二级 Python 程序设计教程

人生苦短，我用 Python

版本：2020.2.10

作者：陈福明

内容简介

本书是全国计算机等级考试二级 Python 考试的学习精品。本书从 0 开始，由浅入深，涵盖了 Python 程序设计的所有基础知识，既适合 0 基础的学生学习，也适合有一定编程基础的学生学习，也可以作为从事科研的自学者的 Python 入门书籍。

前 言

人生苦短，我用 Python。由于 Python 语言的简洁性、易读性以及可扩展性，全球使用 Python 的个人和机构日益增多，一些知名大学已经采用 Python 来教授程序设计课程，例如卡耐基梅隆大学的编程基础、麻省理工学院的计算机科学及编程导论使用 Python 语言讲授。众多开源的软件包都提供了 Python 的调用接口，例如著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK、2D 游戏库 Cocos2d 和 3D 游戏库 Panda3d 等。而且中国教育部考试中心，也已经于 2018 年底把 Python 列入了全国计算机等级考试二级考试中，本书就是为此而编写的。

本书从 0 开始，由浅入深，涵盖了二级 Python 程序设计的所有知识点，既适合 0 基础的学生学习，也适合有一定编程基础的学生学习，也可以作为从事科研的自学者的 Python 入门书籍。

本书前 10 章包括概述、注释、变量、数字数据类型、选择语句、循环语句、异常处理、函数和模块、字符串、高级数据类型，除了用于二级 Python 外，也适合非信息类专业 32 课时的教学。11 章以后是进阶部分，适合非信息类专业 48 课时、64 课时或者信息类专业的学生有选择的教学。

本书编写过程中，中国电科院电网技术杂志社的编辑马晓华整理和校对了本书的初稿并对本书的编写提出了宝贵意见。此外其他一些人也对本书的架构、章节提出了宝贵的建议。这里一并表示衷心的感谢！

本书课后习题的答案、各章节的代码和课件都可以从 http://www.pythonlearning.com 网站（Python 学习网）中的二级 Python 导航页上找到并下载。该网站今后对本书还会进一步支持，具体进展，敬请关注 http://www.pythonlearning.com 网站。此外和本书配套的《七巧板二级 Python》微信小程序，有二级 Python 选择题的分类学习、智能学习和模拟学习供学生练习，今后准备增加其他题型的试题。

因为个人水平所限，书中难免存在一定的错误，作者衷心希望广大读者多提宝贵意见，我们将在后续的版本中百尺竿头，更进一步！

2020 年 2 月陈福明于北京市马甸桥南

目录

第 1 章 Python 概述

——认识 Python

知识点

(1) Python 的起源

(2) 为什么要用 Python？

(3) Python 的特点和优缺点

(4) Python的安装

1.1 人生苦短 我用 Python

人生苦短，我用 Python —— Life is short, you need Python

图1-1 人生苦短 我用python[1]

欢迎开始Python的学习，这本书带您进入简单而实用的Python世界，让编程变成一种类似于Word、Excell的日常工具，让你顺利通过二级Python考试。

1.2 Python 的起源

Python 的创始人为吉多·范·罗苏姆（Guido van Rossum）

图1-2 吉多·范·罗苏姆[1]

**[1] 图1-1和图1-2都来源与网络公开照片，在此对照片作者深表感谢。

(1) 1989 年的圣诞节期间，吉多·范·罗苏姆为了在阿姆斯特丹打发时间，决心开发一个新的**解释程序**，作为 ABC 语言的一种继承（**感觉下什么叫牛人**）

(2) ABC 是由吉多参加设计的一种教学语言，就吉多本人看来，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，吉多认为是**非开放**造成的。吉多决心在 Python 中避免这一错误，并获取了非常好的效果

(3) 之所以选中 Python（蟒蛇）作为程序的名字，是因为他是 BBC 电视剧——蒙提·派森的飞行马戏团（Monty Python's Flying Circus）的爱好者

(4) 1991 年，第一个 Python **解释器**诞生，它是用 C 语言实现的，并能够调用 C 语言的库文件

1.2.1 解释器

学习 Python 语言，首先得了解什么是解释器。**计算机不能直接理解任何除机器语言以外的语言**，所以必须要把程序员所写的程序语言翻译成机器语言，计算机才能执行程序。**将其他语言翻译成机器语言的工具，被称为编译器。**

编译器翻译的方式有两种：一个是**编译**，另外一个是**解释**。两种方式之间的区别在于**翻译时间点的不同**。当编译器**以解释方式运行的时候**，也称之为**解释器。**

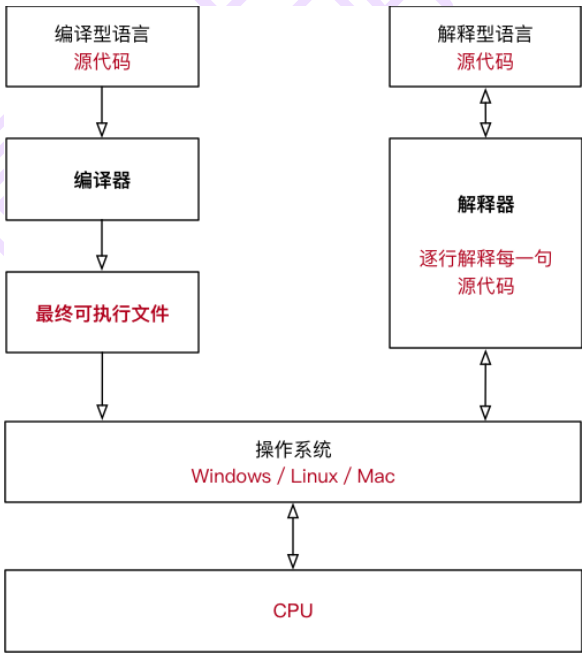


图1-3 编译型和解释型语言工作对比

(1) **编译型语言**：程序在执行之前需要一个专门的编译过程，把程序编译成为机器语言的文件，运行时不需要重新翻译，直接使用编译的结果就行了。程序执行效率高，依赖编译器，跨平台性差些，如 C、C++等。

(2) **解释型语言**：解释型语言编写的程序不进行预先编译，以文本方式存储程序代码，会将代码一句一句的直接运行。在发布程序时，看起来省了道编译工序，但是在运行程序的时候，必须先解释再运行。Python 是解释型语言。

编译型语言和解释型语言对比

(1) **速度**——编译型语言比解释型语言执行速度快

(2) **跨平台性**——解释型语言比编译型语言跨平台性好

1.2.2 Python 的设计目标

1999 年，吉多·范·罗苏姆向 DARPA 提交了一条名为“Computer Programming for Everybody”的资金申请，并在后来说明了他对 Python 的目标：

(1) 一门**简单直观的语言**并与主要竞争者一样强大

(2) **开源**，以便任何人都可以为它做贡献

(3) 代码**像纯英语那样容易理解**

(4) 适用于**短期**开发的日常任务

这些想法基本都已经成为现实，Python 已经成为一门流行的编程语言。

1.2.3 Python 的设计哲学

(1) 优雅

(2) 明确

(3) 简单

- Python 开发者的哲学是：**用一种方法，最好是只有一种方法来做一件事**
- 如果面临多种选择，Python 开发者一般会拒绝花俏的语法，而选择**明确没有或者很少有歧义的语法**

在 Python 社区，吉多被称为“仁慈的独裁者”。

1.3 为什么选择 Python？

(1) 代码量少

(2) ……

同一样问题，用不同的语言解决，代码量差距还是很多的，一般情况下 Python 是 Java 的 **1/5**，所以说**人生苦短，我用 Python。**

1.4 Python 特点

1. Python 具有通用性

Python 语言可以用于几乎任何与程序设计相关 应用的开发，不仅适合训练变成思维，更适合诸如 数据分析、机器学习、人工智能、Web 开发等具体的 技术领域。

2. Python 语法简洁

Python 语法主要用来精确表达问题逻辑，更接近自然语言，只有 33 个保留字，十分简洁。

3. Python 生态高产

Python 解释器提供了几百个内置类和函数库， 此外，世界各地程序员通过开源社区贡献了十几万个第三方函数库，几乎覆盖了计算机技术的各个领 域，编写 Python 程序可以大量利用已有内置或第三方代码，具备良好的编程生态。

除了 Python 语法的三个重要特点外，Python 程序还有一些具体特点。即：平台无关、强制可读和支持中文。

(1) 平台无关

Python 程序可以在任何安装解释器的计算机环 境中执行，因此，可以不经修改地实现跨操作系统 运行。

(2) 强制可读

Python 通过强制缩进（类似文章段落的首行空 格）来体现语句间的逻辑关系，显著提高了程序的 可读性，进而增强了 Python 程序的可维护性。

(3) 支持中文

Python 3.x 版本采用 Unicode 编码表达所有字符信息。Unicode 是一种国际通用表达字符的编码体系，这使得 Python 程序可以直接支持英文、中文、 法文、德文等各类自然语言字符，在处理中文时更加灵活且高效。

此外 Python 还有一些其他特性：

1. Python 是**完全面向对象的语言**

(1) **函数、模块、数字、字符串**都是对象，**在 Python 中一切皆对象**

(2) 完全支持继承、重载、多重继承

(3) 支持重载运算符，也支持泛型设计

2. Python **拥有一个强大的标准库**

Python 语言的核心只包含**数字、字符串、列表、字典、文件** 等常见类型和函数，而由 Python 标准库提供了**系统管理、网络通信、文本处理、数据库接口、图形系统、XML 处理**等额外的功能。

3. Python 社区提供了**大量的第三方模块(三方库)**

Python 大量的第三方模块，使用方式与标准库类似。它们的功能覆盖**科学计算、人工智能、机器学习、Web 开发、数据库接口、图形系统**等多个领域。

1.5 Python 的优缺点

1.5.1 优点

(1) 简单、易学

(2) 免费、开源

(3) **面向对象**

(4) 丰富的第三方模块 (三方库)

(5) 可扩展性

- 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，可以把这部分程序用 C 或 C++ 编写，然后在 Python 程序中使用它们。

(6) ……

1.5.2 缺点

(1) 运行速度

(2) 移动开发很少

(3) 版本兼容性不好

(4) ……

运行速度慢，这一缺点，可以通过寻找合适的第三方模块，或者干脆自己用 C 语言给 Python 实现一个模块，基本上可以解决。此外，根据 28 定律，一个软件，只有 20%的代码是核心代码，是经常用的代码，只需要优化和提高这部分的代码的速度就可以显著提高

软件的运行速度，因此，大家可以只集中精力优化和提高核心部分的运行速度即可，其他80%的部分，基本上不在乎速度的，完全可以用 Python 快速实现。

1.6 Python 的安装

1.6.1 下载和安装

在 Windows 下安装，浏览器地址栏输入 https://www.python.org/downloads/windows/ 找到自己想安装的 Python 的版本(注意操作系统是否为 64 位)下载并安装，Python 的版本最好选择 3.5 以上的，因为很多三方库，要求 Python 的版本在 3.5 以上。在 Windows 下安装，建议下载 Windows 版的可执行安装包安装，这样安装完成后可以直接关联.py 文件；并且安装时，对于 3.5 以上版本，如果勾选了“Add Python 3.x to PATH”那么就可以直接命令行下运行 Python，这非常适合于初学者。

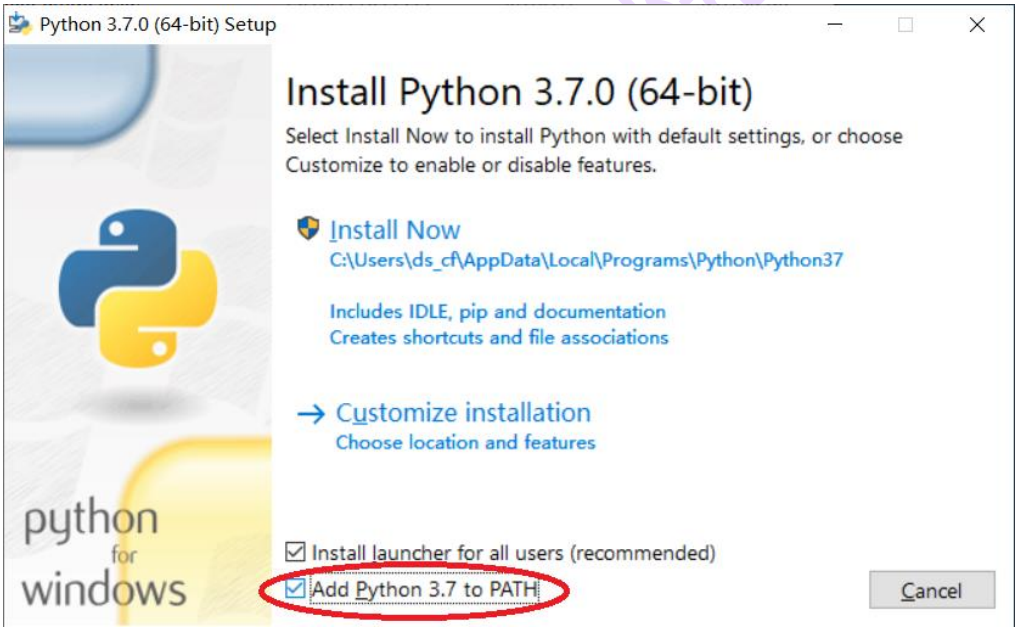


图1-4 勾选【Add Python 3.x to PATH】

在 Linux 下安装，在 https://www.python.org/downloads/source/地址下载相应源代码编译安装，最后一章会详细介绍，这里就不做进一步介绍了。

1.6.2 搜索路径设置

安装完成后，在相应的 Linux 或者 Windows 操作系统中，给环境变量 PATH 中添加 Python 可执行程序的搜索路径（Windows 版的 3.5 以上版本的可执行安装包，勾选过“Add Python 3.x to PATH”的搜索路径已经设置）。初学者这一步骤可暂时不学，因此建

议初学者下载 Windows 下的 Python3.5 以上版本的可执行安装包，勾选过“Add Python 3.x to PATH”即可。

习 题

一、简答题

1、Python 有哪些特点?

2、编译型语言和解释型语言有哪些相同和不同？Python 是哪一类型语言？

3、下载并安装 Python 3.x

第 2 章 第一个 Python 程序

知识点

(1) 第一个 Hello Python 程序

(2) 注释

(3) Python 2.x 与 3.x 版本简介

(4) 执行 Python 程序的三种方式

1) 解释器—— python / python3

2) 交互式—— ipython

3) 集成开发环境 IDE

a) 简单的开发环境——IDLE

b) 集成开发环境—— PyCharm

2.1 第一个 Hello Python 程序

2.1.1 Python 源程序的基本概念

Python 源程序就是**一个特殊格式的文本文件**，可以**使用任意文本编辑软件**做 Python 的开发。比如 Windows 下的记事本，Linux 下的 vi 等。

Python 程序的**文件扩展名**通常都是**.py** 或者**.pyw**

演练步骤

(1) Python 下载安装完成后，在桌面下，新建认识 Python 文件夹

(2) 在认识 Python 文件夹下新建 hellopython.py 文件

(3) 使用记事本等文本编辑器编辑 hellopython.py 并且输入以下内容：

```
print("hello world ")
print("你好！！ ")
```

在 Windows 终端中输入以下命令执行 hellopython.py（Linux 或 Windows 需要提前安装了 Python 并设置了搜索路径）：

```
C:\> python hellopython.py
```

上面 C:\> 打头的是 Windows 的 cmd 提示符。

此外，在 Windows 中只要安装了 Python 就可以文件夹中，右击 hellopython.py 文件，选择 Edit with IDLE 打开，然后按 F5 键运行。

print 是 python 中我们学习的第一个**函数。**

print 函数的作用，可以把双引号""内部的内容，输出到屏幕上。

2.1.3 演练扩展——认识错误（BUG）

关于错误

(1) 编写的程序**不能正常执行**，或者**执行的结果不是我们期望的**

(2) 俗称 **BUG**，是程序员在开发时非常常见的，初学者常见错误的原因包括：

1) 手误

2) 对已经学习过的知识理解还存在不足

3) 对语言还有需要学习和提升的内容

(3) 在学习语言时，不仅要**学会语言的语法**，而且还要**学会如何认识错误和解决错误的方法**

每一个程序员都是在不断地修改错误中成长的。

第一个演练中的常见错误

(1) **手误**，例如使用 pirnt("Hello world")

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

名称错误：'pirnt' 名字没有定义

(2) 将多条 print 写在一行

```
SyntaxError: invalid syntax
```

语法错误：语法无效

每行代码负责完成一个动作

(3) 缩进错误

```
IndentationError: unexpected indent
```

缩进错误：不期望出现的缩进

- Python 是一个格式非常严格的程序设计语言
- 目前而言，大家记住每行代码前面都不要增加空格

(4) **python 2.x 默认不支持中文**

目前市场上有两个 Python 的版本并存着，分别是 Python 2.x 和 Python 3.x

1) **Python 2.x 默认不支持中文**

2) Python 2.x 的解释器名称是 **python**，用户可以建立链接名 python2

3) Python 3.x 的解释器名称也是 **python**，用户可以建立链接名 python3

例如下面的错误：

```
SyntaxError: Non-ASCII character '\xe4' in file 01-HelloPython.py on line 3,
but no encoding declared;
see http://python.org/dev/peps/pep-0263/ for details
```

语法错误： 在 hellopython.py 中第 3 行出现了非 ASCII 字符 '\xe4'，但是没有声明文件编码

请访问 http://python.org/dev/peps/pep-0263/ 了解详细信息

1) ASCII 字符只包含 256 个字符，不支持中文

2) 有关字符编码的问题，后续会讲

常见错误单词列表：

```
* error 错误
* name 名字
* defined 已经定义
* syntax 语法
* invalid 无效
* Indentation 索引
* unexpected 意外的，不期望的
* character 字符
* line 行
* encoding 编码
* declared 声明
* details 细节，详细信息
* ASCII 一种字符编码
```

2.2 注释

2.2.1 注释的作用

使用自己熟悉的语言，在程序中对某些代码进行标注说明，增强程序的可读性。注释主要是为了解释程序段，及屏蔽不被再次利用的代码。一般编程语言的注释分两种：单行注释和多行注释。在 C 语言里，单行注释是最常用的是//，多行注释最常用的是/* */。Python 里单行注释是常用#，多行注释常用三对单引号''' '''或者三对双引号""" """。

2.2.2 单行注释(行注释)

1. 整行的单行注释

以#开头，#右边的所有东西都被当作说明文字，而不是真正要执行的程序，只起到辅助说明作用

示例代码如下：

```
#这是第一个单行注释
print("hello python")
```

为了保证代码的可读性，#后面建议先添加一个空格，然后再编写相应的说明文字。

2. 代码后面增加的单行注释

在程序开发时，同样可以使用#在代码的后面（旁边）增加说明性的文字，但是，需要注意的是，**为了保证代码的可读性，注释和代码之间**至少要有**两个空格**

示例代码如下：

```
print("hello python")    # 输出 `hello python`
```

2.2.3 多行注释(块注释)

如果希望编写的**注释信息很多，一行无法显示**，就可以使用多行注释

要在 Python 程序中使用多行注释，可以用 **一对连续的三个引号**(单引号和双引号都可以)

示例代码如下：

```
"""
```

```
这是一个多行注释
在多行注释之间，可以写很多很多的内容……
"""
print("hello python")
```

2.2.4 什么时候需要使用注释？

(1) **注释不是越多越好**，对于一目了然的代码，不需要添加注释

(2) 对于**复杂的操作**，应该在操作开始前写上若干行注释

(3) 对于**不是一目了然的代码**，应在其行尾添加注释（为了提高可读性，注释应该至少离开代码 2 个空格）

(4) 绝不要描述代码，假设阅读代码的人比你更懂 Python，他只是不知道你的代码要做什么。

其实有一种简单而实用的方法，就是假定你写的程序，在一年之后，要求你自己能很快看懂，那么，你应当在你现在编写的代码中加什么样的注释？

在一些正规的开发团队，通常会有**代码审核**的惯例，就是一个团队中彼此阅读对方的代码

2.2.5 关于代码规范

Python 官方提供有一系列 PEP（Python Enhancement Proposals）文档

(1) 其中第 8 篇文档专门针对 **Python 的代码格式**给出了建议，也就是俗称的 **PEP 8**

(2) 文档地址：https://www.python.org/dev/peps/pep-0008/

(3) 谷歌有对应的中文文档：http://zh-google-styleguide.readthedocs.io/en/latest/google-python-styleguide/python_style_rules/

任何语言的程序员，编写出符合规范的代码，是开始程序生涯的第一步。

2.3 Python 2.x 与 3.x 版本简介

市场上有两个 Python 的版本并存着，分别是 Python 2.x 和 Python 3.x，Python 程序建议使用 Python 3.0 版本的语法。

1. Python 2.x 是**过去的版本**

   - 解释器名称是 **python**

2. Python 3.x 是**现在和未来主流的版本**

(1) 解释器名称也是 **python**，和 Python2 共存的时候建议建立 python3 链接

(2) 相对于 Python 的早期版本，这是一个**较大的升级**

(3) 为了不带入过多的累赘，Python 3.0 在设计的时候**没有考虑向下兼容**

- 许多早期 Python 版本设计的程序都无法在 Python 3.0 上正常执行

(4) Python 3.0 发布于 **2008 年**

(5) 到目前为止，Python 3.0 的稳定版本已经有很多年了

1) Python 3.3 发布于 2012

2) Python 3.4 发布于 2014

3) Python 3.5 发布于 2015

4) Python 3.6 发布于 2016

5) Python 3.7 发布于 2018

6) Python 3.8 发布于 2019

3. 为了照顾现有的程序，官方提供了一个过渡版本—— Python 2.6

(1) 基本使用了 Python 2.x 的语法和库

(2) 同时考虑了向 Python 3.0 的迁移，**允许使用部分** Python 3.0 的语法与函数

(3) 2010 年中推出的 Python 2.7 被确定为**最后一个 Python 2.x 版本**

(4) Python 2.x 现在基本上被淘汰了

新开发的 Python 程序基本上都是 Python3.x 的，而且旧的 Python2.x 的程序也逐渐的被迁移到了 Python3.x 了，因此 Python2.x 基本上被淘汰了。

