只有两种循环：for 循环和 while 循环。

2.10.1 for 循环

for 循环语法：

```
for i in iterable_object:
   # do something
```

注脚 11 *所有在 for 循环或者 while 循环中的声明，必须使用相同的缩进值。否则会出现语法错误。*

我们看下面这段代码：

```
mylist = [1, 2, 3, 4]

for i in mylist:
    print(i)
```

在第一次循环时，值 1 被传递给 i，第二次循环时，值 2 被传递给 i。循环一直到列表变量 mylist 没有更多元素时停止。运行结果为：

```
1
2
3
4
```

2.10.2 范围循环

range() 函数能够指定循环的起始值和结束值，从而让循环体在指定的范围内循环。例如：

```
for i in range(10):
    print(i)        # 0-9
for i in range(1,10):
    print(i)        # 1-9
for i in range(1,10,2):
    print(i)        # 1,3,5,7
```

range() 函数只有 1 个参数时，表示从 0 开始循环；两个参数时，第一个参数是起始值，第二个参数是结束值；三个参数时，第三个参数表示循环步长。

2.10.3 while 循环

语法：

```
while condition:
    # do something
```

While 循环会一直执行循环体内部的声明，直到条件变成 false。每次循环都会检查判断条件，如果为真，就继续循环。例如：

```
count = 0

while count < 10:
    print(count)
    count += 1
```

这段代码将会打印出 0-9，直到 count 等于 10。

2.10.4 中断循环

使用 break 语句，可以中断循环，例如：

```
count = 0

while count < 10:
    count += 1
    if count == 5:
        break
    print("inside␣loop", count)

print("out␣of␣while␣loop")
```

运行结果为：

```
inside loop 1
inside loop 2
inside loop 3
inside loop 4
out of while loop
```

2.10.5 继续循环

当循环体内部出现 continue 声明时，会结束本次循环，跳转到循环体开始位置，开始下一次循环。例如：

```
count = 0

while count < 10:
```

```
    count += 1
    if count % 2 == 0:
        continue
    print(count)
```

运行结果将打印出 1，3，5，7，9。

第 3 章 Python 中的面向对象编程

面向对象程序设计（英语：Object-oriented programming，缩写：OOP）是一种程序设计范型，同时也是一种程序开发的方法。对象指的是类的实例。它将对象作为程序的基本单元，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性。[**?**]

面向对象程序设计可以看作一种在程序中包含各种独立而又互相调用的对象的思想，这与传统的思想刚好相反：传统的程序设计主张将程序看作一系列函数的集合，或者直接就是一系列对电脑下达的指令。面向对象程序设计中的每一个对象都应该能够接受数据、处理数据并将数据传达给其它对象，因此它们都可以被看作一个小型的“机器”，即对象。

目前已经被证实的是，面向对象程序设计推广了程序的灵活性和可维护性，并且在大型项目设计中广为应用。

3.1 Python 对象和类

3.1.1 创建类

Python 一门面向对象的语言。在 Python 中所有的东西都是对象，比如之前学习的整型、字符串等等，甚至模块、函数也都是对象。

面向对象编程时使用对象创建程序，使用对象存储数据和行为。

在 Python 中，使用关键字 class 定义类。类通常包括数据区域，用以存数数据和方法的定义。Python 中的所有类，都包含一个特殊的方法，叫作初始化（initializer），或者叫作构造方法。构造方法会在使用类创建新的对象时自动执行。例如：

```
class Person:


    # 构造函数
    def __init__(self, name):
        self.name = name


    # 定义方法
    def whoami(self):
        return "You␣are␣" + self.name
```

上述代码中，我们创建了一个名叫 Person 的类，这个类中包含数据字段 name 和方法 whoami()。

注脚 12 *What is self ?? Python 中的所有方法，包括构造方法，首个参数都是 self。这个参数指向对象本身。当我们创建一个新的对象时候，self 参数就会自动指向新创建的对象。*

3.1.2 从类中创建对象

使用类名就可创建对象。当我们调用方法时，不需要传递 self 参数，Python 会自动传递。例如：

```
p1 = Person('tom')
print(p1.whoami())
print(p1.name)
```