2.4 执行 Python 程序的三种方式

执行 Python 的方式有：解释器，交互式和集成开发环境(IDE)。下面逐一进行介绍

2.4.1 解释器

Python 的解释器：

使用 python 3.x 解释器(用户建立了 Python 的搜索路径)：

```
C:\ > python xxx.py
```

上面的 C:\ >打头的是 Windows 下 cmd 中的运行方式。

Python 的其他解释器（了解即可）如今有多个语言的实现，包括：

(1) CPython —— 官方版本的 C 语言实现

(2) Jython —— 可以运行在 Java 平台

(3) IronPython —— 可以运行在 .NET 和 Mono 平台

(4) PyPy —— Python 实现的，支持 JIT 即时编译

2.4.2 交互式运行 Python 程序

1. 交互式运行 Python 程序的特点：

(1) 交互式编程不需要创建脚本文件，通过 Python 解释器的交互模式进来编写代码。

(2) 直接在终端中运行解释器，而不输入要执行的文件名

(3) 在 Python 的 Shell 中直接输入 **Python 的代码**，会立即看到程序执行结果

2. 交互式运行 Python 的优缺点

优点：

- 适合于学习/验证 Python 语法或者局部代码

缺点：

(1) 代码不能保存

(2) 不适合运行太大的程序

3. 退出官方的交互式解释器

(1) 直接输入 exit()

```
>>> exit()
```

(2) 使用热键退出

在 python 解释器中，按热键 ctrl + d 可以退出解释器

4. IPython

- IPython 中的“I”代表**交互 interactive**

(1) IPython **特点：**

1) IPython 是一个 python 的 **交互式 shell**，比默认的 python shell 好用

a) 支持自动补全

b) 自动缩进

c) 支持 bash shell 命令

d) 内置了许多很有用的功能和函数

2) IPython 是基于 BSD 开源的

(2) IPython **版本：**

1) Python 2.x 使用的解释器是 **ipython**

2) Python 3.x 使用的解释器也是 **ipython**，和 Python2.x 共存的时候，建议改为链接 ipython3

(3) **IPython 的运行：**

运行 IPython 只需要在命令行下输入 ipython 回车即可(用户建立了 Python 的搜索路径)：

```
C:\ > ipython
```

上面的 C:\ >打头的是 Windows 下 cmd 中的提示符。运行后，会出现类似下面的界面：

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

(4) **退出 IPython 解释器：**

要退出解释器可以有以下两种方式。

1) 直接输入 exit

```
In [1]: exit
```

2) 使用热键退出

在 IPython 解释器中，按热键 ctrl + d，IPython 会询问是否退出解释器

(5) **IPython 的安装**（一般安装 Python 的时候已经默认安装了）

```
C:\ >pip install ipython
```

上面的 C:\ >打头的是 Windows 下 cmd 中的提示符，Windows 下需要提前设置 Python 的搜索路径。

2.4.3 Python 的 IDE —— IDLE

1. 集成开发环境（IDE）

集成开发环境（IDE，Integrated Development Environment）—— **集成了开发软件需要的所有工具**，一般包括以下工具：

(1) 图形用户界面

(2) 代码编辑器（支持**代码补全 / 自动缩进**）

(3) 编译器 / 解释器

(4) 调试器（**断点 / 单步执行**）

(5) ……

Python 的集成开发环境（IDE）有很多，PyCharm、Spyder 等等，其中 Python 自带的简洁的集成开发环境**是 IDLE**。

2. IDLE 介绍

(1) IDLE 是 Python 自带的一款简洁的集成开发环境

(2) IDLE 除了具有一般 IDE 所必备功能外，还可以在 Windows、Linux、macOS 下使用

(3) IDLE 适合开发中小型项目

1) 一个项目通常会包含**很多源文件**

2) 每个**源文件**的代码行数是有限的，通常在几百行之内

3) 每个**源文件**各司其职，共同完成复杂的业务功能

Python 安装后，默认自带此工具，启用：开始->程序->Python 2.*/3.*-> IDLE (Python GUI)如此就打开了 Python Shell，可以输入语句命令进行交互练习：

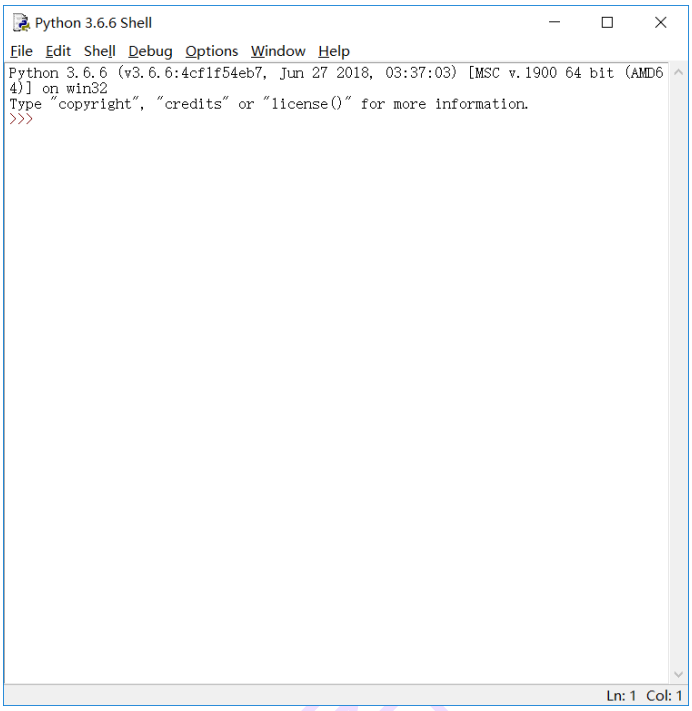


图 2-1 IDLE 的 Python Shell

3. IDLE 快速体验

IDLE 的 Python Shell 的菜单 File->New File(Ctrl+N)可以打开 Python 文件编辑器（右击任何一个.py 文件，弹出菜单中的“Edit with IDLE”也可以调用 IDLE 打开这个.py 文件然后进行调试）。

(1) **文件(File)菜单**能够**新建**／**定位**／**打开** Python 文件

(2) **文件编辑区域**能够**编辑**当前打开的文件

(3) **运行(Run)菜单**能够**执行(F5)**代码

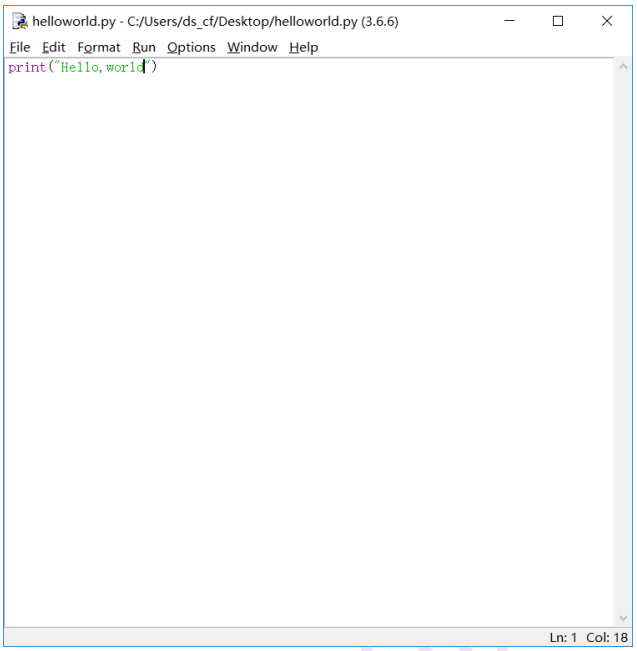


图 2-2 IDLE 编辑器

2.4.4 Python 的 IDE —— PyCharm

1. PyCharm 介绍

(1) PyCharm 是 Python 的一款非常优秀的集成开发环境

(2) PyCharm 除了具有一般 IDE 所必备功能外，还可以在 Windows、Linux、macOS 下使用

(3) PyCharm 适合开发大型项目

1) 一个项目通常会包含**很多源文件**

2) 每个**源文件**的代码行数是有限的，通常在几百行之内

3) 每个**源文件**各司其职，共同完成复杂的业务功能

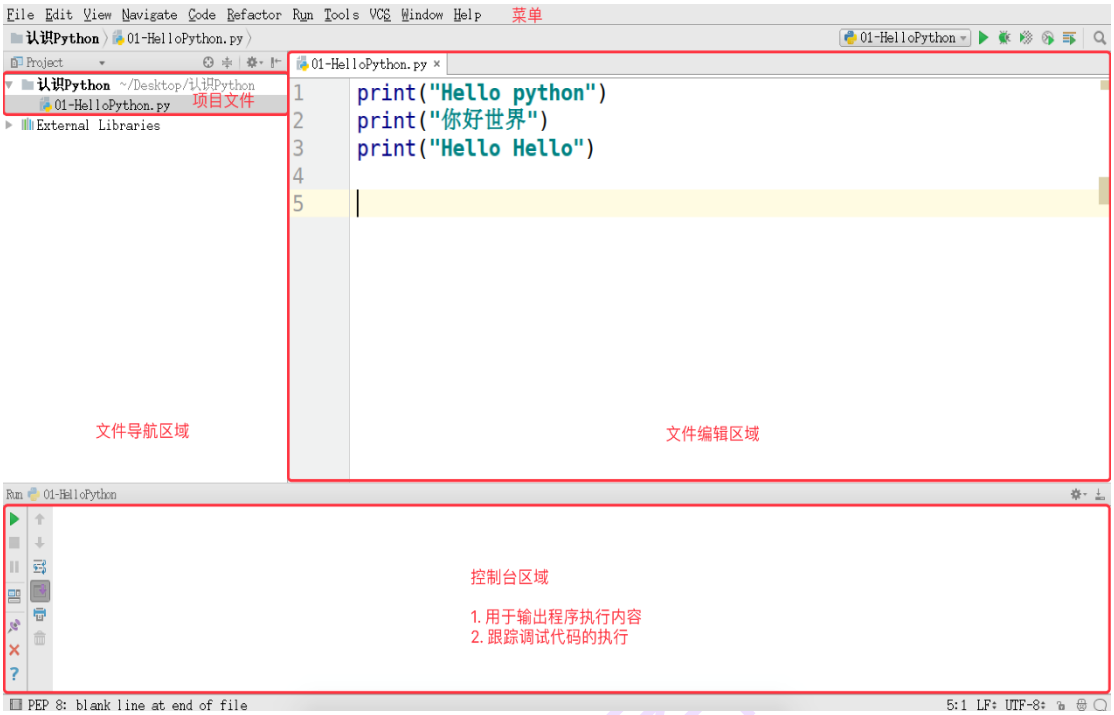


图 2-3 PyCharm 的界面结构

2. PyCharm 快速体验

(1) **文件导航区域**能够**浏览** / **定位** / **打开**项目文件

(2) **文件编辑区域**能够**编辑**当前打开的文件

(3) **控制台区域**能够：

1) 输出程序执行内容

2) 跟踪调试代码的执行

(4) 右上角的**工具栏**能够**执行(SHIFT + F10)** / **调试(SHIFT + F9)**代码

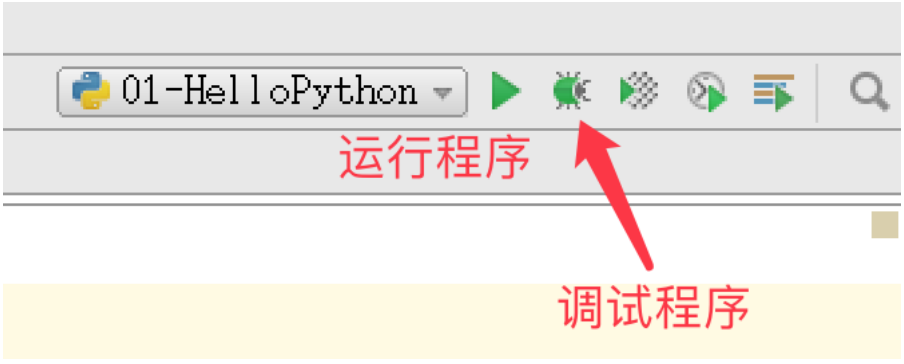


图 2-4 PyCharm 运行工具栏

(5) 通过控制台上方的**单步执行按钮(F8)**，可以单步执行代码

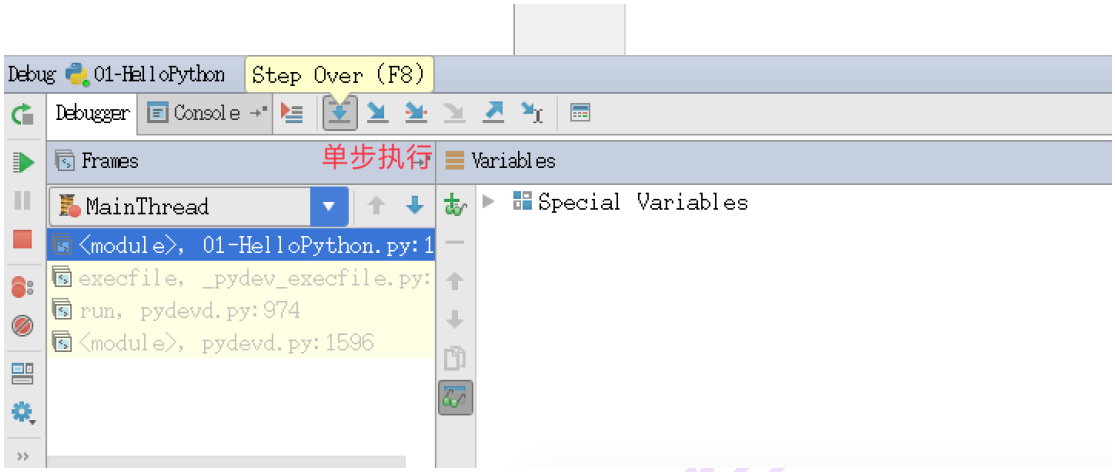


图 2-5　PyCharm 调试器

习 题

一、简答题

1、 执行 Python 有哪三种方式？

2、 简单介绍 2 种以上用于 Python 开发的 IDE。

第 3 章 变量

知识点

(1) 标识符和关键字

(2) 变量的命名规则

(3) 变量定义

(4) 变量的类型

(5) 变量的命名

3.1. 变量的命名

3.1.1 标识符和关键字

3.1.1.1 标识符

标识符就是程序员定义的**变量名**、**函数名**

名字需要有**见名知义**的效果

(1) 标示符可以由**字母**、**下划线**和**数字**组成

(2) **不能以数字开头**

(3) **不能与关键字重名**

思考：下面的标示符哪些是正确的，哪些不正确为什么？

```
fromNo12
from#12
my_Boolean
my-Boolean
Obj2
2ndObj
myInt
My_tExt
```

```
_test
test!32
haha(da)tt
jack_rose
jack&rose
GUI
G.U.I
```

3.1.1.2 关键字

(1) **关键字**就是在 Python 内部已经使用的标识符

(2) **关键字**具有特殊的功能和含义

(3) 开发者**不允许定义和关键字相同的名字的标示符**

通过以下命令可以查看 Python 中的关键字

```
>>>import keyword
>>>print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
```

★提示：**关键字的学习及使用**，会在后面的课程中不断介绍

(1) import **关键字**可以导入一个 **“模块”**

(2) 在 Python 中不同的模块，提供有不同的功能

3.1.2 变量的命名规则

命名规则可以被视为一种**惯例**，并无绝对与强制 目的是为了**增加代码的识别和可读性**

注意：Python 中的**标识符**是**区分大小写的**

(1) 在定义变量时，为了保证代码格式，=的左右应该各保留一个空格

(2) 在 Python 中，如果**变量名**需要由**二个**或**多个单词**组成时，可以按照以下方式命名：

1) 每个单词都使用小写字母

2) 单词与单词之间使用_**下划线**连接

3) 例如：first_name、last_name、qq_number、qq_password

3.1.2.1 驼峰命名法

(1) 当**变量名**是由二个或多个单词组成时，还可以利用驼峰命名法来命名

(2) **小驼峰式命名法**

1) 第一个单词以小写字母开始，后续单词的首字母大写

2) 例如：firstName、lastName

图 3-1 驼峰命名法

(3) **大驼峰式命名法**

1) 每一个单词的首字母都采用大写字母

2) 例如：FirstName、LastName、CamelCase

习惯上，小驼峰命名法，经常用来命名函数名、变量名和方法名；大驼峰命名法，经常用来命名类名。关于类和面向对象的概念以及类的方法，后面会详细介绍。

3.2. 变量的使用

程序就是用来处理数据的，而变量就是用来存储数据的。Python 中的变量不需要先做类型声明。每个变量在使用前都必须赋值，一个变量赋值以后其才被真正创建，并有了数据类型，当下次该变量被重新赋值时，原来的变量消失，新的类型变量被创建。

3.2.1 变量定义

(1) 在 Python 中，每个变量**在使用前都必须赋值**，变量**赋值以后**，该变量**才会被创建**

(2) 变量定义格式：

变量名 = 值

(3) 等号（=）用来给变量赋值

1) = 左边是一个变量名

2) = 右边是存储在变量中的值

变量定义之后，后续就可以直接使用了。没有被赋值的变量，只能被赋值，不能直接用于其他运算。

3.2.1.1 变量演练 1——交互式 Python

定义 qq_number 的变量用来保存 qq 号码

```
>>>qq_number = "1234567"
```

输出 qq_number 中保存的内容

```
>>>qq_number
'1234567'
```

定义 qq_password 的变量用来保存 qq 密码

```
>>>qq_password = "123"
```

输出 qq_password 中保存的内容

```
>>>qq_password
 '123'
```

使用交互式方式，如果要查看变量内容，直接输入变量名即可，不需要使用 print 函数

3.2.1.2 变量演练 2—— IDE

Python 的 IDE（如 IDLE 或者 PyCharm）中输入下面的程序代码并运行：

```
# 定义 qq 号码变量
qq_number = "1234567"
# 定义 qq 密码变量
```

```
qq_password = "123"
# 在程序中，如果要输出变量的内容，需要使用 print 函数
print(qq_number)
print(qq_password)
```

使用 IDE 执行，如果要输出变量的内容，必须要要使用 print 函数

3.2.1.3 变量演练 3——超市买苹果

(1) 可以用**其他变量的计算结果**来定义变量

(2) 变量定义之后，后续就可以直接使用了

需求：

(1) 苹果的价格是 **8.5 元/斤**

(2) 买了 **7.5 斤** 苹果

(3) 计算付款金额

```
# 定义苹果价格变量
price = 8.5
# 定义购买重量
weight = 7.5
# 计算金额
money = price * weight
print(money)
```

思考题：

(1) 如果**只要买苹果，就返 5 块钱**

(2) 请重新计算购买金额

```
# 定义苹果价格变量
price = 8.5
# 定义购买重量
```

```
weight = 7.5
# 计算金额
money = price * weight
# 只要买苹果就返 5 元
money = money - 5
print(money)
```

提问：

(1) 上述代码中，一共定义有几个变量？

- 三个：price / weight / money

(2) money = money - 5 是在定义新的变量还是在使用变量？

1) 直接使用之前已经定义的变量

2) 变量名只有在**第一次出现**才是**定义变量**

3) 变量名再次出现，不是定义变量，而是直接使用之前定义过的变量

(3) 在程序开发中，可以修改之前定义变量中保存的值吗？

1) 可以

2) 变量中存储的值，就是可以**变**的

3.2.2 变量的类型

在内存中创建一个变量，会包括：

(1) 变量的名称

(2) 变量保存的数据

(3) 变量存储数据的类型

(4) 变量的地址（标示）

3.2.2.1 变量类型的演练——个人信息

需求：

(1) 定义变量保存小明的个人信息

(2) 姓名：**小明**

(3) 年龄：**18** 岁

(4) 性别：**是**男生

(5) 身高：**1.75** 米

(6) 体重：**75.0** 公斤

利用**单步调试**确认变量中保存数据的类型

提问

(1) 在演练中，一共有几种数据类型？

1) 4 种

2) str —— 字符串

3) bool —— 布尔（真假）

4) int —— 整数

5) float —— 浮点数（小数）

(2) 在 Python 中定义变量时需要指定类型吗？

1) 不需要

2) Python 可以根据 = 等号右侧的值，自动推导出变量中存储数据的类型

3.2.2.2 变量的类型分类简介

通过前面的演练，可以看出在 Python 定义变量是**不需要指定类型**（在其他很多高级语言中都需要）,而是通过赋值右侧表达式的结果来确定变量的数据类型的。Python 的数据类型有很多分类方法，下面给出了常用的两种分类方法：

1. 数据类型根据是否是数字可以分为数字型和非数字型

(1) 数字型

1) 整型 (int)

2) 浮点型（float）

3) 布尔型（bool）

- 真 True 非 0 数 —— **非零即真**
- 假 False 0

4) 复数型 (complex)

- 主要用于科学计算，例如：平面场问题、波动问题、电感电容等问题

(2) 非数字型

1) 字符串

2) 列表

3) 元组

4) 字典

★提示：在 Python 2.x 中，**整数**根据保存数值的长度还分为：

(1) int（整数）

(2) long（长整数）

使用 type 函数可以查看一个变量的类型

```
type(name)
```

2. 数据类型还可以根据复杂性分为简单类型和高级类型

(1) 简单类型

1) 整型 (int)

2) 浮点型（float）

3) 布尔型（bool）

4) 复数型 (complex)

5) 字符串

(2) 高级类型

1) 列表

2) 元组

3) 集合

4) 字典

3.2.2.3 不同类型变量之间的计算

3.2.2.3.1 数字型变量之间可以直接计算

(1) 在 Python 中，两个数字型变量是可以直接进行算数运算的

(2) 如果变量是 bool 型，在计算时

1) True 对应的数字是 1

2) False 对应的数字是 0

演练步骤

(1) 定义整数 i = 10

(2) 定义浮点数 f = 10.5

(3) 定义布尔型 b = True

(4) 在 交互式 Python 中，使用上述三个变量相互进行算术运算

3.2.2.3.2 字符串变量之间使用“+”拼接字符串

在 Python 中，字符串之间可以使用“+” 拼接生成新的字符串

```
>>> first_name = "福明"
>>> last_name = "陈"
>>> first_name + last_name
'福明陈'
```

3.2.2.3.3 字符串变量可以和整数使用“*”重复拼接相同的字符串

```
>>> "-" * 50
 '--------------------------------------------------'
```

3.2.2.3.4 数字型变量和字符串之间不能进行其他计算

```
>>> first_name = "zhang"
>>> x = 10
>>>    x + first_name
---------------------------------------------------------------------------
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

类型错误：`+` 不支持的操作类型：`int` 和 `str`

3.2.3 变量的输入

(1) 所谓**输入**，就是**用代码获取**用户通过**键盘**输入的信息

(2) 例如：去银行取钱，在 ATM 上输入密码

(3) 在 Python 中，如果要获取用户在**键盘**上的输入信息，需要使用到 input 函数

3.2.3.1 关于函数

(1) 一个**提前准备好的功能**(别人或者自己写的代码)，**可以直接使用**，而**不用关心内部的细节**

(2) 目前已经学习过的函数

表 3-1 已学过的函数

函数	说明
print(x)	将 x 输出到控制台
type(x)	查看 x 的变量类型

函数后面会详细介绍，这里只是简单了解。

3.2.3.2 input 函数实现键盘输入

(1) 在 Python 中可以使用 input 函数从键盘等待用户的输入

(2) 用户输入的任何内容 Python3 都认为是一个字符串

(3) 语法如下：

字符串变量 = input("提示信息：")

```
>>> s = input('Please input:')
Please input:12
>>> s
'12'
>>> type(s)
<class 'str'>
```

3.2.3.3 类型转换函数

表 3-2 类型转换函数

函数	说明
int(x)	将 x 转换为一个整数
float(x)	将 x 转换到一个浮点数

3.2.3.4 变量输入演练——超市买苹果增强版

需求：

(1) **收银员输入**苹果的价格，单位：**元 / 斤**

(2) **收银员输入**用户购买苹果的重量，单位：**斤**

(3) 计算并且**输出**付款金额

演练方式 1

```
# 1. 输入苹果单价
price_str = input("请输入苹果价格：")
# 2. 要求苹果重量
weight_str = input("请输入苹果重量：")
# 3. 计算金额
# 1> 将苹果单价转换成小数
price = float(price_str)
# 2> 将苹果重量转换成小数
weight = float(weight_str)
# 3> 计算付款金额
money = price * weight
print(money)
```

提问：

(1) 演练中，针对**价格**定义了几个变量？

1) **两个**

2) price_str 记录用户输入的价格字符串

3) price 记录转换后的价格数值

(2) **思考**——如果开发中，需要用户通过控制台输入**很多个数字**，针对每一个数字都要定义两个变量，**方便吗**？

演练方式 2 —— 买苹果改进版

(1) **定义**一个**浮点变量**接收用户输入的同时，就使用 float 函数进行转换

```
price = float(input("请输入价格:"))
```

改进后的好处：

(1) 节约空间，只需要为一个变量分配空间

(2) 起名字方便，不需要为中间变量起名字

改进后的“缺点”：

- 初学者需要知道，两个函数能够嵌套使用，稍微有一些难度

★提示

- 如果输入的不是一个数字，程序执行时会出错，有关数据转换的高级话题，后续会讲！

3.2.4 eval 函数

演练方式 3 ——买苹果 eval 版

定义一个**浮点变量**接收用户输入的同时，就使用 eval 函数进行转换

```
price = eval(input("请输入价格:"))
```

3.2.4.1 eval 函数的功能：

(1) eval 函数的参数**必须是字符串**。所以经常和 input 联合使用。

(2) eval 函数的功能，通俗的说，就是先**去掉字符串的一对引号**，使得字符串变成表达式

(3) eval 函数的最终功能，就是返回表达式经过**计算**的值

```
la=20
```

```
lb = eval("la") #相当于  lb=la
print(lb)
#结果是 20
```

演练 1:

```
la=20
lb = eval("la+2") #相当于  lb=la+2
print(lb)
#结果是 22
```

演练 2:

```
lb = eval("la+2") #相当于  lb=la+2, la 赋值，直接报错
print(lb)
#结果是 22
```

3.2.5 print 的参数

在 Python 中可以使用 print 函数将信息输出到控制台该函数的语法如下：

print(*objects, sep=' ', end='\n', file=sys.stdout)

参数的具体含义如下：

(1) objects --表示输出的对象，输出多个对象时，需要用,（逗号）分隔；

(2) sep -- 用来间隔多个对象；

(3) end -- 用来设定以什么结尾，默认值是换行符 \n，我们可以换成其他字符；

(4) file -- 要写入的文件对象。默认是标准输出设备，一般就是屏幕。

一般数据类型，如数值型，布尔型，列表变量，字典变量等都可以用 print 直接输出。

print 参数的演练

```
#变量的输出
num = 19
print (num)                    #19   输出数值型变量
str = 'Duan Yixuan'
```

```
print(str)                    #Duan Yixuan   输出字符串变量
list = [1,2,'a']
print (list)                  #[1, 2, 'a']   输出列表变量
tuple = (1,2,'a')
print (tuple)                 #(1, 2, 'a')  输出元组变量
dict = {'a':1, 'b':2}
print (dict)                  # {'a': 1, 'b': 2}  输出字典变量
```

利用 end 参数，可以换行与防止换行

在 Python 中，输出函数总是默认换行，比如说：

```
for x in range(0,5):
    print(x)
```

运行结果：

```
0
1
2
3
4
```

显然，这种输出太占“空间”，这是因为每个 print 语句默认结束符是 \n，我们可以使用 end 设定以特定字符结尾，如使用空格、逗号等表示结束。

```
for x in range(0, 5):
    print(x, end=' ')
```

运行结果：

```
0 1 2 3 4
```

再举一例：

```
for x in range(0, 5):
    print(x, end=',')
```

运行结果：

```
0,1,2,3,4,
```

可以结合 print()本身带默认换行功能，实现更为高效的输出换行，如下：

```
for x in range(0, 5):
    print(x, end=' ')
print()  #本身自带换行，完美输出
for x in range(0, 5):
    print(x, end=',')
```

运行结果：

```
0 1 2 3 4
0,1,2,3,4,
```

3.2.6 变量的格式化符%输出(了解)

如果希望输出文字信息的同时，**一起输出数据**，就需要使用到**格式化操作符**

1. %**格式化操作符**

%被称为**格式化操作符**，专门用于处理字符串中的格式

(1) 包含%的字符串，被称为**格式化字符串**

(2) % 和不同的**字符**连用，**不同类型的数据**需要使用**不同的格式化字符**

表 3-3 格式化字符含义

%s	字符串
%d	有符号十进制整数，%06d 表示输出的整数显示位数，不足的地方使用 0 补全
%f	浮点数，%.2f 表示小数点后只显示两位
%%	输出 %

- 语法格式如下：

  print("格式化字符串" % 变量 1)

  print("格式化字符串" % (变量 1, 变量 2...))

我们可以先看一个简单的例子：

```
s='hello Python'
x=len(s)
print( 'The length of %s is %d' %(s, x) )
```

运行结果：

```
The length of Duan Yixuan is 12
```

'The length of %s is %d' 这部分叫做格式控制符，(s ,x) 这部分叫做转换说明符。

2. 格式化字符串的详细用法

如果要具体的控制输出格式，那么就得掌握格式化字符串的详细用法：

(1) 字段宽度和精度

如下面的例子，点（.）前数字表示最小字段宽度：转换后的字符串至少应该具有该值指定的宽度；如果是*（星号），则宽度会从值元组中第一个数字读出。点（.）后数字表示精度值：如果需要输出实数，精度值表示出现在小数点后的位数；如果需要输出字符串，那么该数字就表示最大字段宽度；如果是*，那么精度将从元组中第一个数字读出。

★注意：字段宽度中，小数点也占一位。

```
PI=3.1415926
print("PI=%5.*f"%(3, PI))
```

运行结果：

```
PI=3.142
```

下面的例子用*从后面的元组中读取字段宽度；精度是 3 位。

```
print("PI=%*.3f"% (10,PI))
```

运行结果：

```
PI=       3.142      #精度为 3，总长为 10
```

(2) 转换标志

转换标志主要用来表示对齐、填充方式及数字的正负号。-表示左对齐；+表示在数值前要加上正负号；" "(空白字符)表示正数之前保留空格；0 表示转换值若位数不够则用 0 填充。　　具体我们可以看一下例子。

1) 左对齐：

```
PI=3.1415926
print('%-10.3f' %PI)  #左对齐，空格显示在右边，还是 10 个字符
```

运行结果：

```
3.142
```

2) 显示正负号：

```
PI=3.1415926
print('%+f' % PI)  #显示正负号  #+3.141593，类型 f 的默认精度为 6 位小数
```

运行结果：

```
PI=+3.1415926
```

3) 用 0 填充空白：

```
print('%010.3f'%PI) #字段宽度为 10，精度为 3，不足处用 0 填充空白
```

运行结果：

```
000003.142        #0 表示转换值若位数不够则用 0 填充
```

格式化输出课堂练习——基本练习

需求：

(1) 定义字符串变量 name，输出：**我的名字叫小明，请多多关照！**

(2) 定义整数变量 student_no，输出：**我的学号是 000001**

(3) 定义小数 price、weight、money，输出：**苹果单价 9.00 元 / 斤，购买了 5.00 斤，需要支付 45.00 元**

(4) 定义一个小数 scale，输出：**数据比例是 10.00%**

```
print("我的名字叫 %s，请多多关照！" % name)
print("我的学号是 %06d" % student_no)
print("苹果单价 %.02f 元 / 斤，购买 %.02f 斤，需要支付 %.02f 元" % (price, weight, money))
print("数据比例是 %.02f%%" % (scale * 100))
```

习 题

一、填空题

1、Python 中的单行注释标记是_____。

2、Python 中的多行注释标记是一对_____。

3、Python 中的标识符可以包含_____、_____或_____。

4、Python 中的标识符只可以以_____或_____打头。

5、查看变量内存地址的 Python 内置函数是_____。

二、判断题

1、Python 中的变量先声明类型后使用。（）

2、Python 中的变量先赋值定义后使用。（）

三、程序题

1、编一个程序，分别输入整数，浮点数，并打印输出。

第 4 章 数字数据类型及其运算

知识点

(1) 整数类型

(2) 浮点类型

(3) 布尔类型

(4) 复数类型

(5) format 格式化输出

(6) 算术运算符的基本使用

(7) 其他运算符的基本使用

4.1 数字数据类型

上一章介绍变量的时候，简单介绍了数据类型，Python 依据数据是否为数字，可以把 Python 数据类型划分为数字数据类型和非数字数据类型。本章进一步详细介绍数字数据类型。

4.1.1 整数类型

Python 中整数类型与数学中整数概念一致，共有 4 种进制表示：十进制，二进制，八进制和十六进制。默认情况，整数采用十进制，其它进制需要增加相应的引导符号，具体如下：

(1) **十进制整数**如，0、-1、9、123

(2) **十六进制整数**，需要 16 个数字 0、1、2、3、4、5、6、7、8、9、a、b、c、d、e、f 来表示整数，必须以 0x 开头，如 0x10、0xfa、0xabcdef

(3) **八进制整数**，只需要 8 个数字 0、1、2、3、4、5、6、7 来表示整数，必须以 0o 开头，如 0o35、0o11

(4) **二进制整数**，只需要 2 个数字 0、1 来表示整数，必须以 0b 开头如，0b101、0b100

整数类型的取值范围在理论上没有限制，实际上受限制于运行 Python 程序的计算机内存大小。

以下那些是正确的整数类型？那些是错误的整数类型？

0xaaa, 0o921, 0b210, 0XAA90, 987, 0b1100, 0o172

整数类型变量，**默认打印值是十进制整数**，如：

```
>>> n = 0xa1
>>> print(n)
161
>>> m = 0b1010
>>> print(m)
10
```

4.1.2 浮点类型

浮点数类型表示有小数点的数值。浮点数有两种表示方法：小数表示和科学计数法表示，如：

15.0、0.37、-11.2、1.2e2、314.15e-2

Python 浮点数的取值范围和小数精度受不同计算机系统的限制，sys.float_info 详细列出了 Python 解释器所运行系统的浮点数各项参数，例如：

```
>>> import sys
>>> sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
>>> sys.float_info.max
1.7976931348623157e+308
>>>
```

浮点数类型直接表示或科学计数法中的系数（E 或 e 前面的数）最长可输出 16 个数字，浮点数运算结果中最长输出 17 个数字。然而根据 sys.float_info.dig 的值(dig=15)，计算

机只能提供 15 个数字的准确性。浮点数在超过 15 位数字计算中产生的误差与计算机内部采用二进制运算有关。

4.1.3 布尔类型

布尔型（bool）：

(1) 真 True 非 0 数 —— **非零即真**

(2) 假 False 0

此外布尔型的 True 和 False 的值是 1 和 0，还可和数字相加。

4.1.4 复数类型

(1) 复数类型表示数学中的复数。复数有一个基本单位元素 j，叫作“虚数单位”。含有虚数单位的数被称为复数。例如：

(2) 11.3+4j -5.6+7j 1.23e-4+5.67e+89j

(3) Python 语言中，复数可以看作是二元有序实数对（a, b），表示为：a + bj，其中，a 是实数部分，简称实部，b 是虚数部分，简称虚部。**虚数部分通过后缀“J”或者“j”来表示**。需要注意，当 b 为 1 时，1 不能省略，即 1j 表示复数，而 j 则表示 Python 程序中的一个变量。

(4) 复数类型中实部和虚部都是浮点类型，对于复数 z，可以用 z.real 和 z.imag 分别获得它的实数部分和虚数部分

★注意：abs()函数，用于复数，是求复数实部和虚部的均方根：

```
>>> abs(3+4j)
5.0
```

4.1.5 数字类型数据演练

```
>>> 1+True-False
2
>>> a, b, c, d = 20, 5.5, True, 4+3j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

4.1.6 格式化输出——format()方法

如上一章例题，苹果单价 9.00 元 / 斤，购买了 5.00 斤，需要支付 45.00 元，输出这些数字的时候，就得按照格式要求输出，Python 提供了 format()方法进行格式化输出。

format()方法是字符串的一个方法，专门用于处理字符串中的格式

字符串 format()方法的语法格式如下：

<模板字符串>.format(<逗号分隔的参数>)

其中，模板字符串是一个由字符串和槽组成的字符串，用来控制字符串和变量的显示效果。槽用大括号{}表示，对应 format()方法中逗号分隔的参数。如下面的样例：

print("模板字符串".format(变量 1))

print("模板字符串".format (变量 1, 变量 2...))

(1) 如果模板字符串有多个槽，且槽内没有指定序号，则按照槽出现的顺序分别对应.format()方法中的不同参数。

(2) format()方法中模板字符串的槽，除了包括参数序号，还可以包括格式控制信息。

{<参数序号>: <格式控制标记>}

(3) 其中，参数序号是 format 函数的参数的序号(从 0 开始)。此外参数序号也可以通过后面函数一章介绍的关键字参数，甚至可以是后面学到的高级数据中的一个元素或者对象的属性。

(4) 其中，格式控制标记用来控制参数显示时的格式 。格式控制标记包括：<填充><对齐><宽度>,<.精度><类型>6 个字段，这些字段都是可选的，可以组合使用

:	<填充>	<对齐>	<宽度>	,	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %

图 4-1 格式控制标记

(5) <填充>、<对齐>和<宽度>主要用于对显示格式的规范。

(6) 宽度指当前槽的设定输出字符宽度，如果该槽参数实际值比宽度设定值大，则使用参数实际长度。如果该值的实际位数小于指定宽度，则按照对齐指定方式在宽度内对齐，默认以空格字符补充。

(7) 对齐字段分别使用<、>和^三个符号表示左对齐、右对齐和居中对齐。

(8) 填充字段可以修改默认填充字符，填充字符只能有一个。

(9) <.精度><类型>主要用于对数值本身的规范

(10)<.精度>由小数点（.）开头。对于浮点数，精度表示小数部分输出的有效位数。对于字符串，精度表示输出的最大长度。小数点可以理解为对数值的有效截断。

(11)<类型>表示输出整数和浮点数类型的格式规则。

(12)对于整数类型，输出格式包括 6 种：

1) b: 输出整数的二进制方式；

2) c: 输出整数对应的 Unicode 字符；

3) d: 输出整数的十进制方式；

4) o: 输出整数的八进制方式；

5) x: 输出整数的小写十六进制方式；

6) X: 输出整数的大写十六进制方式；

(13)对于浮点数类型，输出格式包括 4 种：

1) e: 输出浮点数对应的小写字母 e 的指数形式；

2) E: 输出浮点数对应的大写字母 E 的指数形式；

3) f: 输出浮点数的标准浮点形式；

4) %: 输出浮点数的百分形式。

4.1.6.1 输出演练——基本练习

需求

(1) 定义字符串变量 name，输出：**我的名字叫小明，请多多关照！**

(2) 定义整数变量 student_no，输出：**我的学号是 000001**

(3) 定义小数 price、weight、money，输出：**苹果单价 9.00 元 / 斤，购买了 5.00 斤，需要支付 45.00 元**

(4) 定义一个小数 scale，输出：**数据比例是 10.00%**

```
print("我的名字叫{}，请多多关照！".format(name))
print("我的学号是 {:0<6d}".format(student_no))
print("苹果单价 {:0<.2f} 元 / 斤，购买{:0<.2f}斤，需要支付{:0<.2f}元" .format (price, weight, money))
print("数据比例是{:0<.2f}%".format (scale * 100))
```

4.1.6.2 输出演练——高级练习

需求

(1) 以各种进制输出**汉字'陈'的编码**

(2) 12 次投球中了 1 次，输出百分比的投中率，要求 6 位宽度并保留两位小数，左对齐，左侧补 0

(3) 收入 1238.24 元，支出 899.2 元，即+1238.24 和-889.2，输出收支情况，要求显示正负号，8 位宽度并保留两位小数，左对齐

(4) 给定一个字符串，输出字符串的前 3 个字符，前 18 个字符和整个字符串。

```
print("二进制{0:b}，八进制{0:o}，十进制{0:d}，十六进制{0:x}".format('陈'))
print("投中率 {:0>6.2%}".format(1/12))
print("收支情况: {:>+8.2f}, {:>+8.2f}" .format (+1238.24, -889.2))
url="www.pythonlearning.com"
print("{0:.3s},{0:.18s},{0:s}".format (url))
```

4.1.6.3 课后练习——个人名片

需求

- 在控制台依次提示用户输入：**姓名、公司、职位、电话、邮箱**
- 按照以下格式输出：

  ****

  公司名称

姓名 (职位)

电话：电话

邮箱：邮箱

****

实现代码如下：

```
"""
在控制台依次提示用户输入：姓名、公司、职位、电话、电子邮箱
"""
name = input("请输入姓名：")
company = input("请输入公司：")
title = input("请输入职位：")
phone = input("请输入电话：")
email = input("请输入邮箱：")

print("*" * 50)
print(company)
print()
print("{} ({})" .format(name, title))
print()
print("电话：{}" .format(phone))
print("邮箱：{}" .format(email))
print("*" * 50)
```

4.2 算数运算符

计算机，顾名思义就是负责进行**数学计算**并且**存储计算结果**的电子设备，因此算数运算是计算机的最基本功能。

4.2.1 算数运算符

(1) 算数运算符是**运算符的一种**

(2) 算数运算符是完成基本的算术运算使用的符号，用来处理四则运算

(3) 由运算符构成的式子，叫表达式，由算数运算符构成的式子，叫算数表达式

表 4-1 算数运算符

运算	描述	实例
+	加	10 + 20 = 30
-	减	10 - 20 = -10
*	乘	10 * 20 = 200
/	除	10 / 20 = 0.5
//	取整除	返回除法的整数部分（商） 9 // 2 输出结果 4
%	取余数	返回除法的余数 9 % 2 = 1
**	幂	又称次方、乘方，2 ** 3 = 8

- 在 Python 中*运算符还可以用于字符串，计算结果就是字符串重复指定次数的结果

```
>>> "-" * 50
'--------------------------------------------------'
```

★注意：

(1) //不对复数运算。整数商，即不大于 x 与 y 之商的最大整数。1//2 结果为 0，-1//2 的结果为-1。

(2) %不对复数运算。恒等式 x % y = x - (x // y) * y 。

(3) Python 规定 0**0 的值为 1，这也是编程语言的通用做法。

三种数字类型之间存在一种扩展关系：整数-> 浮点数 -> 复数。不同数字类型之间的运算所生成的结果是更宽的类型。

4.2.2 算数运算符的优先级

(1) 和数学中的运算符的优先级一致，在 Python 中进行数学计算时，同样也是：

1) **先乘除后加减**

2) 同级运算符是**从左至右**计算

3) 可以使用()调整计算的优先级

(2) 以下表格的算数优先级由高到最低顺序排列

表 4-2 算数运算符优先级

运算符	描述
**	幂 (最高优先级)
* / % //	乘、除、取余数、取整除
+ -	加法、减法

- 例如：

2 + 3 * 5 值为 17

(2 + 3) * 5 值为 25

2 * 3 + 5 值为 11

2 * (3 + 5) 值为 16

2**3+3*2 值为 14

4.3 其他运算符简介

Python 除了算术运算符之外，还有比较(关系)运算符、逻辑运算符、赋值运算符等。下面逐一进行简单介绍。

4.3.1 比较(关系)运算符

表 4-3 关系运算符

运算符	描述
==	检查两个操作数的值是否 **相等**，如果是，则条件成立，返回 True
!=	检查两个操作数的值是否 **不相等**，如果是，则条件成立，返回 True
>	检查左操作数的值是否 **大于** 右操作数的值，如果是，则条件成立，返回 True

表 4-3 关系运算符

运算符	描述
<	检查左操作数的值是否 **小于** 右操作数的值，如果是，则条件成立，返回 True
>=	检查左操作数的值是否 **大于或等于** 右操作数的值，如果是，则条件成立，返回 True
<=	检查左操作数的值是否 **小于或等于** 右操作数的值，如果是，则条件成立，返回 True

Python 2.x 中判断**不等于**还可以使用 <> 运算符，!= 在 Python 2.x 中同样可以用来判断**不等于。**

4.3.2 逻辑运算符

表 4-4 逻辑运算符

运算符	逻辑表达式	描述
and	x and y	只有 x 和 y 的值都为 True，才会返回 True 否则只要 x 或者 y 有一个值为 False，就返回 False
or	x or y	只要 x 或者 y 有一个值为 True，就返回 True 只有 x 和 y 的值都为 False，才会返回 False
not	not x	如果 x 为 True，返回 False 如果 x 为 False，返回 True

4.3.3 赋值运算符

(1) 在 Python 中，使用 = 可以给变量赋值

(2) 在算术运算时，为了简化代码的编写，Python 还提供了一系列的与**算术运算符**对应的**赋值运算符**

(3) 注意：**赋值运算符中间不能使用空格**

表 4-5 赋值运算符

运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a

表 4-5 赋值运算符

运算符	描述	实例
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
//=	取整除赋值运算符	c //= a 等效于 c = c // a
%=	取 **模** (余数)赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c *= *a* 等效于 *c* = *c* * a

此外赋值运算符中还有两个重要的知识点：

(1) 同步赋值：

```
>>> a, b = 4, 6
>>> print(a, b)
4 6
>>> a, b = b, a
>>> print(a, b)
6 4
```

(2) 复合赋值语句与普通赋值语句的比较：

```
>>> x = 3
>>> x *= 3 + 4
>>> x
21
>>> y = 3
>>> y = y * 3 + 4
>>> y
13
```

这些运算符，后面用到的时候，进一步深入学习。

4.3.4 运算符的优先级

以下表格的算数优先级由高到最低顺序排列

表 4-6 运算符优先级

运算符	描述
**	幂 (最高优先级)
* / % //	乘、除、取余数、取整除
+ -	加法、减法
<= < > >=	比较运算符
== !=	等于运算符
= %= /= //= -= += = *=	赋值运算符
not or and	逻辑运算符

4.4 常用内置函数

4.4.1 数学内置函数

表 4-7 数学内置函数

函数	描述	实例	实例结果
abs(a)	求取绝对值	abs(-1)	1
max(list)	求取 list 最大值	max(1,2,3)	3
min(list)	求取 list 最小值	min(1,2,3)	1
sum(list)	求取 list 元素的和	sum([1,2,3])	6
divmod(a,b)	获取商和余数	divmod(5,2)	(2,1)
pow(a,b)	获取乘方数	pow(2,3)	8
round(a,b)	获取指定位数的小数	round(3.1415926,2)	3.14
range(a[,b])	生成一个 a 到 b 的数组,左闭右开	range(1,10)	[1,2,3,4,5,6,7,8,9]
sorted(list)	排序，返回排序后的 list		

表 4-7 数学内置函数

函数	描述	实例	实例结果
len(list)	list 长度	len([1,2,3])	3

下面对最常用的几个数学内置函数，进一步举例说明。

(1) abs：求数值的绝对值

```
>>> abs(-2)
2
```

(2) divmod：返回两个数值的商和余数

```
>>> divmod(5,2)
(2, 1)
>> divmod(5.5,2)
(2.0, 1.5)
```

(3) max：返回可迭代对象中的元素中的最大值或者所有参数的最大值

```
>>> max(1,2,3) # 传入 3 个参数 取 3 个中较大者
3
>>> max('1234') # 传入 1 个可迭代对象，取其最大元素值
'4'
>>> max(-1,0) # 数值默认去数值较大者
0
>>> max(-1,0,key = abs) # 传入了求绝对值函数，则参数都会进行求绝对值后再取较大者
-1
```