输出结果为：

```
You are tom
tom
```

我们还可以改变数据字段的值：

```
p1.name = 'jerry'
print(p1.name)
```

输出结果为 jerry。然而，像这样从类的外部获取数据字段，属于不太好的操作方式，下面我们看如何阻止这种操作。

3.1.3 隐藏数据字段

为了隐藏数据字段，我们需要定义私有数据字段。在 Python 中，使用两个前置下划线，就可定义私有数据字段和私有方法。比如：

```
class BankAccount:

    # 构造函数
    def __init__(self, name, money):
        self.__name = name # 定义私有数据字段
        self.__balance = money # 定义私有数据字段

    def deposit(self, money):
        self.__balance += money

    def withdraw(self, money):
        if self.__balance > money:
            self.__balance -= money
            return money
        else:
            return "Insufficient funds"

    def checkbalance(self):
        return self.__balance

b1 = BankAccount('tim', 400)
print(b1.withdraw(500))
b1.deposit(500)
print(b1.checkbalance())
print(b1.withdraw(800))
print(b1.checkbalance())
```

在上述代码中，我们定义了 BankAccout 类，这个类有两个数据字段，但都是私有字段。代码运行结果为：

```
Insufficient funds
900
```

```
800
100
```

现在，让我们尝试访问私有数据字段：

```
print(b1.__balance)
```

结果显示：

```
AttributeError: 'BankAccount' object has no attribute '__balance'
```

这就表明，设置为私有的数据字段，无法在类的外部访问。

3.2 操作符重载

我们之前已经看到 + 运算符不但能加数字，还能连接字符串。这之所以可能，是以为 + 运算符在 int 类和 str 类中都被重载。运算符实际上对应着类中相应的方法。为运算符定义方法就是所谓的运算符重载。比如，为让自定义对象能使用 + 运算符，我们需要定义名叫 __add__ 的方法。

让我们看个例子：

```
import math


class Circle:

    def __init__(self, radius):
        self.__radius = radius

    def setRadius(self, radius):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def area(self):
        return math.pi * self.__radius ** 2

    def __add__(self, another_circle):
        return Circle(self.__radius + another_circle.__radius)

c1 = Circle(4)
print(c1.getRadius())

c2 = Circle(5)
print(c2.getRadius())

c3 = c1 + c2 # 之所以能使用加法运算符，是因为我们定义了__add__方法
print(c3.getRadius())
```

在上面的例子中，我们为类添加了 __add__ 方法，该方法允许使用 + 运算符对两个 circle 对象求和。在 __add__ 方法中，我们创建了一个新的对象，并将其返回给调用者。运行结果如下：

```
4
5
9
```

在 Python 中，除 __add__ 方法对应 + 运算符之外，还有其他能够重载运算符的方法：如 __mul__、__sub__ 等等 [**?**]。

3.3 继承和多态

继承（inheritance）允许开发人员先创建一个通用的类，然后扩展为特定类。使用继承机制，我们可以获得类的数据字段和方法，还可以增加自定义的字段和方法，因此，继承提供了一种组织代码、重用代码的方式。

在面向对象的术语中，当类 X 继承自类 Y 时，Y 被叫做超类（super class）或基类（base class），而 X 被成为子类（subclass）或者衍生类（derived class）。

注脚 13 *私有数据字段和私有方法只在类的内部使用。子类只能继承父类的非私有数据字段和非私有方法。*

继承的语法如下：

```
class SubClass(SuperClass):
  # data fields
  # instance methods
```

让我们看个例子：

```
class Vehicle:

    def __init__(self, name, color):
        self.__name = name    # __name是私有数据字段
        self.__color = color

    def getColor(self):
        return self.__color

    def setColor(self, color):
        self.__color = color

    def getName(self):
        return self.__name


class Car(Vehicle):

    def __init__(self, name, color, model):
        # 调用父类的构造方法
```

```
        super().__init__(name, color)
        self.__model = model

    def getDescription(self):
        return self.getName() + self.__model + "␣in␣" + self.getColor() + "␣color"

c = Car("Ford␣Mustang", "red", "GT350")
print(c.getDescription())
print(c.getName())
```