(4) min：返回可迭代对象中的元素中的最小值或者所有参数的最小值

```
>>> min(1,2,3) # 传入 3 个参数 取 3 个中较小者
```

```
1
>>> min('1234') # 传入 1 个可迭代对象，取其最小元素值
'1'
>>> min(-1,-2) # 数值默认去数值较小者
-2
>>> min(-1,-2,key = abs)  # 传入了求绝对值函数，则参数都会进行求绝对值后再取较小者
-1
```

(5) pow：返回两个数值的幂运算值或其与指定整数的模值

```
>>> pow(2,3)
8
>>> 2**3
8
>>> pow(2,3,5)
3
>>> pow(2,3)%5
3
```

(6) round：对浮点数进行四舍五入求值

```
>>> round(1.1314926,1)
1.1
>>> round(1.1314926,5)
1.13149
```

(7) sum：对元素类型是数值的可迭代对象中的每个元素求和

```
# 传入可迭代对象
>>> sum((1,2,3,4))
```

```
10
# 元素类型必须是数值型
>>> sum((1.5,2.5,3.5,4.5))
12.0
>>> sum((1,2,3,4),-10)
0
```

(8) range：根据传入的参数创建一个新的 range 对象

```
>>> a = range(10)
>>> b = range(1,10)
>>> c = range(1,10,3)
>>> a,b,c # 分别输出 a,b,c
(range(0, 10), range(1, 10), range(1, 10, 3))
>>> list(a),list(b),list(c) # 分别输出 a,b,c 的元素
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [1, 2, 3, 4, 5, 6, 7, 8, 9], [1, 4, 7])
>>>
```

从(8)可见，range(1,10)在 Python3.x 以上版本会显示为 range(1,10)，这是生成器，可以用 list(range(1,10))看到数组结果。

4.4.2 类型转换内置函数

表 4-8 类型转换内置函数

函数	描述	实例	实例结果
int(str)	转换为 int 型	int('168')	168
float(int/str)	将 int 或字符型转换为浮点型	float('2')	2.0
str(int)	转换为字符型	str(134)	'134'
bool(int)	转换为布尔类型	str(0)	False

表 4-8 类型转换内置函数

函数	描述	实例	实例结果
bytes(str,code)	接收一个字符串，与所要编码的格式，返回一个字节流类型	bytes('abc', 'utf-8') bytes(u'爬虫', 'utf-8')	b'abc' b'\xe7\x88\xac\xe8\x99\xab'
hex(int)	转换为 16 进制	hex(1024)	'0x400'
oct(int)	转换为 8 进制	oct(1024)	'0o2000'
bin(int)	转换为 2 进制	bin(1024)	'0b10000000000'
chr(int)	转换数字为相应 ASCI 码字符	chr(65)	'A'
ord(str)	转换 ASCI 字符为相应的数字	ord('A')	65

下面对类型转换内置函数，进一步举例说明。

(1) bool：根据传入的参数的逻辑值创建一个新的布尔值

```
>>> bool() #未传入参数
False
>>> bool(0) #数值 0、空序列等值为 False
False
>>> bool(1)
True
```

(2) int：根据传入的参数创建一个新的整数

```
>>> int() #不传入参数时，得到结果 0。
0
>>> int(3)
3
>>> int(3.6)
3
```

(3) float：根据传入的参数创建一个新的浮点数

```
>>> float() #不提供参数的时候，返回 0.0
0.0
>>> float(3)
3.0
>>> float('3')
3.0
```

(4) complex：根据传入参数创建一个新的复数

```
>>> complex() #当两个参数都不提供时，返回复数 0j。
0j
>>> complex('1+2j') #传入字符串创建复数
(1+2j)
>>> complex(1,2) #传入数值创建复数
(1+2j)
```

(5) str：返回一个对象的字符串表现形式(给用户)

```
>>> str()
''
>>> str(None)
'None'
>>> str('abc')
'abc'
>>> str(123)
'123'
```

(6) bytes：根据传入的参数创建一个新的不可变字节数组

```
>>> bytes('中文','utf-8')
```

```
b'\xe4\xb8\xad\xe6\x96\x87'
```

(7) ord：返回 Unicode 字符对应的整数

```
>>> ord('a')
97
```

(8) chr：返回整数所对应的 Unicode 字符

```
>>> chr(97) #参数类型为整数
'a'
```

(9) bin：将整数转换成 2 进制字符串

```
>>> bin(3)
'0b11'
```

(10) oct：将整数转化成 8 进制数字符串

```
>>> oct(10)
'0o12'
```

(11) hex：将整数转换成 16 进制字符串

```
>>> hex(15)
'0xf'
```

4.5 常用标准库函数

4.5.1 math 库

math 库是 Python 提供的内置数学类函数库，不支持复数运算。math 库中的函数不能直接使用，需要使用 import 导入该库，导入的方法有两种：

(1) import math

以 math.函数名()形式调用函数。（建议，不会覆盖内置函数）

(2) from math import 函数名 1 [,函数名 2,...]

直接以函数名()的方式调用。特殊地，使用 from math import *，math 库的所有函数都可以直接使用。

实际上，所有的函数库的导入都可以自由选择这两种方式。

除了明确的说明，math 库函数的返回值为浮点数。

math 库包括 acos(), sin()等函数，可以用 dir(math)查看包含那些数学函数和常量：

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

math 库常用函数举例如下：

(1) pi:数字常量，圆周率

```
>>> print(math.pi)
3.141592653589793
```

(2) e 表示一个常量

```
>>> math.e
2.718281828459
```

(3) ceil()取大于等于 x 的最小的整数值，如果 x 是一个整数，则返回 x

```
>>> math.ceil(4.12)
5
```

(4) floor()取小于等于 x 的最大的整数值，如果 x 是一个整数，则返回自身

```
>>> math.floor(4.999)
4
```

(5) trunc()返回 x 的整数部分

```
>>> math.trunc(6.789)
6
```

(6) copysign()把 y 的正负号加到 x 前面，可以使用 0

```
>>> math.copysign(2,-3)
-2.0
```

(7) cos()求 x 的余弦，x 必须是弧度

```
>>> math.cos(math.pi/4)
0.7071067811865476
```

(8) sin()求 x(x 为弧度)的正弦值

```
>>> math.sin(math.pi/4)
0.7071067811865476
```

(9) tan()返回 x(x 为弧度)的正切值

```
>>> math.tan(math.pi/4)
0.9999999999999999
```

(10) degrees 把 x 从弧度转换成角度

```
>>> math.degrees(math.pi/4)
45.0
>>> math.degrees(3.14)
179.9087476710785
```

```
045
```

(11) radians()把角度 x 转换成弧度

```
>>> math.radians(45)
0.7853981633974483
>>> math.radians(180)
3.141592653589793
```

(12) sqrt()求 x 的平方根

```
>>> math.sqrt(100)
10.0
>>> math.sqrt(4)
2.0
```

(13) pow()返回 x 的 y 次方，即 x**y

```
>>> math.pow(3,4)
81.0
```

(14) log(x,a) 如果不指定 a，则默认以 e 为基数，a 参数给定时，将 x 以 a 为底的对数返回。

```
>>> math.log(math.e)
1.0
>>> math.log(32,2)
5.0
>>>
```

(15) log10()返回 x 的以 10 为底的对数

```
>>> math.log(10)
```

```
2.302585092994046
```

(16) log2()返回 x 的基 2 对数

```
>>> math.log2(32)
5.0
```

(17) exp()返回 math.e(其值为 2.71828)的 x 次方

```
>>> math.exp(2)
7.38905609893065
```

(18) expm1()返回 math.e 的 x(其值为 2.71828)次方的值减 1

```
>>> math.expm1(2)
6.38905609893065
```

(19) fabs()返回 x 的绝对值

```
>>> math.fabs(-0.03)
0.03
```

(20) factorial()取 x 的阶乘的值

```
>>> math.factorial(3)
6
```

(21) fmod()得到 x/y 的余数，其值是一个浮点数

```
>>> math.fmod(20,3)
2.0
```

(22) fsum()对括号里的每个元素进行求和操作

```
>>> math.fsum((1,2,3,4))
10.0
```

(23) gcd()返回 x 和 y 的最大公约数

```
>>> math.gcd(8,6)
2
```

(24) hypot()得到(x^2+y^2)的平方根的值

```
>>> math.hypot(3,4)
5.0
```

(25) isfinite()如果 x 不是无穷大的数字,则返回 True,否则返回 False

```
>>> math.isfinite(0.1)
True
```

(26) isinf()如果 x 是正无穷大或负无穷大，则返回 True,否则返回 False

```
>>> math.isinf(234)
False
```

(27) isnan()如果 x 不是数字 True,否则返回 False

```
>>> math.isnan(23)
False
```

(28) ldexp()返回 x*(2**i)的值

```
>>> math.ldexp(5,5)
160.0
```

(29) modf()返回由 x 的小数部分和整数部分组成的元组

```
>>> math.modf(math.pi)
(0.14159265358979312, 3.0)
```

4.5.2 random 库

内置库 random 库，是 Python 常用的随机库，random 库产生随机数使用随机数种子函数 seed()来产生(只要种子相同，产生的随机序列，无论是每一个数，还是数与数之间的关系都是确定的，所以随机数种子确定了随机序列的产生), seed()初始化给定的随机数种子，默认为当前系统时间。random 库常用的函数包括产生 0 到 1 之间的小数的 random()函数，列表中随机选择一个元素的 choice()函数，产生随机整数的 randint()函数， 产生正态分布的 uniform()函数，随机取样的 sample()函数，随机乱序的 shuffle()函数，生成一个 k 比特长的随机整数等。

```
>>> import random
>>> random.seed()
>>> random.choice(['C++','Java','Python'])
'Java'
>>> random.randint(1,100)
57
>>> random.randrange(0,10,2)
8
>>> random.random()
0.7906454183842933
>>> random.uniform(5,10)
7.753307224388041
>>> random.sample(range(100),10)
[91, 15, 67, 38, 55, 72, 62, 97, 51, 77]
>>> nums=[1001,1002,1003,1004,1005]
>>> random.shuffle(nums)
>>> nums
[1002, 1004, 1005, 1001, 1003]
>>>random.getrandbits(16)
```

```
64636
>>>
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom', 'TWOPI', '_BuiltinMethodType', '_MethodType', '_Sequence', '_Set',
'__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_inst', '_itertools', '_log',
'_pi', '_random', '_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn',
'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate',
'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random', 'randrange', 'sample',
'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
```

习 题

一、填空题

1、Python 标准库 math 中用来计算平方根的函数是_____。

2、查看变量类型的 Python 内置函数是_____。

3、以 3 为实部 4 为虚部，Python 复数的表达形式为_____或_____。

4、Python 运算符中用来计算整商的是_____。

5、查看变量内存地址的 Python 内置函数是_____。

6、已知 x = 3，那么执行语句 x += 6 之后，x 的值为_____。

7、已知 x = 3，并且 id(x)的返回值为 496103280，那么执行语句 x += 6 之后，表达式 id(x) == 496103280 的值为_____。

8、已知 x = 3，那么执行语句 x *= 6 之后，x 的值为_____。

9、表达式 int('123', 16) 的值为_____。

10、表达式 int('123', 8) 的值为_____。

11、表达式 int('123') 的值为_____。

12、表达式 int('101',2) 的值为_____。

13、表达式 abs(-3) 的值为_____。

14、表达式 int(4**0.5) 的值为_____。

15、Python 内置函数_____用来返回序列中的最大元素。

16、Python 内置函数_____用来返回序列中的最小元素。

17、Python 内置函数_____用来返回数值型序列中所有元素之和。

18、已知 x=3 和 y=5，执行语句 x, y = y, x 后 x 的值是____。

19、表达式 3<5>2 的值为_____。

20、表达式 1<2<3 的值为_____。

21、表达式 3 or 5 的值为_____。

22、表达式 0 or 5 的值为_____。

23、表达式 3 and 5 的值为_____。

24、表达式 3 and not 5 的值为_____。

25、表达式 3 | 5 的值为_____。

26、表达式 3 & 6 的值为_____。

27、表达式 3 ** 2 的值为_____。

28、表达式 3 * 2 的值为_____。

29、表达式 chr(ord('a')-32) 的值为_____。

30、表达式 abs(3+4j) 的值为_____。

31、表达式 round(3.4) 的值为_____。

32、表达式 round(3.7) 的值为_____。

33、表达式 type(3) in (int, float, complex) 的值为_____。

34、表达式 type(3.0) in (int, float, complex) 的值为_____。

35、表达式 type(3+4j) in (int, float, complex) 的值为_____。

36、表达式 type('3') in (int, float, complex) 的值为_____。

37、表达式 type(3) == int 的值为_____。

38、表达式 eval('__import__('math').sqrt(9)') 的值为_____。

39、表达式 eval('__import__('math').sqrt(3**2+4**2)') 的值为_____。

40、表达式 eval('3+5') 的值为_____。

41、假设 math 标准库已导入，那么表达式 eval('math.sqrt(4)') 的值为_____。

42、表达式 not 3 的值为_____。

43、表达式 3 // 5 的值为_____。

44、表达式 isinstance(4j, (int, float, complex)) 的值为_____。

45、表达式 isinstance('4', (int, float, complex)) 的值为_____。

二、判断题

1、加法运算符可以用来连接字符串并生成新字符串。（）

2、9999**9999 这样的命令在 Python 中无法运行。（）

3、3+4j 不是合法的 Python 表达式。（）

4、0o12f 是合法的八进制数字。（）

5、3+4j 是合法 Python 数字类型。（）

6、在 Python 中 0oa1 是合法的八进制数字表示形式。（）

三、程序题

1、输入三位整数，分解出个位、十位和百位并打印输出。

2、仅使用 Python 基本语法，即不使用任何模块，编写 Python 程序计算下列数学表达式的结果并输出，小数点后保留 2 位。$x = \frac{\sqrt{4^2+2\times7^3}}{4}$

3、0x660E 是一个十六进制数，它对应的 Unicode 字符是汉字“明”，请输出汉字“明”对应的 Unicode 字符编码的二进制、十进制、八进制和十六进制格式。即：

print("二进制{____①____}、十进制{____②____}、八进制{____③____}、十六进制{____④____}".format(____⑤____))。

提示 1：format 中可以用 ord('明')或者 0x660E，即 format(0x660E)。

提示 2：{}中可以在“:”前指定序号，本例中可以指定为 0，即{0:***}。

第 5 章 判断语句

知识点

(1) 开发中的应用场景

(2) if 语句体验

(3) if 语句进阶

(4) 程序的格式框架

(5) 综合应用

5.1 开发中的应用场景

生活中的判断几乎是无所不在的，我们每天都在做各种各样的选择，如果这样？如果那样？……

人生的关键点就是一场场选择……

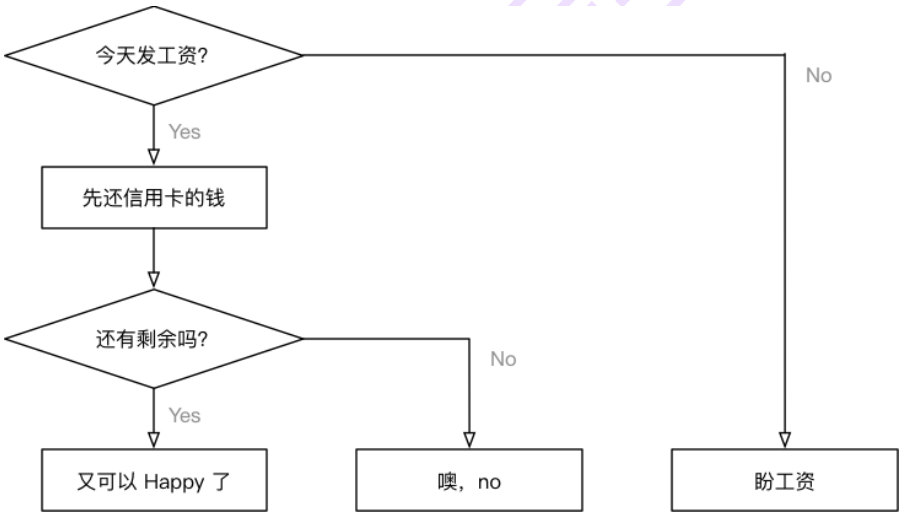


图 5-1 无处不在的选择

5.1.1 程序中的判断

if 今天发工资:

先还信用卡的钱

if 有剩余:

又可以 happy 了，O(∩_∩)O 哈哈~

else:

噢，no。。。还的等 30 天

else:

盼着发工资

5.1.2 判断的定义

(1) 如果**条件满足**，才能做某件事情，

(2) 如果**条件不满足**，就做另外一件事情，或者什么也不做

正是因为有了判断，才使得程序世界丰富多彩，充满变化！

判断语句又被称为 “分支语句”，正是因为有了判断，才让程序有了很多的分支

5.2. if 语句体验

5.2.1 if 判断语句基本语法

在 Python 中，**if 语句** 就是用来进行判断的，格式如下：

if 要判断的条件:

条件成立时，要做的事情

……

★注意：代码的缩进为一个 tab 键，或者 **4 个空格**——**建议使用空格**

- 在 Python 开发中，Tab 和空格不要混用！

我们可以把整个 if 语句看成一个完整的代码块

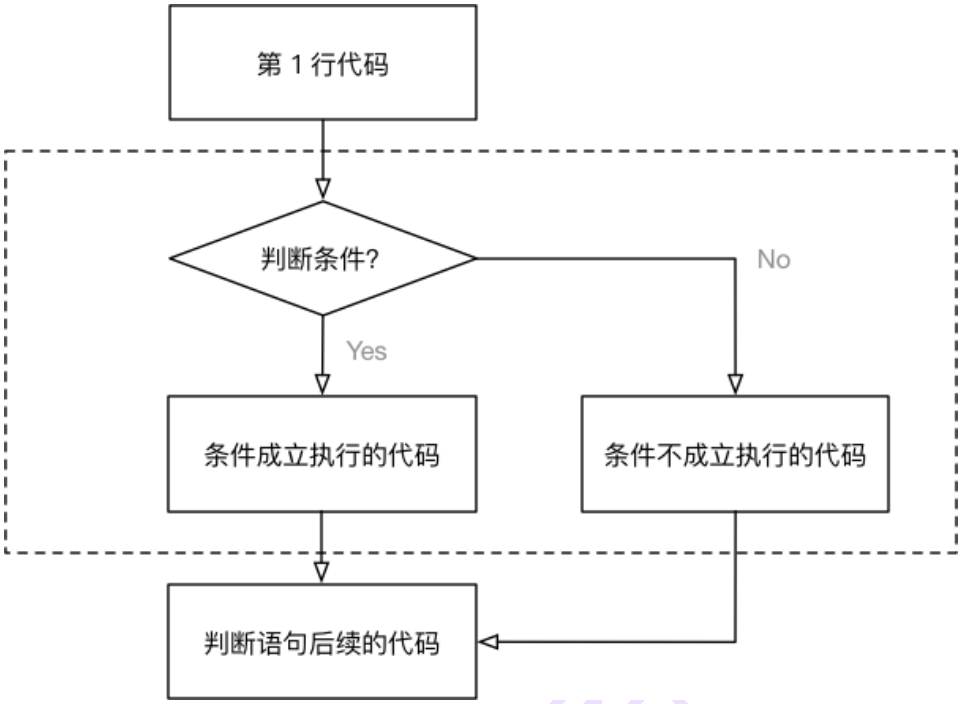


图 5-2 选择结构

5.2.2 判断语句演练——判断年龄

需求：

(1) 定义一个整数变量记录年龄

(2) 判断是否满 18 岁（>=）

(3) 如果满 18 岁，允许进网吧嗨皮

```
# 1. 定义年龄变量
age = 18


# 2. 判断是否满 18 岁
# if 语句以及缩进部分的代码是一个完整的代码块
if age >= 18:
    print("可以进网吧嗨皮……")


# 3. 思考！- 无论条件是否满足都会执行
```

```
print("这句代码什么时候执行?")
```

★注意：

- if 语句以及缩进部分是一个**完整的代码块**

5.2.3 else 处理条件不满足的情况

思考：

在使用 if 判断时，只能做到满足条件时要做的事情。那如果需要在**不满足条件的时候**，做某些事情，该如何做呢？

答案：

else 格式如下

if 要判断的条件:

条件成立时，要做的事情

……

else:

条件不成立时，要做的事情

……

★注意：

- if 和 else 语句以及各自的缩进部分共同是一个**完整的代码块**

5.2.4 判断语句演练——判断年龄改进

需求：

(1) 输入用户年龄

(2) 判断是否满 18 岁 （>=）

(3) 如果满 18 岁，允许进网吧嗨皮

(4) 如果未满 18 岁，提示回家写作业

```
# 1. 输入用户年龄
age = int(input("今年多大了？"))
```

```
# 2. 判断是否满 18 岁
# if 语句以及缩进部分的代码是一个完整的语法块
if age >= 18:
    print("可以进网吧嗨皮……")
else:
    print("你还没长大，应该回家写作业！")

# 3. 思考！- 无论条件是否满足都会执行
print("这句代码什么时候执行?")
```

5.3 逻辑运算

(1) 在程序开发中，通常**在判断条件时**，会需要同时判断多个条件

(2) 只有多个条件都满足，才能够执行后续代码，这个时候需要使用到**逻辑运算符**

(3) **逻辑运算符**可以把**多个条件**按照**逻辑**进行**连接**，变成**更复杂的条件**

(4) Python 中的**逻辑运算符**包括：**与 and** / **或 or** / **非 not** 三种

5.3.1 and

条件 1 and 条件 2

(1) **与** / 并且

(2) 两个条件同时满足，返回 True

(3) 只要有一个不满足，就返回 False

表 5-1 and

条件 1	条件 2	结果
成立	成立	成立
成立	不成立	不成立
不成立	成立	不成立
不成立	不成立	不成立

5.3.2 or

条件 1 or 条件 2

(1) **或 /** 或者

(2) 两个条件只要有一个满足，返回 True

(3) 两个条件都不满足，返回 False

表 5-2 or

条件 1	条件 2	结果
成立	成立	成立
成立	不成立	成立
不成立	成立	成立
不成立	不成立	不成立

5.3.3 not

not 条件

- **非 /** 不是

表 5-3 not

条件	结果
成立	不成立
不成立	成立

5.3.4 逻辑运算演练

(1) 练习 1: 定义一个整数变量 age，编写代码判断年龄是否正确

- 要求人的年龄在 0-120 之间

(2) 练习 2: 定义两个整数变量 python_score、c_score，编写代码判断成绩

- 要求只要有一门成绩> 60 分就算合格

(3) 练习 3: 定义一个布尔型变量 is_employee，编写代码判断是否是本公司员工

- 如果不是提示不允许入内

答案 1：

```
# 练习 1: 定义一个整数变量 age，编写代码判断年龄是否正确
age = 100
```

```
# 要求人的年龄在 0-120 之间
if age >= 0 and age <= 120:
    print("年龄正确")
else:
    print("年龄不正确")
```

答案 2：

```
# 练习 2: 定义两个整数变量 python_score、c_score，编写代码判断成绩
python_score = 50
c_score = 50

# 要求只要有一门成绩 > 60 分就算合格
if python_score > 60 or c_score > 60:
    print("考试通过")
else:
    print("再接再厉！")
```

答案 3：

```
# 练习 3: 定义一个布尔型变量 `is_employee`，编写代码判断是否是本公司员工
is_employee = True

# 如果不是提示不允许入内
if not is_employee:
    print("非公勿内")
```

5.4 if 语句进阶

5.4.1 elif

(1) 在开发中，使用 if 可以**判断条件**

(2) 使用 else 可以处理**条件不成立**的情况

(3) 但是，如果希望**再增加一些条件，条件不同，需要执行的代码也不同**时，就可以使用 elif

(4) 语法格式如下：

```
if 条件 1:
    条件 1 满足执行的代码
    ……
elif 条件 2:
    条件 2 满足时，执行的代码
    ……
elif 条件 3:
    条件 3 满足时，执行的代码
    ……
else:
    以上条件都不满足时，执行的代码
    ……
```

- 对比逻辑运算符的代码

```
if 条件 1 and 条件 2:
    条件 1 满足 并且 条件 2 满足 执行的代码
    ……
```

★注意

(1) elif 和 else 都必须和 if 联合使用，而不能单独使用

(2) 可以将 if、elif 和 else 以及各自缩进的代码，看成一个**完整的代码块**

elif 演练 1——女友的节日

需求：

(1) 定义 holiday_name 字符串变量记录节日名称

(2) 如果是**情人节**应该**买玫瑰 / 看电影**

(3) 如果是**平安夜**应该**买苹果 / 吃大餐**

(4) 如果是**生日**应该**买蛋糕**

(5) 其他的日子每天都是节日啊……

```
holiday_name = "平安夜"


if holiday_name == "情人节":
    print("买玫瑰")
    print("看电影")
elif holiday_name == "平安夜":
    print("买苹果")
    print("吃大餐")
elif holiday_name == "生日":
    print("买蛋糕")
else:
    print("每天都是节日啊……")
```

elif 演练 2——分段函数

需求：

企业发放的奖金根据利润提成。利润低于或等于 10 万元时，奖金可提 10%；利润高于 10 万元，低于 20 万元时，低于 10 万元的部分按 10%提成，高于 10 万元的部分，可提成 7.5%；20 万到 40 万之间时，高于 20 万元的部分，可提成 5%；40 万到 60 万之间时高于 40 万元的部分，可提成 3%；60 万到 100 万之间时，高于 60 万元的部分，可提成 1.5%，高于 100 万元时，超过 100 万元的部分按 1%提成。从键盘输入当月利润，求应发放奖金总数？

```
profit = int(input("请输入当月利润（万元）："))
if profit<=10:
        reward = profit*0.1
elif profit<=20:
      reward = (profit-10)*0.075+1
elif profit<=40:
      reward = (profit-20)*0.05+10*0.1+10*0.075
elif profit<=60:
      reward = (profit-40)*0.03+20*0.05+10*0.075+10*0.1
elif profit<=100:
      reward = (profit-60)*0.015+20*0.03+20*0.05+10*0.075+10*0.1
elif profit>100:
    reward = (profit-100)*0.01+40*0.015+20*0.03+20*0.05+10*0.075+10*0.1
print ("应发放奖金总数:",reward(profit)*10000, "(元)")
```

对于数学上称为分段函数这种类型的问题，可以采用从小到大或者从大到小的分割方法。比如从小到大，先分割出 10 万元以内的，然后分割 20 万元以内的(profit<=20 and profit>10)，分割 20 万元以内的时候，10 万元已经被分割掉了，不再需要 and profit>10，后面的可以以此类推。这种采用 if~elif 从小到大或者从大到小的分割方法，可以让编程看起来更加简单明了。

5.4.2 if 的嵌套

前面学习了 elif，elif 的应用场景是：**同时**判断**多个条件**，所有的条件是**平级**的。

但是，在开发中，使用 if 进行条件判断，如果希望**在条件成立的执行语句中**再**增加条件判断**，就可以使用 **if 的嵌套**

if 的嵌套的应用场景就是：**在之前条件满足的前提下，再增加额外的判断。if 的嵌套**的语法格式，**除了缩进之外**和之前的没有区别。

if 的嵌套语法格式如下：

if 条件 1:

条件 1 满足执行的代码

……

if 条件 1 基础上的条件 2:

条件 2 满足时，执行的代码

……

 # 条件 2 不满足的处理

else:

条件 2 不满足时，执行的代码

 # 条件 1 不满足的处理

else:

条件 1 不满足时，执行的代码

……

if 的嵌套演练——火车站安检

需求：

(1) 定义布尔型变量 has_ticket 表示是否有车票

(2) 定义整型变量 knife_length 表示刀的长度，单位：厘米

(3) 首先检查是否有车票，如果有，才允许进行**安检**

(4) 安检时，需要检查刀的长度，判断是否超过 20 厘米

1) 如果超过 20 厘米，提示刀的长度，不允许上车

2) 如果不超过 20 厘米，安检通过

(5) 如果没有车票，不允许进门

```
# 定义布尔型变量 has_ticket 表示是否有车票
has_ticket = True
# 定义整数型变量 knife_length 表示刀的长度，单位：厘米
knife_length = 20
```

```
# 首先检查是否有车票，如果有，才允许进行 安检
if has_ticket:
    print("有车票，可以开始安检...")
    # 安检时，需要检查刀的长度，判断是否超过 20 厘米
    # 如果超过 20 厘米，提示刀的长度，不允许上车
    if knife_length >= 20:
        print("不允许携带 %d 厘米长的刀上车" % knife_length)
    # 如果不超过 20 厘米，安检通过
    else:
        print("安检通过，祝您旅途愉快……")
# 如果没有车票，不允许进门
else:
    print("大哥，您要先买票啊")
```

5.5 程序的格式框架

冒号和缩进：

(1) Python 语言采用严格的“缩进”来表明程序的格式框架。缩进指每一行代码开始前的空白区域，用来表示代码之间的**包含和层次关系**。

(2) 1 个缩进 = 4 个空格

缩进是 Python 语言中表明程序框架的**唯一手段**

(3) 当表达分支、循环、函数、类等程序含义时，在 if、while、for、def、class 等保留字所在完整语句后通过英文冒号（:）结尾并在之后进行缩进，表明后续代码与紧邻无缩进语句的所属关系。

```
TempStr = input("请输入带有符号
if TempStr[-1] in ['F','f']:
    C = (eval(TempStr[0:-1]) -
    print("转换后的温度是{:.2f}C
elif TempStr[-1] in ['C','c']
    F = 1.8*eval(TempStr[0:-1]
    print("转换后的温度是{:.2f}F
else:
    print("输入格式错误")
```

图 5-3 单层缩进

```
DARTS = 1000
hits = 0.0
clock()
for i in range(1, DARTS):
    x, y = random(), random()
    dist = sqrt(x ** 2 + y **
    if dist <= 1.0:
        hits = hits + 1
pi = 4 * (hits/DARTS)
print("Pi的值是{:.2f}".format
```

图 5-4 多层缩进

5.6 三元表达式

在 Python 中，除了使用 if 语句之外，对于比较简单的选择，还可以使用三元表达式(if else 构成的表达式)。三元表达式基本格式如下：

条件为真时的结果 if 判段的条件 else 条件为假时的结果

常见的应用如下：

```
# 求变量 a 和 b 的最大值并赋值给 x
```

```
a=3; b=4
x = a if a>b else b
print(x)


# 求变量 x 的符号 sign
x=-2
sign = 1 if x>0 else -1 if x<0 else 0
#加括号后看起来更清晰： sign = 1 if x>0 else (-1 if x<0 else 0)
print(sign)
#根据成绩给出优秀、及格和不及格
score=61
grade = "优秀" if score>80 else "及格" if score>=60 else "不及格"
print(grade)
```

运行结果：

```
4
-1
及格
```

5.7 综合应用——石头剪刀布

目标：

(1) 强化**多个条件**的**逻辑运算**

(2) 体会 import 导入模块（“工具包”）的使用

需求

(1) 从控制台输入要出的拳 —— 石头（1） / 剪刀（2） / 布（3）

(2) 电脑**随机**出拳 —— 先假定电脑只会出石头，完成整体代码功能

(3) 比较胜负

表 5-4 规则

序号	规则
1	石头 胜 剪刀
2	剪刀 胜 布
3	布 胜 石头

1 基础代码实现

- 先**假定电脑就只会出石头**，完成整体代码功能：

```
# 从控制台输入要出的拳 —— 石头（1） / 剪刀（2） / 布（3）
player = int(input("请出拳 石头（1） / 剪刀（2） / 布（3）： "))
# 电脑随机出拳 - 假定电脑永远出石头
computer = 1
# 比较胜负
# 如果条件判断的内容太长，可以在最外侧的条件增加一对大括号
# 再在每一个条件之间，使用回车，PyCharm 可以自动增加 8 个空格
if ((player == 1 and computer == 2) or
        (player == 2 and computer == 3) or
        (player == 3 and computer == 1)):
    print("噢耶！！！电脑弱爆了！！！")
elif player == computer:
    print("心有灵犀，再来一盘！")
else:
    print("不行，我要和你决战到天亮！")
```

2 随机数的处理

(1) 前面已经学过，在 Python 中，要使用随机数，首先需要导入**随机数**的**模块** —— “random 库”

```
import random
```

(2) 导入模块后，可以直接在**模块名称**后面敲一个. 然后按 Tab 键，会提示该模块中包含的所有函数

(3) random.randint(a, b) ，返回 [a, b] 之间的整数，包含 a 和 b

(4) 例如：

```
random.randint(12, 20)   # 生成的随机数 n: 12 <= n <= 20
random.randint(20, 20)   # 结果永远是 20
random.randint(20, 10)   # 该语句是错误的，下限必须小于上限
```

课后演练：把上例中的 computer = 1 改为 computer = random.randint(1,3)，要记得导入随机模块，即 import random。

习题

一、填空题

1、Python 关键字 elif 表示_____和_____两个单词的缩写。（else、if）

二、程序题

1、编写程序，运行后用户输入 4 位整数作为年份，判断其是否为闰年。如果年份能被 400 整除，则为闰年；如果年份能被 4 整除但不能被 100 整除也为闰年。

2、编写程序，实现分段函数计算，如下表所示。

表 5-5 分段函数

x	y
x<0	0
0<=x<5	x
5<=x<10	3x-5
10<=x<20	0.5x-2
20<=x	0

第 6 章 循环语句

知识点

(1) 程序的三大流程

(2) 循环基本使用

(3) break 、continue 和 else

(4) 循环嵌套

6.1 程序的三大流程

在程序开发中，一共有三种流程方式：

(1) **顺序**——**从上向下**，顺序执行代码

(2) **分支**——根据条件判断，决定执行代码的**分支**

(3) **循环**——让**特定代码重复**执行

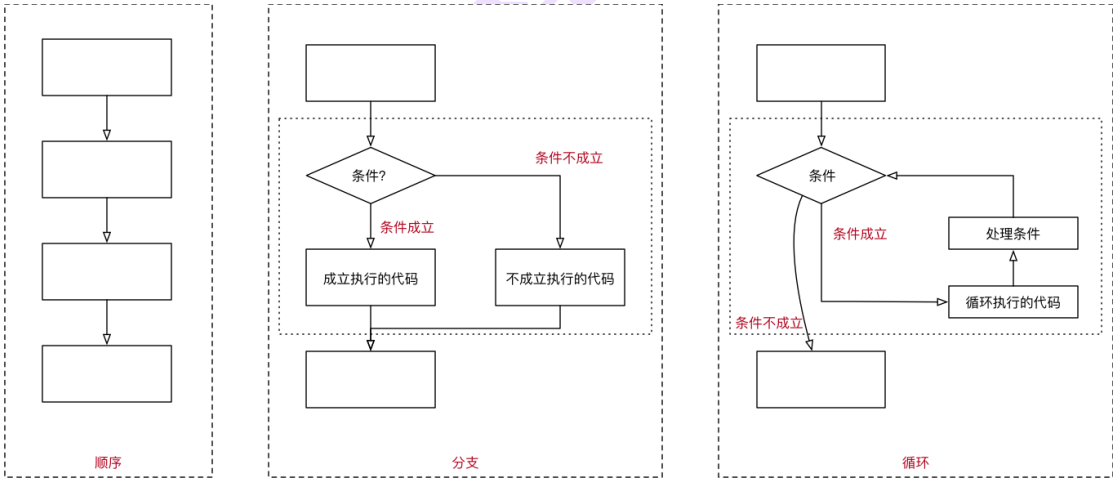


图 6-1 三种流程

6.2 循环基本使用

(1) 循环的作用就是让**指定的代码**重复的执行

(2) 循环最常用的应用场景就是**让执行的代码**按照**指定的次数重复**执行

- 需求——打印 5 遍 Hello Python
- 思考——如果要求打印 100 遍怎么办？

6.2.1 while 和 for 语句基本语法

1. while 循环格式

初始条件设置 —— 通常是重复执行的 计数器

while 条件(判断 计数器 是否达到 目标次数):

条件满足时，做的事情 1

条件满足时，做的事情 2

条件满足时，做的事情 3

...(省略)...

处理条件(计数器 + 1)

2. for 循环格式

for 变量 in 序列:

条件满足时，做的事情 1

条件满足时，做的事情 2

条件满足时，做的事情 3

...(省略)...

★注意：

- while 或者 for 语句以及缩进部分是一个**完整的代码块**

第一个循环

需求：

- 打印 5 遍 Hello Python

使用 while 循环:

```
#while 循环
# 1. 定义重复次数计数器
i = 1
# 2. 使用 while 判断条件
while i <= 5:
    # 要重复执行的代码
```


```
        print("Hello Python")
        # 处理计数器 i
        i = i + 1
print("循环结束后的 i ={} ".format( i))
```

使用 for 循环：

```
#for 循环
#使用 for in 循环
for i in range(5):
        # 要重复执行的代码
        print("Hello Python")
print("循环结束后的 i ={} ".format( i))
```

★注意：循环结束后，之前定义的计数器条件的数值是依旧存在的。

下面程序输出由（*）组成的菱形图案：

```
def main(n):
        for i in range(n):
                print(("*" * i).center(n * 3))
        for i in range(n,0,-1)
                print(("*" * i).center(n * 3))
main(6)
```

运行结果：

```
          *
         **
        ***
       ****
      *****
```

```
******
*****
 ****
 ***
  **
  *
```

6.2.1 死循环

由于程序员的原因，**忘记**在循环内部**修改循环的判断条件**，导致循环持续执行，程序无法终止！

```
# while 死循环
while 2:
    pass
#pass 关键字用在类或函数的定义中或者选择结构中，表示空语句，即执行时什么都不发生
```

6.2.2 Python 中的计数方法

常见的计数方法有两种，可以分别称为：

(1) **自然计数法**（从 1 开始）—— 更符合人类的习惯

(2) **程序计数法**（从 0 开始）—— 几乎所有的程序语言都选择从 0 开始计数

因此，大家在编写程序时，应该尽量养成习惯：**除非需求的特殊要求，否则循环的计数都从 0 开始**

6.2.3 循环计算

在程序开发中，通常会遇到**利用循环重复计算**的需求

遇到这种需求，可以：

(1) 在 while 上方定义一个变量，用于**存放最终计算结果**

(2) 在循环体内部，每次循环都用**最新的计算结果**，**更新**之前定义的变量

需求

- 计算 0 ~ 100 之间所有数字的累计求和结果

使用 while 循环:

```
# 计算 0 ~ 100 之间所有数字的累计求和结果
# 0. 定义最终结果的变量
result = 0
# 1. 定义一个整数的变量记录循环的次数
i = 0
# 2. 开始循环
while i <= 100:
    print(i)
    # 每一次循环，都让 result 这个变量和 i 这个计数器相加
    result += i
    # 处理计数器
    i += 1
print("0~100 之间的数字求和结果{}".format( result))
```

使用 for 循环：

```
# 计算 0 ~ 100 之间所有数字的累计求和结果
# 0. 定义最终结果的变量
result = 0
#1. 开始循环
for i in range(101):
    print(i)
    # 每一次循环，都让 result 这个变量和 i 这个计数器相加
    result += i
print("0~100 之间的数字求和结果{}".format( result))
```

需求进阶 1

计算 0 ~ 100 之间所有**偶数**的累计求和结果

开发步骤

(1) 编写循环**确认要计算的数字**

(2) 添加**结果**变量，在循环内部**处理计算结果**

使用 while 循环:

```
# 0. 最终结果
result = 0
# 1. 计数器
i = 0
# 2. 开始循环
while i <= 100:
    # 判断偶数
    if i % 2 == 0:
        print(i)
        result += i
    # 处理计数器
    i += 1
print("0~100 之间偶数求和结果 = %d" % result)
```

使用 for 循环：

```
# 0. 最终结果
result = 0
# 1. 开始循环
for i in range(101):
    # 判断偶数
    if i % 2 == 0:
```

```
        print(i)
        result += i
print("0~100 之间偶数求和结果 = %d" % result)
```

这个问题，也可以不使用 if 语句，而是每次 i 增加 2，使用 for 循环如下：

```
# 0. 最终结果
result = 0
# 1. 开始循环
for i in range(0,101,2):   #range 的最后一个参数 2，表示每次增加 2
    result += i
print("0~100 之间偶数求和结果 = %d" % result)
```

需求进阶 2

计算 S=1-1/3+1/5-1/7+1/9-…+1/101

开发步骤

(1) 编写循环**确认要计算的数字**:1,3,5,7…,101

(2) 计算通项

(3) 添加**结果**变量，在循环内部**处理计算结果**

(4) 处理正负号的循环变化：循环中，可以使用 sign=-sign，使得每次正负号变化

这个例子，这里只使用 for 循环：

```
# 0. 初始化最终结果、正负号变化变量 sign
S = 0
sign = 1
# 1. 开始循环
for i in range(1,102,2):
    term = sign/i      # 计算通项
    S = S+ term      #累加
    sign=-sign   #处理正负号的循环变化
```

```
print("S=1-1/3+1/5-1/7+1/9-…+1/101 = {}".format(S) )
```

归纳总结

高中数学中的递推公式 $S_n=S_{n-1}+(-1)^n*A_n$ 或 $S_n=S_{n-1}*(-1)^n A_n$ 这一类问题怎么解决？

```
# 0. 初始化 S 最终结果、正负号变化变量 sign 等
n = 100
S = 0
sign = 1
# 1. 开始循环
for i in range(n):   #根据通项的情况，可以进一步细化，如上面例子中的 range(1,102,2)
    #下面循环体内的三步，顺序可以根据具体情况改变
    term = sign*…      # 根据具体情况计算通项
    s=s+ term      #累加或者累积
    sign=-sign   #处理正负号的循环变化
print("S = {}" .format(S) )
```

6.3 break、continue 和 else

break 和 continue 是专门在循环中使用的关键字。此外循环中还有 else 的用法。

(1) break **某一条件满足时**，退出循环，不再执行后续重复的代码

(2) continue **某一条件满足时**，不执行后续重复的代码

break 和 continue 只针对**当前所在循环**有效

6.3.1 break

在循环过程中，如果 **某一个条件满足后，不再希望循环继续执行**，可以使用 break 退出循环

```
i = 0
while i < 10:
    # break 某一条件满足时，退出循环，不再执行后续重复的代码
    # i == 3
```

```
    if  i == 3:
        break
    print(i)
    i += 1
print("over")
```

课后演练：上面的代码，改为 for 循环。

break 只针对当前所在循环有效。

下面的代码，判断 200 到 300 之间是否有水仙花数，大家可以把 200 和 301 改成 100 和 201 等进一步测试。

```
for n in range(200,301):
    a = n % 10   #个位数
    b = n // 10 % 10   #十位数
    c = n // 100    #百位数
    if a ** 3 + b ** 3 + c ** 3 == n: #判断条件
        print(n)
        break
else:
        print("200 到 300 之间没有水仙花数")
```

6.3.2 continue

(1) **在循环过程中**，如果 **某一个条件满足后**，**不希望执行本次循环代码**，**但是又不希望退出循环**，可以使用 continue

(2) 也就是：在整个循环中，**只有某些条件**，不需要执行循环代码，而其他条件都需要执行

```
i = 0
while i < 10:
    # 当 i == 7 时，不希望执行需要重复执行的代码
    if i == 7:
```

```
        # 在使用 continue 之前，同样应该修改计数器
        # 否则会出现死循环
        i += 1
        continue
    # 重复执行的代码
    print(i)
    i += 1
```

课后演练：上面的代码，改为 for 循环。

- 需要注意：使用 continue 时，**条件处理部分的代码，需要特别注意**，不小心会出现**死循环**

continue 只针对当前所在循环有效。

6.3.3 else

在 Python 中完整的 for 循环的语法如下：

```
for 变量 in 集合:
    循环体代码
else:
    没有通过 break 退出循环，循环结束后，会执行的代码
```

while 循环也有 else 部分。

应用场景

在**迭代遍历**嵌套的数据类型时，例如 **一个列表包含了多个字典**

- 需求：要判断 某一个字典中 是否存在**指定的值**

(1) 如果**存在**，提示并且退出循环

(2) 如果**不存在**，在**循环整体结束**后，希望**得到一个统一的提示**

```
students = [
    {"name": "阿土",
     "age": 20,
     "gender": True,
     "height": 1.7,
     "weight": 75.0},
    {"name": "小美",
     "age": 19,
     "gender": False,
     "height": 1.6,
     "weight": 45.0},
```

```
]
find_name = "阿土"
for stu_dict in students:
    print(stu_dict)
    # 判断当前遍历的字典中姓名是否为 find_name
    if stu_dict["name"] == find_name:
        print("找到了")
        # 如果已经找到，直接退出循环，就不需要再对后续的数据进行比较
        break
else:
    print("没有找到")
print("循环结束")
```

也就是说，for 或者 while 循环中的 else 的功能，是循环正常结束的时候，也就是没有执行 break 语句的时候，执行 else 的代码，否则不执行。再举例一例，定义了一个求素数的函数，判断 n，素数返回真，否则返回假：

```
def prime(n):
    for i in range(2,n-1):
        if n%i==0:
            break
    else:
        return True
    return False
```

这个函数中，for 循环判断 n 是否能被 2 到 n-1 之间的任一个数整除，能整除就 break，这时候 else 代码块就不会被执行，而是执行最后的 return False 语句，即不是素数；否则就执行 else 代码块，即执行 return True，返回真，为素数。函数的具体用法后面会详细介绍，这里只在这个函数中简单介绍。

6.4 循环嵌套

6.4.1 循环嵌套

(1) while 嵌套就是：while 里面还有 while

(2) for 嵌套就是：for 里面还有 for

(3) 也可以混合嵌套

while 条件 1:

条件满足时，做的事情 1

条件满足时，做的事情 2
条件满足时，做的事情 3
...(省略)...

while 条件 2:
条件满足时，做的事情 1
条件满足时，做的事情 2
条件满足时，做的事情 3
...(省略)...
处理条件 2
处理条件 1

6.4.2 循环嵌套演练——九九乘法表

第 1 步：用嵌套打印小星星

需求：

- 在控制台连续输出五行 *，每一行星号的数量依次递增

```
*
**
***
****
*****
```

- 使用字符串 * 打印

```
# 1. 定义一个计数器变量，从数字 1 开始，循环会比较方便
row = 1
while row <= 5:
    print("*" * row)
    row += 1
```

课后演练：上面的代码，改为 for 循环。

第 2 步：使用循环嵌套打印小星星

知识点回顾： print 函数的详细使用方法

(1) 在默认情况下，print 函数输出内容之后，会自动在内容末尾增加换行

(2) 如果不希望末尾增加换行，可以在 print 函数输出内容的后面增加 , end=""