上述代码中，我们创建了基类 Vehicle 和子类 Car。在子类 Car 中，我们没有定义 getName() 方法，但我们仍然可以访问 getName()，这是因为类 Car 继承自 Vehicle 类。在这段代码中，super() 方法用来调用基类的方法。上述代码的运行结果如下：

```
Ford MustangGT350 in red color
Ford Mustang
```

3.3.1 多重继承

不像 Java 和 C# 语言，Python 允许多重继承。即一次继承多个基类，比如：

```
class Subclass(SuperClass1, SuperClass2, ...):
   # initializer
   # methods
```

看如下实例：

```
class MySuperClass1():

    def method_super1(self):
        print("method_super1␣method␣called")


class MySuperClass2():

    def method_super2(self):
        print("method_super2␣method␣called")


class ChildClass(MySuperClass1, MySuperClass2):

    def child_method(self):
        print("child␣method")

c = ChildClass()
c.method_super1()
c.method_super2()
```

输出结果为：

```
method_super1 method called
method_super2 method called
```

因为子类 ChildClass 继承自 MySuperClass1 , MySuperClass2，因此，ChildClass 对象 c 可以访问 method_super1() 方法和 method_super2() 方法。

3.3.2 重写方法

为重写基类的某个方法，子类需要定义一个同名的方法（即拥有相同名称和相同数量的参数）。例如：

```
class A():

    def __init__(self):
        self.__x = 1

    def m1(self):
        print("m1␣from␣A")


class B(A):

    def __init__(self):
        self.__y = 1

    def m1(self):
        print("m1␣from␣B")

c = B()
c.m1()
```

在这段代码中，我们重写了基类的 m1() 方法。输出结果为：

```
m1 from B
```

3.3.3 判断对象是否属于某类

isinstance() 方法用来检测指定对象是否是某个类的实例。例如：

```
>>> isinstance(1, int)
True

>>> isinstance(1.2, int)
False

>>> isinstance([1,2,3,4], list)
True
```

第 4 章 异常处理

异常是指程序中的例外，违例情况。异常机制是指程序出现错误后，程序的处理方法。当出现错误后，程序的执行流程发生改变，程序的控制权转移到异常处理。[?]

4.1 捕获异常

异常处理可以使开发人员能以优雅的方式处理错误。

4.1.1 try-except

Python 使用 try-except 语句处理异常。语法如下：

```
try:
    # write some code
    # that might throw exception
except <ExceptionType>:
    # Exception handler, alert the user
```

在 try 语句块中，我们写入可能会产生异常的代码，当异常发生时，try 语句块中的代码会被忽略，转而进入 except 语句块中处理异常。例如：

```
try:
    f = open('somefile.txt', 'r')
    print(f.read())
    f.close()
except IOError:
    print('file␣not␣found')
```

上述代码的执行流程如下：

1. 先执行介于 try 和 except 之间的语句。
2. 如果没有异常，则 except 语句块中的代码会被跳过。
3. 如果文件不存在，则产生异常，在 try 语句块中的其他代码会被跳过。
4. 当异常发生时，如果异常类型和 except 关键字后的异常名称匹配，就执行 except 分支中的代码。上述代码中只能处理 IOError 异常，如要处理其他类型的异常，还需要添加更多的 except 分支。

4.1.2 多个 except

try 声明可以有多个 except 分支，它还可以选择 else 和 finally 分支。语法如下：

```
try:
    <body>
except <ExceptionType1>:
    <handler1>
except <ExceptionTypeN>:
    <handlerN>
except:
```

```
    <handlerExcept>
else:
    <process_else>
finally:
    <process_finally>
```

except 分支类似于 elif。当异常发生时，将检查 except 分支是否和异常类型匹配。如果匹配，就执行对应 except 分支中的代码。在最后一个 except 分支中，异常类型是被忽略了的。如果异常发生，但没有匹配到最后一个 except 之前的分支，则最后的 except 分支中的代码会被执行。如果没有任何异常发生，则执行 else 语句中的代码。finally 分支中的代码，无论是否有异常发生，都会被执行。例如：

```
try:
    num1, num2 = eval(input("Enter␣two␣numbers,␣separated␣by␣a␣comma␣:␣"))
    result = num1 / num2
    print("Result␣is", result)

except ZeroDivisionError:
    print("Division␣by␣zero␣is␣error␣!!")

except SyntaxError:
    print("Comma␣is␣missing.␣Enter␣numbers␣separated␣by␣comma␣like␣this␣1,␣2")

except:
    print("Wrong␣input")

else:
    print("No␣exceptions")

finally:
    print("This␣will␣execute␣no␣matter␣what")
```