(3) 其中""中间可以指定 print 函数输出内容之后，继续希望显示的内容

- 语法格式如下：

```
# 向控制台输出内容结束之后，不会换行
print("*", end="")
# 单纯的换行
print("")
```

end=""表示向控制台输出内容结束之后，不会换行

假设 Python 没有提供字符串的*操作，拼接字符串

需求：

- 在控制台连续输出五行 *，每一行星号的数量依次递增

```
*
**
***
****
*****
```

开发步骤

(1) 完成 5 行内容的简单输出

(2) 分析每行内部的*应该如何处理？

1) 每行显示的星星和当前所在的行数是一致的

2) 嵌套一个小的循环，专门处理每一行中列的星星显示

```
row = 1
while row <= 5:
    # 假设 python 没有提供字符串 * 操作
    # 在循环内部，再增加一个循环，实现每一行的 星星 打印
    col = 1
```

```
        while col <= row:
            print("*", end="")
            col += 1
        # 每一行星号输出完成后，再增加一个换行
        print("")
        row += 1
```

课后演练：上面的代码，改为 for 循环。

第 3 步： 九九乘法表

需求： 输出九九乘法表，格式如下：

```
1 * 1 = 1
1 * 2 = 2   2 * 2 = 4
1 * 3 = 3   2 * 3 = 6   3 * 3 = 9
1 * 4 = 4   2 * 4 = 8   3 * 4 = 12   4 * 4 = 16
1 * 5 = 5   2 * 5 = 10   3 * 5 = 15   4 * 5 = 20   5 * 5 = 25
1 * 6 = 6   2 * 6 = 12   3 * 6 = 18   4 * 6 = 24   5 * 6 = 30   6 * 6 = 36
1 * 7 = 7   2 * 7 = 14   3 * 7 = 21   4 * 7 = 28   5 * 7 = 35   6 * 7 = 42   7 * 7 = 49
1 * 8 = 8   2 * 8 = 16   3 * 8 = 24   4 * 8 = 32   5 * 8 = 40   6 * 8 = 48   7 * 8 = 56   8 * 8 = 64
1 * 9 = 9   2 * 9 = 18   3 * 9 = 27   4 * 9 = 36   5 * 9 = 45   6 * 9 = 54   7 * 9 = 63   8 * 9 = 72   9 * 9 = 81
```

开发步骤

- 打印 9 行小星星

```
*
**
***
****
*****
******
*******
```


****

****

- 将每一个*替换成对应的行与列相乘

```
# 定义起始行
row = 1
# 最大打印 9 行
while row <= 9:
    # 定义起始列
    col = 1
    # 最大打印 row 列
    while col <= row:
        # end = ""，表示输出结束后，不换行
        # "\t" 可以在控制台输出一个制表符，协助在输出文本时对齐
        print("%d * %d = %d" % (col, row, row * col), end="\t")
        # 列数 + 1
        col += 1
    # 一行打印完成的换行
    print("")
    # 行数 + 1
    row += 1
```

课后演练：上面的代码，改为 for 循环。

注意 **字符串中的转义字符(提前了解)**

(1) \t 在控制台输出一个**制表符**，协助在输出文本时**垂直方向**保持对齐

(2) \n 在控制台输出一个**换行符**

制表符的功能是在不使用表格的情况下在**垂直方向**按列对齐文本

表 6-2 转义字符

转义字符	描述
\\	反斜杠符号
\'	单引号
\"	双引号
\n	换行
\t	横向制表符
\r	回车

习题

一、填空题

1、Python 3.x 语句 for i in range(3):print(i, end=',') 的输出结果为_____。

2、Python 3.x 语句 print(1, 2, 3, sep=',') 的输出结果为_____。

3、对于带有 else 子句的 for 循环和 while 循环，当循环因循环条件不成立而自然结束时_____(会？不会？)执行 else 中的代码。

4、在循环语句中，_____语句的作用是提前结束本层循环。

5、在循环语句中，_____语句的作用是提前进入下一次循环。

二、判断题

1、带有 else 子句的循环如果因为执行了 break 语句而退出的话，则会执行 else 子句中的代码。（ ）

2、对于带有 else 子句的循环语句，如果是因为循环条件表达式不成立而自然结束循环，则执行 else 子句中的代码。（ ）

3、对于生成器对象 x = (3 for i in range(5))，连续两次执行 list(x)的结果是一样的。（ ）

4、在循环中 continue 语句的作用是跳出当前循环。（ ）

5、在编写多层循环时，为了提高运行效率，应尽量减少内循环中不必要的计算。（ ）

三、程序题

1、编写程序，判断一个数字是否为素数，是则返回字符串 YES，否则返回字符串 NO。

2、根据斐波那契数列的定义，$F(0)=0$，$F(1)=1$, $F(n)=F(n–1)+F(n–2)(n≥2)$，输出不大于 100 的序列元素。

3、写出下面代码的运行结果_____。。

```
def Sum(a, b=3, c=5):
    return sum([a, b, c])
print(Sum(a=8, c=2))
print(Sum(8))
print(Sum(8,2))
```

4、写出下面代码的运行结果_____。。

```
def Sum(*p):
    return sum(p)
print(Sum(3, 5, 8))
print(Sum(8))
print(Sum(8, 2, 10))
```

5、下面程序的执行结果是_____。

```
s = 0
for i in range(1,101):
    s += i
else:
    print(1)
```

6、下面程序的执行结果是_____。

```
s = 0
```

```
for i in range(1,101):
    s += i
    if i == 50:
        print(s)
        break
else:
    print(1)
```


第 7 章 程序的异常处理

知识点

(1) 掌握基本的异常处理方法

(2) 理解高级异常处理方法

(3) 理解 raise

7.1 异常处理

Python 程序一般对输入有一定要求，但当实际输入不满足程序要求时，可能会产生程序的运行错误。

```
>>>n = eval(input("请输入一个数字: "))
请输入一个整数: python


Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    n = eval(input("请输入一个数字: "))
  File "<string>", line 1, in <module>
NameError: name 'python' is not defined
```

由于使用了 eval()函数，如果用户输入不是一个数字则可能报错。这类由于输入与预期不匹配造成的错误有很多种可能，不能逐一列出可能性进行判断。为了保证程序运行的稳定性，这类运行错误应该被程序捕获并合理控制。

Python 语言使用保留字 try 和 except 进行异常处理，基本的语法格式如下：。

try:

<语句块 1>

except:

<语句块 2>

- 语句块 1 是正常执行的程序内容，当执行这个语句块发生异常时，则执行 except 保留字后面的语句块 2。例如：

```
try:
    n = eval(input("请输入一个数字: "))
    print("输入数字的 3 次方值为: ", n**3)
except:
     print("输入错误，请输入一个数字!")
```

运行结果：

```
>>>
请输入一个数字: 1010
输入数字的 3 次方值为:  1030301000
```

再次运行：

```
>>>
请输入一个数字: python
输入错误，请输入一个数字!
```

- 除了输入之外，异常处理还可以处理程序执行中的运行异常。

```
>>>for i in range(5):
        print(10/i, end=" ")
Traceback (most recent call last):
  File "<pyshell#12>", line 2, in <module>
    print(10/i, end=" ")
ZeroDivisionError: division by zero
```

改为：

```
try:
```

```
    for i in range(5):
        print(10/i, end=" ")
except:
    print("某种原因，出错了！ ")
```

运行结果：

```
某种原因，出错了！
```

7.2 异常处理的高级用法

7.2.1 try/except/else

在 try 语句后也可以跟一个 else 语句，这样当 try 语句块正常执行，没有发生异常，则将执行 else 语句后的内容：

```
try:
    pass
except Exception as e:
    print ("Exception:", e)
else:
    print( "No exception")
```

7.2.2 try/except/finally

在 try 语句后边跟一个 finally 语句，则不管 try 语句块有没有发生异常，都会在执行 try 之后执行 finally 语句后的内容：

```
try:
   pass
except Exception as e:
   print( "Exception: ",e)
finally:
   print( "try is done")
```

该格式还可以加入 else，构成 try/except/else/finally。

7.2.3 raise 抛出异常

可以使用 raise 主动抛出一个异常：

```
a = 0
if a == 0:
    raise Exception("a must not be zero")
else:
    y=5/a
```

在 Python 中，当不清楚异常需要使用哪个标准异常名称时，可以直接使用 BaseException 异常名称或 Exception 异常名称，BaseException 异常是所有异常的基类，Exception 异常是常规错误的基类。当然，如果没有自己想要的异常名称，可以自定义一个异常。

习 题

一、填空题

1、Python 内建异常类的基类是_____。

2、在异常处理结构中，不论是否发生异常，_____子句中的代码总是会执行的。

二、判断题

1、带有 else 子句的异常处理结构，如果不发生异常则执行 else 子句中的代码。（ ）

2、异常处理结构也不是万能的，处理异常的代码也有引发异常的可能。（ ）

3、在异常处理结构中，不论是否发生异常，finally 子句中的代码总是会执行的。（ ）

三、简答题

1、异常和错误有什么区别？（异常是指因为程序执行过程中出错而在正常控制流以外采取的行为。严格来说，语法错误和逻辑错误不属于异常，但有些语法错误往往会导致异常，例如由于大小写拼写错误而访问不存在的对象，或者试图访问不存在的文件，等等。）

2、阅读下面的代码，并分析假设文件“D:\test.txt”不存在的情况下两段代码可能发生的问题。

代码 1：

```
try:
        fp = open(r'd:\test.txt')
        print('Hello world!', file=fp)
finally:
        fp.close()
```

代码 2：

```
try:
        fp = open(r'd:\test.txt', 'a+')
        print('Hello world!', file=fp)
finally:
        fp.close()
```

答：假设文件“D:\test.txt”不存在，那么第一段代码会抛出异常，提示 fp 没有定义；第二段代码执行正常。原因是第二段代码使用内置函数 open()打开指定文件时如果不存在则会创建该文件，从而不会抛出异常。

3、下面的代码本意是把当前文件夹中所有 html 文件都改为 htm 文件，仔细阅读代码，简要说明可能存在的问题。

```
import os
file_list=os.listdir(".")
for filename in file_list:
        pos = filename.rindex(".")
        if filename[pos+1:] == "html":
                newname = filename[:pos+1]+"htm"
                os.rename(filename,newname)
```

```
                print(filename+"更名为："+newname)
```

答：对于字符串对象，如果要查找的子字符串不存在，则 rindex()方法会抛出异常。所以，如果当前文件夹中有不包含圆点的文件名或者子文件夹名，上面的代码会抛出异常而崩溃。

第 8 章 函数与模块

知识点

(1) 函数的快速体验

(2) 函数的基本使用

(3) 函数的参数

(4) 函数的返回值

(5) 函数的嵌套调用

(6) 使用模块中定义函数

(7) lamda 函数

(8) 变量作用域

8.1 函数的快速体验

1. 快速体验

(1) 所谓**函数**，就是把**具有独立功能的代码块**组织为一个小模块，在需要的时候**调用**

(2) 函数的使用包含两个步骤：

1) 定义函数——**封装**独立的功能

2) 调用函数——享受**封装**的成果

(3) **函数的作用**，在开发程序时，使用函数可以提高编写的效率以及代码的**重用**

2. 演练步骤

(1) 新建 Section10_函数与模块文件夹

(2) 复制之前完成的**乘法表**文件 table.py 到新建的文件夹

(3) 修改文件，增加函数定义 def multiple_table():

```
def multiple_table():
    # 定义起始行
    row = 1
    # 最大打印 9 行
```

```
while row <= 9:
    # 定义起始列
    col = 1
    # 最大打印 row 列
    while col <= row:
        # end = ""，表示输出结束后，不换行
        # "\t" 可以在控制台输出一个制表符，协助在输出文本时对齐
        print("%d * %d = %d" % (col, row, row * col), end="\t")
        # 列数 + 1
        col += 1
    # 一行打印完成的换行
    print("")
    # 行数 + 1
    row += 1
```

(4) 新建另外一个文件，使用 import 导入 table 并且调用 multiple_table()函数并运行

```
import table   #当前文件夹下的 table.py 文件作为模块导入
table. multiple_table()   #调用 table.py 文件中的 multiple_table()函数
```

8.2 函数基本使用

8.2.1 函数的定义

定义函数的格式如下：

def 函数名():

函数封装的代码

……

(1) def 是英文 define 的缩写

(2) **函数名称**应该能够表达**函数封装代码**的功能，方便后续的调用

(3) **函数名称**的命名应该**符合标识符的命名规则**

1) 可以由**字母、下划线**和**数字**组成

2) **不能以数字开头**

3) **不能与关键字重名**

8.2.2 函数调用

调用函数很简单的，通过：函数名() ，即可完成对函数的调用

8.2.3 第一个函数演练

需求：

- 编写一个打招呼 say_hello 的函数，封装三行打招呼的代码，在函数下方调用打招呼的代码

```
name = "小明"
# 解释器知道这里定义了一个函数
def say_hello():
    print("hello 1")
    print("hello 2")
    print("hello 3")
print(name)
# 只有在调用函数时，之前定义的函数才会被执行
# 函数执行完成之后，会重新回到之前的程序中，继续执行后续的代码
say_hello()
print(name)
```

用**单步执行 F8 和 F7** 观察以下代码的执行过程

(1) 定义好函数之后，只表示这个函数封装了一段代码而已

(2) 如果不主动调用函数，函数是不会主动执行的

★思考：能否将函数调用**放在**函数定义**的上方？**

不能！

因为在**使用函数名**调用函数之前，必须要保证 Python 已经知道函数的存在，否则控制台会提示 NameError: name 'say_hello' is not defined (**名称错误：say_hello 这个名字没有被定义**)

8.2.4 PyCharm 的调试工具

(1) **F8 Step Over** 可以单步执行代码，会把函数调用看作是一行代码直接执行

(2) **F7 Step Into** 可以单步执行代码，如果是函数，会进入函数内部

8.2.5 函数的文档注释

(1) 在开发中，如果希望给函数添加注释，应该在**定义函数**的下方，使用**连续的三对引号**，单双引号都可以

(2) 在**连续的三对引号**之间编写对函数的说明文字

(3) 在**函数调用**位置，使用快捷键 CTRL + Q 可以查看函数的说明信息

★注意：

因为**函数体相对比较独立，函数定义的上方**，应该和其他代码（包括注释）保留**两个空行。**

8.3 函数的参数和返回值

演练需求

(1) 开发一个 sum_2_num 的函数

(2) 函数能够实现**两个数字的求和**功能

演练代码如下：

```
def sum_2_num():


    num1 = 10

    num2 = 20

    result = num1 + num2


    print("%d + %d = %d" % (num1, num2, result))
```

```
sum_2_num()
```

思考一下存在什么问题：

函数只能处理**固定数值**的相加

如何解决？

如果能够把需要计算的数字，在调用函数时，传递到函数内部就好了！

8.3.1 函数参数的使用

(1) 在函数名的后面的小括号内部填写**参数**

(2) 多个参数之间使用 ，分隔

```
def sum_2_num(num1, num2):
    result = num1 + num2
    print("%d + %d = %d" % (num1, num2, result))
sum_2_num(50, 20)
```

8.3.2 参数的作用

(1) **函数**，把**具有独立功能的代码块**组织为一个小模块，在需要的时候**调用**

(2) **函数的参数**，增加函数的**通用性**，针对**相同的数据处理逻辑**，能够 **应更多的数据**

1) 在函数**内部**，把参数当做**变量**使用，进行需要的数据处理

2) 函数调用时，按照函数定义的**参数顺序**，把**希望在函数内部处理的数据**，**通过参数**传递

8.3.3 形参和实参

(1) **形参**：**定义**函数时，小括号中的参数，是用来接收参数用的，在函数内部**作为变量使用**

(2) **实参**：**调用**函数时，小括号中的参数，是用来把数据传递到**函数内部**用的

8.3.4 函数的返回值

(1) 在程序开发中，有时候，会希望**一个函数执行结束后，告诉调用者一个结果**，以便调用者针对具体的结果做后续的处理

(2) **返回值**是函数**完成工作**后，**最后**给调用者的**一个结果**

(3) 在函数中使用 return 关键字可以返回结果

(4) 调用函数一方，可以**使用变量**来**接收**函数的返回结果

★注意：return 表示返回，后续的代码都不会被执行

```
def sum_2_num(num1, num2):
    """对两个数字的求和"""
    return num1 + num2
# 调用函数，并使用 result 变量接收计算结果
result = sum_2_num(10, 20)
print("计算结果是 %d" .format( result))
```

8.4 默认参数

Python 支持定义参数个数可变的函数。实现参数个数可变可以采用三种方式：参数默认值、可变参数*args 和字典参数**kargs。后面两种方式在高级数据类型中详细介绍。

8.4.1 参数默认值

Python 一个有用的函数参数形式是给一个或多个**参数指定默认值**，这样创建的函数可以用较少的参数来调用。默认值参数必需从参数列表的最右侧开始，两个有默认值的参数中间，不能存在没有默认值的参数，如 def func(a,b=2,c,d=3)是非法的。

定义一个带参数默认值的函数：

```
def sum4data(a, b=4, c=5,d=6):
    s = a + b + c + d
    print(s)
sum4data (3)
```

数字 3 被赋值给函数中的 a，而 b，c，d 使用默认值赋值。这个函数还可以用这样调用：sum4data (1, 2)，数字 1，2 被依次赋值给 a，b，c=5，d=6。也就说如果参数默认赋值是从右向左进行的。可以看出参数 a 必需给赋值，可以称为**必需参数**。

函数的默认值在函数定义范围内被解析，离开这个范围无效，下面这段代码说明了这一点。

```
i = 5
```

```
def fun ( arg = i):
    print(arg)
i = 6
fun ()
```

以上代码会打印 5。

★重要警告：

默认值只会解析一次。当默认值是一个可变对象，诸如链表、字典或大部分类实例时，会产生一些差异。

8.4.2 关键字参数

对于有默认值参数的函数可以通过**关键字参数**形式来**调用**，形如"***keyword = value***"。例如以下的函数：

关键字参数调用：

```
def func4data(c, x=1, y=2,z=3):
    s = x*x*x + y*y + z + c
    print(s)
```

可以用以下的任一方法调用：

```
func4data (1000)
func4data (1, y = 3)   #1 为必需参数，y=3 为关键字参数
func4data (2, z=10)
func4data (1, 2, 3)
```

不过以下几种调用是无效的：

```
func4data ()                    #没有必需参数
func4data (y=5, 2)              #必需参数必需放在前面，
func4data (11, c=1)             #参数重复
func4data (b=7)                 #未知的关键字参数
```

通常，形式参数没有默认值，那么值就并不固定。实参列表中的每一个关键字都必须来自于形式参数，每个参数都有对应的关键字。不能在同一次调用中对实参同时使用位置和关键字参数。下面的代码演示了在这种约束下所出现的失败情况。

```
def function (a=2):
```

```
    pass
function(0, a=0)
```

运行结果：

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: function() got multiple values for keyword argument 'a'
```

8.5 函数的嵌套调用

(1) 一个函数里面**又调用了另外一个函数**，这就是**函数嵌套调用**

(2) 如果函数 test2 中，调用了另外一个函数 test1

1) 那么执行到调用 test1 函数时，会先把函数 test1 中的任务都执行完

2) 才会回到 test2 中调用函数 test1 的位置，继续执行后续的代码

```
def test1():
    print("*" * 50)
    print("test 1")
    print("*" * 50)
def test2():
    print("-" * 50)
    print("test 2")
    test1()
    print("-" * 50)
test2()
```

函数嵌套的演练——打印分隔线

体会一下工作中**需求是多变**的：

需求 1

- 定义一个 print_line 函数能够打印 * 组成的**一条分隔线**

```
def print_line(char):
```

```
    print("*" * 50)
```

需求 2

- 定义一个函数能够打印**由任意字符组成**的分隔线

```
def print_line(char):
    print(char * 50)
```

需求 3

- 定义一个函数能够打印**任意重复次数**的分隔线

```
def print_line(char, times):
    print(char * times)
```

需求 4

- 定义一个函数能够打印 **5 行**的分隔线，分隔线要求符合**需求 3**

★提示：

工作中针对需求的变化，应该冷静思考，**不要轻易修改之前已经完成的，能够正常执行的函数！**

```
def print_line(char, times):
    print(char * times)
def print_lines(char, times):
    row = 0
    while row < 5:
        print_line(char, times)
        row += 1
```

8.6 使用模块中的函数

模块是 Python 程序架构的一个核心概念

(1) **模块**就好比是**工具包**，要想使用这个工具包中的工具，就需要**导入(import)**这个模块

(2) 每一个以扩展名 py 结尾的 Python 源代码文件都是一个**模块**

(3) 在模块中定义的**全局变量**、**函数**都是模块能够提供给外界直接使用的工具

8.6.1 第一个模块体验

步骤

(1) 新建分隔线模块 hm_10_module01.py

1) 复制 8.5 最后一段程序中的内容

2) 增加一个字符串变量

```
name = "黑马程序员"
```

(2) 新建体验模块 hm_10_ module02.py 文件，并且编写以下代码：

```
import hm_10_ module01


hm_10_ module01.print_line("-", 80)

print(hm_10_ module01.name)
```

体验小结

(1) 可以**在一个 Python 文件**中定义**变量**或者**函数**

(2) 然后在**另外一个文件中**使用，import 导入这个模块

(3) 导入之后，就可以使用**模块名.变量 / 模块名.函数**的方式，使用这个模块中定义的变量或者函数

模块可以让**曾经编写过的代码**方便的被**复用**！

8.6.2 模块名也是一个标识符

(1) 标示符可以由**字母**、**下划线**和**数字**组成

(2) **不能以数字开头**

(3) **不能与关键字重名**

★注意：

如果在给 Python 文件起名时，**以数字开头**是无法在 Python 中导入这个模块的。

8.6.3 模块的分类和组织

1. 模块的分类：

(1) 自定义模块

自定义模块，就是自己编写的.py 文件，即自己编写的 Python 程序文件。

(2) 标准库模块

标准库模块，是 Python 自带的库模块，例如 math 库，random 库，os 库等。

(3) 第三方模块

除了自带的库模块，Python 还有大量的第三方模块（三方库），要使用第三方模块，首先必需安装。

2. 模块的组织

对于大型软件的开发，不可能把所有代码都存放到一个文件中，那样会使得代码很难维护。对于复杂的大型系统，可以使用包（特殊文件夹）来管理多个模块。包是 Python 用来组织命名空间和类的重要方式，可看作是包含大量 Python 程序模块的文件夹。在包的每个文件夹中都必须**包含一个__init__.py 文件**，该文件可以是一个空文件，用于表示当前文件夹是一个包。__init__.py 文件的主要用途是设置__all__变量以及执行初始化包所需的代码，其中__all__变量中定义的对象可以通过使用“from xxx import *”全部被正确导入（xxx 为包名）。

3. 第三方模块的安装

安装第三方模块(三方库)有多种不同的方法和工具，其中采用包管理工具 pip 是最常见的管理方式。使用 pip 必需是联网状态，通过简单的命令就可以实现第三方模块的安装、卸载和更新等。常用的 pip 命令参数如下表（xxx 为第三方库名，如 numpy）。

表 10-1 常用 pip 命令使用方法

命令	命令解释
pip install xxx	安装 xxx 三方库
pip install －upgrade xxx	更新 xxx 三方库
pip list	列出所有安装的三方库
pip uninstall xxx	卸载 xxx 库

4. 模块的引用

引用模块使用 import，有以下引用方式：

(1) import 模块名

(2) import 模块名 as 别名

(3) from 模块名 import 模块内的标识符名 [as 别名]

(4) from math import *#谨慎使用

```
>>>import math
>>>math.sin(0.5)
>>>import random
>>>x=random.random( )
>>>y=random.random( )
>>>n=random.randint(1,100)
>>> from math import sin
>>> sin(3)
0.1411200080598672
>>> from math import sin as f      #别名
>>> f(3)
0.141120008059867
```

模块是 Python 程序架构的一个核心概念。

8.6.4 Pyc 文件（了解）

C 是 compiled **编译过** 的意思

操作步骤

(1) 浏览程序目录会发现一个 __pycache__ 的目录

(2) 目录下会有一个 hm_10_分隔线模块.cpython-35.pyc 文件，cpython-35 表示 Python 解释器的版本

(3) 这个 pyc 文件是由 Python 解释器将**模块的源码**转换为**字节码**

- Python 这样保存**字节码**是作为一种启动**速度的优化**

字节码

(1) Python 在解释源程序时是分成两个步骤的

1) 首先处理源代码，**编译**生成一个二进制**字节码**

2) 再对**字节码**进行处理，才会生成 CPU 能够识别的**机器码**

(2) 有了模块的字节码文件之后，下一次运行程序时，如果在 **上次保存字节码之后** 没有修改过源代码，Python 将会加载 .pyc 文件并跳过编译这个步骤

(3) 当 Python 重编译时，它会自动检查源文件和字节码文件的时间戳

(4) 如果你又修改了源代码，下次程序运行时，字节码将自动重新创建

8.7 lamda 函数

通过 lambda 关键字，可以创建短小的匿名函数。例如一个 lambda 函数："lambda a, b: a+b"，返回两个参数的和。lambda 形式可以用于任何需要的函数对象。出于语法限制，它们只能有一个单独的表达式。语义上讲，它们只是普通函数定义中的一个语法技巧。类似于嵌套函数定义，lambda 函数可以从包含范围内引用变量。例如下面的例子使用了 lamda，lamda 引用了参数变量 n。

```
>>>def make_incrementor (n):
    return lambda x: x + n
>>>f = make_incrementor (42)   #可以看出 f 是函数名
>>>f (0)
42
>>>f (1)
43
```

lambda 用于参数为函数名的函数调用例子：实现加减乘除运算的操作函数。

```
>>>def operate(x,y,callback):
    return callback(x,y)
#加法调用
>>>operate (2,3,lambda x,y:x+y)
```

即：x+y 的结果

lambda x,y:x+y 等价于：

```
def ???(x,y):
    return x+y
```

```
#减法
>>>operate (2,3,lambda x,y:x-y)
#乘法
>>>operate (2,3,lambda x,y:x*y)
#除法
>>>operate (2,3,lambda x,y:x/y)
```

8.8 变量作用域

根据程序中变量所在的位置和作用范围，变量分为**全局变量**、**局部变量**和内嵌函数的**闭包**。

8.8.1 局部变量

(1) 局部变量仅在函数内部，且作用域也在函数内部，全局变量的作用域跨越多个函数。

(2) 局部变量指在函数内部使用的变量，仅在函数内部有效，当函数退出时变量将不再存在。

```
>>>def multiply(x, y = 10):
        z = x*y      # z 是函数内部的局部变量
        return z
>>>s = multiply(99, 2)
>>>print(s)
198
>>>print(z)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    print(z)
```

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

(3) 变量 z 是函数 multiple()内部使用的变量，当函数调用后，变量 z 将不存在。

8.8.2 全局变量

全局变量指在函数之外定义的变量，在程序执行全过程有效。全部变量在函数内部使用时，需要提前使用保留字 global 声明，语法形式如下：

global <全局变量>

```
>>>n = 2       #n 是全局变量
>>>def multiply(x, y = 10):
        global n
        return x*y*n        #  使用全局变量 n
>>>s = multiply(99, 2)
>>>print(s)
396
```

上例中，变量 n 是全局变量，在函数 multiply()中使用时需要在函数内部使用 global 声明，定义后即可使用。

如果未使用保留字 global 声明，即使名称相同，也不是全局变量。

```
>>>n = 2       #n 是全局变量
>>>def multiply(x, y = 10):
        n = x*y
        return n        #  此处的 n 不是全局变量
>>>s = multiply(99, 2)
>>>print(s)
198
>>>print(n)     #不改变外部全局变量的值
2
```

8.8.3 内嵌函数与闭包

在 Python 中，由于一切都是对象，因此允许函数内部创建另一个函数。

1. 内嵌函数

内嵌函数（允许在函数内部创建另一个函数，也叫内部函数）

```
def fun1():
    print('fun1 is calling...')
    def fun2():
        print('fun2 is calling...')
    fun2()
    print('fun1 end...')
fun1()
```

运行结果：

```
fun1 is calling...
fun2 is calling...
fun1 end...
```

★说明：内部函数整个作用域都在外部函数之内，内部函数的定义和调用都在外部函数之内，出了外部函数之外，就没有任何对 fun2 的调用了。

2. 闭包

闭包——函数式编程的重要语法结构。

如果在一个内部函数被外部作用域（但不是全局作用域的变量）进行引用，那么内部函数会被认为是闭包 。

```
def fun_x(x):                #外部作用域的变量 x
    def fun_y(y):            #内部函数（闭包）
        return x * y     #引用了 x 变量
    return funY
i = fun_x(8)
```

```
print(i)    #<function fun_x.<locals>.fun_y at 0x000002889E0E8510>
print(type(i))    #<class 'function'>
print(i(9))    #72
print(fun_x(8)(9))    #72
```

运行结果：

```
fun1 is calling...
fun2 is calling...
fun1 end...
```

8.8.3 nonlocal 的使用

变量使用不当会报错，这时候，一般可以在函数内用 nonlocal 关键字声明 x 不是一个局部变量。

例如下面的程序会报错：

```
def fun1():
    x = 8
    def fun2():
        print(x) #8
        x += x    #
        return x
    return fun2()
print(fun1()) #UnboundLocalError: local variable 'x' referenced before assignment
```

要想处理这个错误，一种方法是把 x 改为列表：

```
def fun1():
    x = [8]
    print(x[0])    #8
    def fun2():
```

```
            print(x[0]) #5
            x[0] += x[0]
            return x[0]
        return fun2()
print(fun1())    #16
```

还有一种方法是在内嵌函数中申明 x 为 nonlocal：

```
def fun1():
    x = 8
    def fun2():
        nonlocal x    #申明 x 不是 fun2 这个函数的局部变量
        x += x
        return x
    return fun2()
print(fun1())    #16
```

8.9 函数名的一些特殊用法

前面，大家已经接触了 Python 中函数名有一些特殊的用法，这里进一步总结和介绍。Python 中函数名，首先是可以作为函数的参数，主要有两种用法，一种是正常的函数名作为参数，另外一种是函数装饰器。其次函数名可以直接赋值给一个变量，那么该变量就可以当函数使用：

```
>>> def add_xy(x,y):
        return x+y
>>> addxy = add_xy
>>> addxy(2,3)
```

此外函数还可以返回函数名，功能类似于函数名赋值给变量。函数名作为参数的两种用法下面详细介绍。

8.9.1 函数名作为函数的参数

```
#定义一个函数名作为参数的函数
def operate(x,y,callback):
    return callback(x,y)
#定义两个普通函数
def add(x,y):
    return x+y
def sub(x,y):
    return x-y
#调用
a=operate(2,3,add)
b=operate(5,3,sub)
print(a,b)
```

运行结果:

```
5 2
```

8.9.2 装饰器

Python 中装饰器的使用很广泛，下面的代码演示了用装饰器，获取一个函数的运行时间。

```
import time
def excut_time(func):
    def wrapper():
        start_time = time.time()
        func()
        end_time = time.time()
        execution_time = (end_time - start_time)*1000
        print("运行了{} ms" .format(execution_time) )
    return wrapper
```

```
@excut_time
def f():
    print("hello")
    time.sleep(1)
    print("world")

f()
```

运行结果:

```
hello
world
运行了  1062 ms
```

@excut_time 加在 f 函数前，相当于调用 f 函数的时候，先用 f 函数名作为参数，调用 excut_time 函数，即类似于函数调用 excut_time(f)。此功能经常用于授权检查、日志记录、Web 开发中的路由地址管理等。

习 题

一、填空题

1、Python 安装第三方库常用的是_____工具。

2、使用 pip 工具升级科学计算扩展库 numpy 的完整命令是_____。

3、使用 pip 工具查看当前已安装的 Python 扩展库的完整命令是_____。

4、已知函数定义 def func(*p):return sum(p)，那么表达式 func(1,2,3) 的值为_____。

5、已知函数定义 def func(*p):return sum(p)，那么表达式 func(1,2,3, 4) 的值为_____。

6、已知 f = lambda x: 5，那么表达式 f(3)的值为_____。

二、判断题

1、尽管可以使用 import 语句一次导入任意多个标准库或扩展库，但是仍建议每次只导入一个标准库或扩展库。（）

2、函数是代码复用的一种方式。（）

3、定义函数时，即使该函数不需要接收任何参数，也必须保留一对空的圆括号来表示这是一个函数。（）

4、编写函数时，一般建议先对参数进行合法性检查，然后再编写正常的功能代码。（）

5、一个函数如果带有默认值参数，那么必须所有参数都设置默认值。（）

6、定义 Python 函数时必须指定函数返回值类型。（）

7、定义 Python 函数时，如果函数中没有 return 语句，则默认返回空值 None。（）

8、如果在函数中有语句 return 3，那么该函数一定会返回整数 3。（）

9、函数中必须包含 return 语句。（）

10、函数中的 return 语句一定能够得到执行。（）

11、不同作用域中的同名变量之间互相不影响，也就是说，在不同的作用域内可以定义同名的变量。（）

12、全局变量会增加不同函数之间的隐式耦合度，从而降低代码可读性，因此应尽量避免过多使用全局变量。（）

13、函数内部定义的局部变量当函数调用结束后被自动删除。（）

14、在函数内部，既可以使用 global 来声明使用外部全局变量，也可以使用 global 直接定义全局变量。（）

15、在函数内部没有办法定义全局变量。（）

16、在函数内部直接修改形参的值并不影响外部实参的值。（）

17、在函数内部没有任何方法可以影响实参的值。（）

18、调用带有默认值参数的函数时，不能为默认值参数传递任何值，必须使用函数定义时设置的默认值。（）

19、在同一个作用域内，局部变量会隐藏同名的全局变量。（）

20、形参可以看做是函数内部的局部变量，函数运行结束之后形参就不可访问了。（）

21、在定义函数时，某个参数名字前面带有一个*符号表示可变长度参数，可以接收任意多个普通实参并存放于一个元组之中。（）

22、在定义函数时，某个参数名字前面带有两个*符号表示可变长度参数，可以接收任意多个关键参数并将其存放于一个字典之中。（）

23、定义函数时，带有默认值的参数必须出现在参数列表的最右端，任何一个带有默认值的参数右边不允许出现没有默认值的参数。（）

24、在调用函数时，可以通过关键参数的形式进行传值，从而避免必须记住函数形参顺序的麻烦。（）

25、在调用函数时，必须牢记函数形参顺序才能正确传值。（）

26、调用函数时传递的实参个数必须与函数形参个数相等才行。（）

27、在编写函数时，建议首先对形参进行类型检查和数值范围检查之后再编写功能代码，或者使用异常处理结构，尽量避免代码抛出异常而导致程序崩溃。（）

三、程序题

1、写出下面代码的运行结果。

```
def Sum(a, b=3, c=5):
    return sum([a, b, c])
print(Sum(a=8, c=2))
print(Sum(8))
print(Sum(8,2))
```

2、写出下面代码的运行结果。

```
def Sum(*p):
    return sum(p)
print(Sum(3, 5, 8))
print(Sum(8))
print(Sum(8, 2, 10))
```

3、编写函数，参数为年份，判断参数是否为闰年。如果年份能被 400 整除，则为闰年，返回 True；如果年份能被 4 整除但不能被 100 整除也为闰年，返回 True，否则返回 False。

第 9 章 字符串类型

知识点

(1) 掌握字符串定义

(2) 了解字符串所有方法，掌握字符串常用方法

(3) 掌握字符串索引与切片

(4) 理解字符串常用内置函数和运算

(5) 掌握转义字符和原始字符串

9.1 字符串的定义

字符串就是**一串字符**，是编程语言中表示文字的数据类型。在 Python 中可以使用**一对单引号**' '、**一对双引号**" "、**一对三单引号**''' '''或者**一对三双引号**""" """、定义一个字符串。不同的定界符之间可以相互嵌套，使用要点如下：

(1) 字符串内部出现双引号（"）或者单引号（'）时，可以使用\"或\'做引号字符的转义，但实际开发中字符串内出现双引号（"），一般使用两个单引号包括整个字符串，如 'hello,"it" is a good boy'；字符串内出现双引号（'），可以使用两个双引号包括整个字符串，如"hello,'it' is a good boy"。

(2) 字符串属于不可变序列，不能修改。

(3) 空字符串表示为''或""。

(4) 三引号'''或"""表示的字符串**可以换行**，支持排版较为复杂的字符串；三引号还可以在程序中表示较长的注释。

(5) 使用字符串对象提供的 replace()、translate()等方法后，返回一个修改替换后的新字符串作为结果。

(6) 可以使用**索引**获取一个字符串中**指定位置的字符**，索引计数从 **0** 开始；也可以使用 for **循环遍历**字符串中每一个字符。见图 8-1。

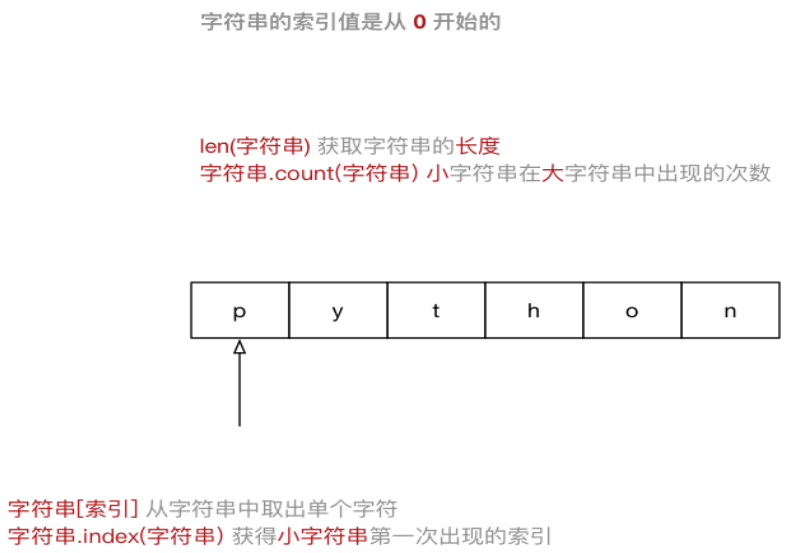


图 8-1 字符串

下面的代码使用双引号来定义字符串，并遍历输出：

```
string = "Hello Python"
for c in string:
    print(c,end=',')
```

运行结果

```
H,e,l,l,o, ,P,y,t,h,o,n,
```

9.2 字符串的常用方法

我们在 ipython3 中定义一个**字符串**，例如：hello_str = ""，然后输入 hello_str，按下 TAB 键，ipython 会提示**字符串**能够使用的**方法**如下：

```
In [1]: hello_str.

hello_str.capitalizehello_str.isidentifierhello_str.rindex

hello_str.casefoldhello_str.islowerhello_str.rjust

hello_str.centerhello_str.isnumerichello_str.rpartition

hello_str.counthello_str.isprintablehello_str.rsplit

hello_str.encodehello_str.isspacehello_str.rstrip
```

```
hello_str.endswithhello_str.istitlehello_str.split
hello_str.expandtabshello_str.isupperhello_str.splitlines
hello_str.findhello_str.joinhello_str.startswith
hello_str.formathello_str.ljusthello_str.strip
hello_str.format_maphello_str.lowerhello_str.swapcase
hello_str.indexhello_str.lstriphello_str.title
hello_str.isalnumhello_str.maketranshello_str.translate
hello_str.isalphahello_str.partitionhello_str.upper
hello_str.isdecimalhello_str.replacehello_str.zfill
hello_str.isdigithello_str.rfind
```

如上所示， Python 内置提供了足够多的字符串操作方法，开发时，能够针对字符串进行更加灵活的操作，应对更多的开发需求。

1. 判断类型的 9 个方法

表 8-1 判断类型

方法	说明
string.isspace()	如果 string 中只包含空格，则返回 True
string.isalnum()	如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True
string.isalpha()	如果 string 至少有一个字符并且所有字符都是字母则返回 True
string.isdecimal()	如果 string 只包含数字则返回 True，全角数字
string.isdigit()	如果 string 只包含数字则返回 True，全角数字、⑴、\u00b2
string.isnumeric()	如果 string 只包含数字则返回 True，全角数字，汉字数字
string.istitle()	如果 string 是标题化的(每个单词的首字母大写)则返回 True
string.islower()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True
string.isupper()	如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True

代码举例如下：

```
>>> "123 一二三".isnumeric()
True
>>> "123 一二三".isdecimal()
False
>>> "123 1  2  3 ".isdecimal()
True
>>> 'abc10'.isalnum()
True
>>> 'abc10'.isalpha()
False
>>> 'abc10'.isdigit()
False
>>> 'abc'.islower()
True
```

2. 查找和替换的 7 个方法如下表：

表 8-2 查找和替换

方法	说明
string.startswith(str)	检查字符串是否是以 str 开头，是则返回 True
string.endswith(str)	检查字符串是否是以 str 结束，是则返回 True
string.find(str, start=0, end=len(string))	检测 str 是否包含在 string 中，如果 start 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1
string.rfind(str, start=0, end=len(string))	类似于 find()，不过是从右边开始查找
string.index(str, start=0, end=len(string))	与 find() 方法类似，不过如果 str 不在 string 会报错
string.rindex(str, start=0, end=len(string))	类似于 index()，不过是从右边开始
string.replace(old_str, new_str, num=string.count(old))	把 string 中的 old_str 替换成 new_str，如果 num 指定，则替换不超过 num 次

代码举例如下：

```
>>> 'abcabcabc'.count('abc')
3
>>> 'Hello world!'.count('l')
3
>>> 'C:\\Windows\\notepad.exe'.startswith('C:')
True
>>>r'c:\windows\notepad.exe'.endswith('.exe')
True
>>>r'c:\windows\notepad.exe'.endswith(('.jpg', '.exe'))
True
>>> 'Beautiful is better than ugly.'.startswith('Be', 5)
False
>>> 'abcab'.replace('a','yy')
'yybcyyb'
>>> 'hello world, hellow every one'.replace('hello', 'hi')
'hi world, hiw every one'
>>> 'apple.peach,banana,pear'.find('p')
1
>>> 'apple.peach,banana,pear'.find('ppp')
-1
>>> 'Hello world. I like Python.'.rfind('Python')
-1
>>> 'abcabcabc'.rindex('abc')
6
```

3. 大小写转换的 5 个方法如下表：

表 8-3 大小写转换

方法	说明
string.capitalize()	把字符串的第一个字符大写
string.title()	把字符串的每个单词首字母大写

方法	说明
string.lower()	转换 string 中所有大写字符为小写
string.upper()	转换 string 中的小写字母为大写
string.swapcase()	翻转 string 中的大小写

```
>>> 'abcabcabc'.capitalize()
‘Abcabcabc’
>>> 'I like programing'.title()
‘I Like Programing’
>>> 'AbcAbcAbc'.lower()
‘abcabcabc’
>>> 'AbcAbcAbc'.swapcase()
‘aBCaBCaBC’
```

4. 文本对齐的 3 个方法如下表：

表 8-4 文本对齐

方法	说明
string.ljust(width)	返回一个原字符串左对齐，并使用空格填充至长度 width 的新字符串
string.rjust(width)	返回一个原字符串右对齐，并使用空格填充至长度 width 的新字符串
string.center(width)	返回一个原字符串居中，并使用空格填充至长度 width 的新字符串

代码举例如下：

```
>>>len('Hello world!'.ljust(20))
20
>>>len('abcdefg'.ljust(3))
7
```

5. 去除（空白）字符 3 个方法如下表：

表 8-5 去除（空白）字符

方法	说明
string.lstrip()	截掉 string 左边（开始）的空白字符
string.rstrip()	截掉 string 右边（末尾）的空白字符
string.strip()	截掉 string 左右两边的空白字符

代码举例如下：

```
>>> 'abcab'.strip('ab')
'c'
>>> 'aaasdf'.lstrip('as')
'df'
>>> 'aaasdf'.lstrip('af')
'sdf'
>>> 'aaasdf'.strip('af')
'sd'
>>> 'aaasdf'.rstrip('af')
'aaasd'
```

6. 拆分和连接的 5 个方法如下表：

表 8-6 拆分和连接

方法	说明
string.partition(str)	把字符串 string 分成一个 3 元素的元组 (str 前面, str, str 后面)
string.rpartition(str)	类似于 partition() 方法，不过是从右边开始查找
string.split(str="", num)	以 str 为分隔符拆分 string，如果 num 有指定值，则仅分隔 num + 1 个子字符串，str 默认包含 '\r', '\t', '\n' 和空格
string.splitlines()	按照行('\r', '\n', '\r\n')分隔，返回一个包含各行作为元素的列表
string.join(seq)	以 string 作为分隔符，将 seq 中所有的元素（的字符串表示）合并为一个新的字符串

代码举例如下：

```
>>> ''.join('asdssfff'.split('sd'))
'assfff'
>>> 'abcdefg'.split('d')
['abc', 'efg']
>>> ','.join('a        b    ccc\n\n\nddd        '.split())
'a,b,ccc,ddd'
>>> 'a'.join('abc'.partition('a'))
'aaabc'
>>> x = 'a     b c        d'
>>> ','.join(x.split())
'a,b,c,d'
```

9.3 字符串的切片

切片方法适用于**字符串**、**列表**、**元组**。**切片**使用**索引值**来限定范围，从一个大的**字符串**中**切出**小的**字符串**；**列表**和**元组**都是**有序**的集合，都能够**通过索引值**获取到对应的数据；**字典**是一个**无序**的集合，是使用**键值对**保存数据，不适用于切片操作。

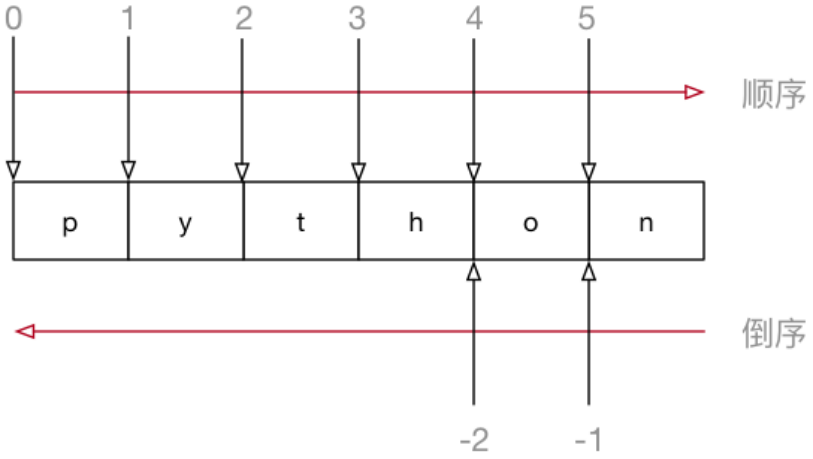


图 8-2 字符串切片

如图 8-2 所示，字符串切片操作指定的区间属于**左闭右开**型[开始索引,结束索引)， 即从**起始位**开始，到**结束位的前一位**结束（**不包含结束位本身**)。总体形如：