注脚 14 *eval() eval() 函数允许在 Python 程序内部运行 Python 代码，了解更多关于 eval() 的信息，请访问 http://stackoverflow.com/questions/9383740/what-does-pythons-eval-do*

4.1.3 自定义异常

使用关键字 raise，可以在方法中自定义异常。语法为：

```
raise ExceptionClass("Your␣argument")
```

例如：

```
def enterage(age):
    if age < 0:
        raise ValueError("Only␣positive␣integers␣are␣allowed")

    if age % 2 == 0:
```

```
        print("age␣is␣even")
    else:
        print("age␣is␣odd")

try:
    num = int(input("Enter␣your␣age:␣"))
    enterage(num)

except ValueError:
    print("Only␣positive␣integers␣are␣allowed")
except:
    print("something␣is␣wrong")
```

当用户输入的年龄小于 0 时，程序显示结果为：

```
only positive integers are allowed
```

第 5 章 模块

Python 模块是一个包含有函数、变量、类和常量等等内容的 python 文件。[?] 模块帮助我们将相关的代码组织在一起，例如 math 模块拥有数学相关的函数。

5.1 创建模块

创建一个名为 mymodule.py 的新文件，并写入下面的代码：

```
foo = 100

def hello():
    print("i␣am␣from␣mymodule.py")
```

在这个文件中，我们定义了一个全部变量 foo 和一个名为 hello() 的方法。现在我们可以使用 import 关键词来引入这个模块，并使用 mymodule.py 中的变量和函数：

```
import mymodule

print(mymodule.foo)
mymodule.hello()
```

上述代码的运行结果如下：

```
100
i am from mymodule.py
```

如之前代码所示，调用模块的变量和函数时，需要指定模块的名称。

5.2 使用模块中的指定内容

当我们使用 import 声明导入模块时，模块中的所有内容都被导入到当前文件中。如果我们只需要模块中的个别内容时该如何操作呢？使用 from 关键词，就可以达到这样的目的，比如：

```
from mymodule import foo
print(foo)
```

上述代码的运行结果为 100。

注脚 15 当使用 *from improt* 语句导入特定内容后，访问这些内容就不需要再使用模块名了。

5.3 dir 函数

内置的 dir() 函数能够返回由对象所定义的名称列表。如果这一对象是一个模块，则该列表会包括函数内所定义的函数、类与变量。该函数接受参数。如果参数是模块名称，函数将返回这一指定模块的名称列表。如果没有提供参数，函数将返回当前模块的名称列表。

```
>>> import sys

# 给出 sys 模块中的属性名称
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
'version', 'version_info']
# only few entries shown here

# 给出当前模块的属性名称
>>> dir()
['__builtins__', '__doc__',
'__name__', '__package__']

# 创建一个新的变量 'a'
>>> a = 5

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a']
```

5.4 包

包是指一个包含模块与一个特殊的 __init__.py 文件的文件夹，后者向 Python 表明这一文件夹是特别的，因为其包含了 Python 模块。

假设你想创建一个名为“world”的包，其中还包含着”asia“、”africa“等其它子包，同时这些子包都包含了诸如”india“、”madagascar“等模块。下面是你会构建出的文件夹的结构：

```
- <some folder present in the sys.path>/
    - world/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - foo.py
        - africa/
            - __init__.py
            - madagascar/
                - __init__.py
                - bar.py
```

包是一种能够方便地分层组织模块的方式。

附录 Python 关键字

以下这些是 Python 中的关键字，这些关键字不能用于变量、函数名、过程和对象名等等标识符，完整列表可以通过如下命令得到：

```
help()
keywords
```

False、def、if、raise、None、del、import、return、True、elif、in、try、and、else、is、while、as、except、lambda、with、assert、finally、nonlocal、yield、break、for、not、class、from、or、continue、global、pass。