```
string[start: end: step]
```

解释如下：

(1) 从头开始，**开始索引数字可以省略，冒号不能省略。**

(2) 到末尾结束，**结束索引数字可以省略，冒号不能省略。**

(3) 步长默认为 1，如果连续切片，**数字和冒号都可以省略**

索引的顺序和倒序：

在 Python 中不仅支持**顺序索引**，同时还支持**倒序索引。**所谓倒序索引就是**从右向左**计算索引：此时最右边的索引值是**-1**，向左依次递减。

下面是字符串索引举的几个例子：

(1) 截取 2~5 位置的字符串

```
num_str = "0123456789"
print(num_str[2:6])
```

运行结果：

```
2345
```

(2) 截取 2~末尾的字符串

```
print(num_str[2:])
```

运行结果：

```
23456789
```

(3) 截取从开始~数字 5 位置的字符串

```
print(num_str[:6])
```

运行结果：

```
012345
```

(4) 截取完整的字符串

```
print(num_str[:])
```

运行结果：

```
0123456789
```

(5) 从开始（位置 0）开始，每隔一个字符截取字符串

```
print(num_str[::2])
```

运行结果：

```
02468
```

(6) 从位置 1 开始，每隔 1 个字符截取字符串

```
print(num_str[1::2])
```

运行结果：

```
13579
```

(7) 倒序切片，-1 表示最后一个字符

```
print(num_str[::-1])
```

运行结果：

```
9876543210
```

(8) 截取位置 8~末尾（-1）的字符串

```
print(num_str[8:-1])
```

运行结果：

```
8
```

(9) 截取字符串末尾两个字符

```
print(num_str[-2:])
```

运行结果：

```
89
```

(10) 字符串的逆序（Python 常考的一个面试题）

```
print(num_str[::-1])
```

运行结果：

```
9876543210
```

9.4 字符串的其他用法

9.4.1 字符串运算

1. 字符串可以进行加（+）、乘（*）数学运算符与比较运算符进行运算。

来看下面的例子。

(1) 字符串的加法运算

```
string1 = 'abcd'
string2 = '1234"
x = string1+string2
print (x)
x = x'efg'                    #不允许这样连接字符串
print (x)
```

运行结果：

```
abcd1234
SyntaxError: invalid syntax
```

以上可见可以通过+进行字符串的连接。

(2) 字符串的乘法运算

```
x="abc"
x=x*5
print(x)
```

运行结果：

```
'abcabcabcabcabc'
```

(3) 字符串的比较运算

```
x="abc"
y="acd"
print (x >= y)
```

```
print (x < y)
```

运行结果：

```
False
True
```

两个字符串按照字符出现顺序逐个字符比较。如果两个字符串某一相同位置的两个字符比较后能够得出布尔结果，返回布尔值，否则继续循环，直到得出结果。

2．字符串进行 in，not in 运算

例子如下：

(1) **字符串的 in 操作**

```
py = "人生苦短，我爱 Python"
print ("人生" in py)
print ("Python" in py)
print ("陈福明" in py)
```

运行结果：

```
True
True
False
```

如上，我们可以通过 in 关键字判断一个字符串是否在另外一个字符串里，如果在，返回 True，否则返回 False。

(2) **字符串的 not in 操作**

```
py = "人生苦短，我爱 Python"
print (陈福明" not in py)
print ("人生苦短" not in py)
```

运行结果：

```
True
False
```

如上，我们可以通过 not in 关键字判断一个字符串是否不在另外一个字符串里，如果不在，返回 True，在返回 False。

9.4.2 字符串内置函数

Python 针对字符串操作，设置了很多内置函数，包括 len()、str()等。

(1) len 函数用来测试字符串的长度，具体使用见下面的例子。

```
>>>len ("hello world")
11
```

以上通过 len 函数计算出字符串的长度为 11。

(2) str 函数将返回一个对象的 string 格式，从而将对象转化为适于人或机器阅读的形式。

```
>>>str1=str(1024)
>>>str1
'1024'
>>>dict = {'baidu': 'baidu.com', 'google': 'google.com', 'pythonlearning': ' pythonlearning.com'}
>>>dict
"{'baidu': 'baidu.com', 'google': 'google.com', 'pythonlearning': ' pythonlearning.com'}"
```

以上的例子通过 str 函数将一个数字和一个字典数据转化为各自对应的字符串类型。

(3) bin 返回一个整数 int 或者长整数 long int 的二进制表示。

```
>>>b1=bin(1024)
>>>b1
'0b10000000000'
```

(4) oct 函数将一个整数转换成 8 进制字符串。

```
>>>o1=oct(1024)
>>>o1
'0o2000'
```

(5) hex() 函数用于将 10 进制整数转换成 16 进制，以字符串形式表示。

```
>>>h1=hex(1024)
>>>h1
```

```
'0x400'
```

(6) chr 函数用一个范围在 range（如：256）内的（就是 0～255）整数作参数，返回一个对应的字符。

```
>>>ch1=chr(65)
>>>ch1
'A'
```

(7) ord 函数是 chr 函数的配对函数，它以一个字符（长度为 1 的字符串）作为参数，返回对应的 ASCII 数值，或者 Unicode 数值，如果所给的 Unicode 字符超出了你的 Python 定义范围，则会引发一个 TypeError 的异常。

```
>>>o1=ord('A')
>>>o1
65
```

9.4.3 常用转义字符

转义字符是指，在字符串中某些特定的符号前加一个斜线，该字符被解释成另外一种含义，即有些字符的表示，需要通过“\”进行专业表达，常用的转移字符及其含义如表 8-7 所示。

表 8-7 转义字符

转义字符	含义	转义字符	含义
\b	退格，把光标移动到前一列位置	\\	一个斜线\
\f	换页符	\’	单引号’
\n	换行符	\”	双引号”
\r	回车	\ooo	3 位八进制数对应的字符
\t	水平制表符	\xhh	2 位十六进制数对应的字符
\v	垂直制表符	\uhhhh	4 位十六进制数表示的 Unicode 字符

转义字符用法举例如下：

(1) 换行符

```
print ('Hello\nWorld')
```

运行结果：

```
Hello
World
```

(2) 八进制转义字符

```
>>>print('\101')
A
```

(3) 其他数字转义字符

```
>>> print('\x41')
A
>>> print("我是\u9648\u798f\u660e")  #四位十六进制数表示 Unicode 字符
我是陈福明
```

9.4.4 原始字符串表达

字符串前面加字母 r 或 R 表示**原始字符串**，其中的特殊字符不进行转义，但字符串的**最后一个字符不能是“\”**。原始字符串主要用于正则表达式、文件路径或者 URL 的场合。

(1) 路径中含转移字符

```
path = 'C:\Windows\notepad.exe'
print(path)
```

运行结果：

```
C:\Windows
```

```
otepad.exe
```

(2) 通过 r 或者 R 去掉路径中转移字符

```
path = r'C:\Windows\notepad.exe' #原始字符串，任何字符都不转义
print (path)
```

运行结果：

```
C:\Windows\notepad.exe
```

9.4.5 Python3 字符编码

(1) Python3 字符编码

Python3 的解释器以“UTF-8”作为默认字符串编码，**其他编码一律归为字节串**。Unicode 为字符集，UTF-8 为 Unicode 字符集的编码规则。所以 **Python3 *字符串*只有 *一种编码*就是 UTF-8**。

(2) 字符串与字节串的互转

字节串—>decode('原来的字符编码')—>Unicode 字符串—>encode('新的字符编码')—>字节串。

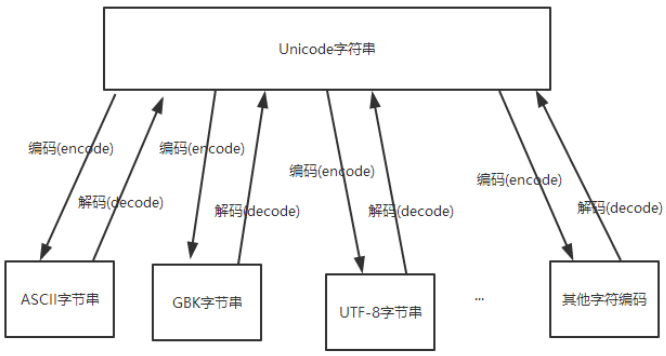


图 8-3 字符串与字节串的互转

(3) chr 和 ord 函数用于在单字符和 Unicode 字符编码值之间进行转换。

```
len('中国'.encode('utf-8'))
len('中国'.encode('gbk'))
```

```
chr(ord('A')+2)
ord('A')
65
```

运行结果：

```
6
4
'C'
65
```

习 题

一、填空题

1、任意长度的 Python 列表、元组和字符串中最后一个元素的下标为_____。

2、Python 语句".join(list('hello world!'))执行的结果是_____。

3、转义字符'\n'的含义是_____。

4、表达式 'ab' in 'acbed' 的值为_____。

5、表达式 str([1, 2, 3]) 的值为_____。

6、表达式 str((1, 2, 3)) 的值为_____。

7、表达式 '{0:#d},{0:#x},{0:#o}'.format(65) 的值为_____。

8、表达式 isinstance('abcdefg', str) 的值为_____。

9、表达式 isinstance('abcdefg', object) 的值为_____。

10、表达式 isinstance(3, object) 的值为_____。

11、表达式 'abcabcabc'.rindex('abc') 的值为_____。

12、表达式 ':'.join('abcdefg'.split('cd')) 的值为_____。

13、表达式 'Hello world. I like Python.'.rfind('Python') 的值为_____。

14、表达式 'abcabcabc'.count('abc') 的值为_____。

15、表达式 'apple.peach,banana,pear'.find('p') 的值为_____。

16、表达式 'apple.peach,banana,pear'.find('ppp') 的值为_____。

17、表达式 'abcdefg'.split('d') 的值为_____。

18、表达式 ':'.join('1,2,3,4,5'.split(',')) 的值为_____。

19、表达式 ','.join('a　　b　ccc\n\n\nddd　　'.split()) 的值为_____。

20、表达式 'Hello world'.upper() 的值为_____。

21、表达式 'Hello world'.lower() 的值为_____。

22、表达式 'Hello world'.lower().upper() 的值为_____。

23、表达式 'Hello world'.swapcase().swapcase() 的值为_____。

24、表达式 r'c:\windows\notepad.exe'.endswith('.exe') 的值为_____。

25、表达式 r'c:\windows\notepad.exe'.endswith(('.jpg', '.exe')) 的值为_____。

26、表达式 'C:\\Windows\\notepad.exe'.startswith('C:') 的值为_____。

27、表达式 len('Hello world!'.ljust(20)) 的值为_____。

28、表达式 len('abcdefg'.ljust(3)) 的值为_____。

29、表达式 'a' + 'b' 的值为_____。

30、已知 x = '123' 和 y = '456'，那么表达式 x + y 的值为_____。

31、表达式 'a'.join('abc'.partition('a')) 的值为_____。

32、表达式 ''.join('asdssfff'.split('sd')) 的值为_____。

33、表达式 ''.join(re.split('[sd]','asdssfff')) 的值为_____。

34、表达式 'Hello world!'[-4] 的值为_____。

35、表达式 'Hello world!'[-4:] 的值为_____。

36、表达式 'test.py'.endswith(('.py', '.pyw')) 的值为_____。

37、当在字符串前加上小写字母_____或大写字母_____表示原始字符串，不对其中的任何字符进行转义。

38、表达式 len('中国'.encode('utf-8')) 的值为_____。

39、表达式 len('中国'.encode('gbk')) 的值为_____。

40、表达式 chr(ord('A')+2) 的值为_____。

41、表达式 'abcab'.replace('a','yy') 的值为_____。

42、已知 table = ''.maketrans('abcw', 'xyzc')，那么表达式 'Hellowworld'.translate(table) 的值为_____。

43、表达式 'hello world, hellow every one'.replace('hello', 'hi') 的值为

_____。

44、已知字符串 x = 'hello world'，那么执行语句 x.replace('hello', 'hi') 之后，x 的值为_____。

45、已知 x = 'a b c d'，那么表达式 ','.join(x.split()) 的值为_____。

46、表达式 'abcab'.strip('ab') 的值为_____。

47、表达式 'abc.txt'.endswith(('.txt', '.doc', '.jpg')) 的值为_____。

48、表达式 isinstance('abc', str) 的值为_____。

49、表达式 isinstance('abc', int) 的值为_____。

50、表达式 'abc10'.isalnum() 的值为_____。

51、表达式 'abc10'.isalpha() 的值为_____。

52、表达式 'abc10'.isdigit() 的值为_____。

53、表达式 'C:\\windows\\notepad.exe'.endswith('.exe') 的值为_____。

54、表达式 len('SDIBT') 的值为_____。

55、表达式 'Hello world!'.count('l') 的值为_____。

56、已知 x = 'abcdefg'，则表达式 x[3:] + x[:3] 的值为_____。

57、表达式 'aaasdf'.lstrip('as') 的值为_____。

58、表达式 'aaasdf'.lstrip('af') 的值为_____。

59、表达式 'aaasdf'.strip('af') 的值为_____。

60、表达式 'aaasdf'.rstrip('af') 的值为_____。

61、表达式 'ac' in 'abce' 的值为_____。

62、表达式 'Beautiful is better than ugly.'.startswith('Be', 5) 的值为_____。

63、表达式 'abc' in 'abdcefg' 的值为_____。

二、判断题

1、在 UTF-8 编码中一个汉字需要占用 3 个字节。

2、在 GBK 和 CP936 编码中一个汉字需要 2 个字节。

3、在 Python 中，任意长的字符串都遵守驻留机制。

4、Python 运算符%不仅可以用来求余数，还可以用来格式化字符串。

5、Python 字符串方法 replace()对字符串进行原地修改。

6、如果需要连接大量字符串成为一个字符串，那么使用字符串对象的 join()方法比运算符+具有更高的效率。

7、表达式 ‘a’+1 的值为’b’。

8、已知 x 为非空字符串，那么表达式 ''.join(x.split()) == x 的值一定为 True。

9、已知 x 为非空字符串，那么表达式 ','.join(x.split(',')) == x 的值一定为 True。

10、相同内容的字符串使用不同的编码格式进行编码得到的结果并不完全相同。

三、程序题

1、以下面中一句话作为字符串变量 s，补充程序，分别输出字符串 s 中汉字和标点符号的个数。提示：请区分票点符合是全角还是半角，本题中的标点符号是全角的。

```
s ="明月几时有？把酒问青天。不知天上宫阙，今夕是何年。我欲乘风归去，又恐琼楼玉宇，高处不胜寒。起舞弄清影，何似在人间？"
n =0# 汉字个数
m =0# 标点符号个数
____①____ # 在这里补充代码，可以多行
print("字符数为{}，标点符号数为{}。".format(n, m))
```

2、一个班选举班长，选举票放在一个字符串中，如：votes="张力,王之夏,jams,韩梅梅,john,韩梅梅,john,韩梅梅"，请编程统计每个人的得票数。假定只有张力,王之夏,jams,韩梅梅,john 这五个候选人。

第 10 章 高级数据类型

知识点

(1) 列表

(2) 元组

(3) 字典

(4) 集合

(5) 高级数据其他用法

10.1 高级数据简介

10.1.1 知识点回顾

(1) Python 中数据类型根据是否为数字分为**数字型**和**非数字型**

(2) Python 中数据类型根据数据的复杂性，分为**简单类型**和**高级类型**，简单类型包括**数字型**和**字符串**

(3) 数字型

1) 整型 (int)

2) 浮点型（float）

3) 布尔型（bool）

a) 真 True 非 0 数 —— **非零即真**

b) 假 False 0

4) 复数型 (complex)

a) 主要用于科学计算，例如：平面场问题、波动问题、电感电容等问题

(4) 非数字型

1) 字符串

2) 列表

3) 元组

4) 集合

5) 字典

(5) 非数字型中，除了字符串，其他都是高级类型

(6) 在 Python 中，所有**非数字型中的序列变量**，即字符串、列表、元组都支持以下特点：

1) 都是一个**序列** sequence，也可以理解为**容器**

2) **取值** []

3) **遍历** for in

4) **计算长度、最大/最小值、比较、删除**

5) **链接**+和**重复***

6) **切片**

10.2.1 高级数据及其分类

高级数据类型往往能够表示多个简单数据。例如：

(1) 给定一个班级的成绩，分别统计各门课程的及格率

(2) 学院召开会议，统计没有参会的各部门人数

(3) 给定一段文字，统计空格、出现最多的字等

python 中常用的三种高级数据类型，分别是：序列类型、集合类型、映射类型。

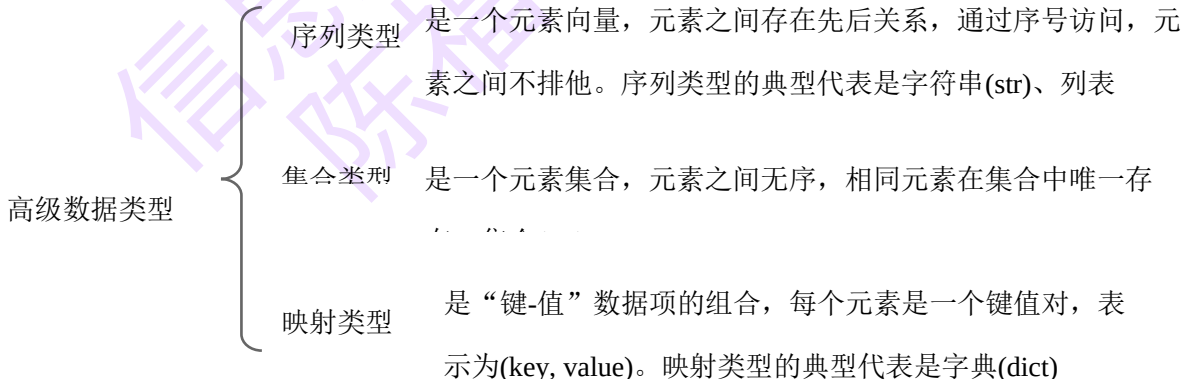


图 9-1 高级数据类型的分类

在有些书中，也把高级数据类型称为组合数据类型或者容器数据类型。而且对于字符串到底属于那种分类，也比较混乱。在本书中，把字符串归类为简单数据类型，但是字符串也属于序列类型。

10.2 列表

10.2.1 列表的定义

(1) List（列表）是 Python 中使用**最频繁**的数据类型，在其他语言中通常叫做**数组**

(2) 专门用于存储**一串信息**

(3) 列表用 [] 定义，**数据**之间使用，分隔

(4) 列表的索引**从 0 开始**

- **索引**就是数据在**列表**中的**位置编号**，**索引**又可以被称为**下标**

★注意：从列表中取值时，如果**超出索引范围**，程序会报错

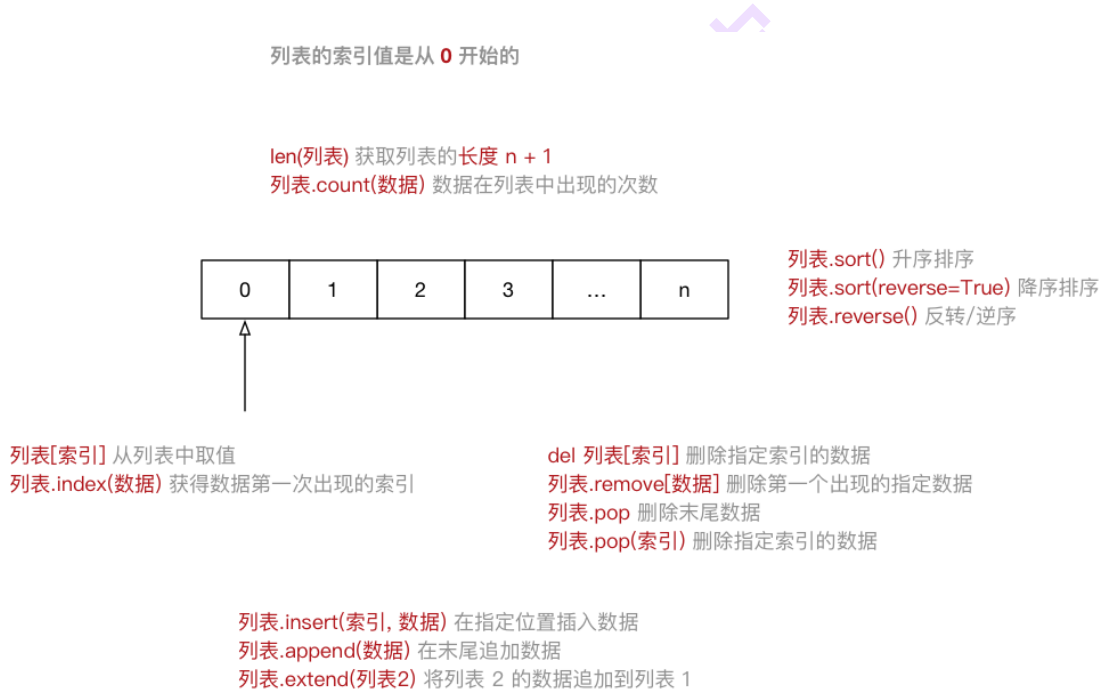


图 9-2 列表

★小提示：

Python 采用的是基于值的内存管理模式，Python 变量不直接存储值，而是存储值的引用，Python 列表中元素也是存储值的引用，所以列表中各元素可以是不同类型的数据。

10.2.2 列表常用操作

(1) 在 ipython3 中定义一个**列表**，例如：name_list = []，定义了一个空列表

(2) 输入 name_list. 按下 TAB 键，ipython 会提示**列表**能够使用的**方法**如下：

```
In [1]: name_list = ["zhangsan", "lisi", "wangwu"]
In [2]: name_list.
name_list.append     name_list.count      name_list.insert     name_list.reverse
name_list.clear      name_list.extend     name_list.pop        name_list.sort
name_list.copy       name_list.index      name_list.remove
```

表 9-1 列表常用操作

序号	分类	关键字 / 函数 / 方法	说明
1	增加	列表.insert(索引，数据)	在指定位置插入数据
|	列表.append(数据)	在末尾追加数据	
|	列表.extend(列表 2)	将列表 2 的数据追加到列表	
2	修改	列表[索引] = 数据	修改指定索引的数据
3	删除	del 列表[索引]	删除指定索引的数据
|	列表.remove(数据)	删除第一个出现的指定数据	
|	列表.pop	删除末尾数据	pop 返回值为被删元素
|	列表.pop(索引)	删除指定索引数据	pop 返回值为被删元素
|	列表.clear	清空列表	
4	统计	len(列表)	列表长度
|	列表.count(数据)	数据在列表中出现的次数	
5	排序	列表.sort()	升序排序
|	列表.sort(reverse=True)	降序排序	
|	列表.reverse()	逆序、反转	

★注意：

列表常用操作中的返回值比较特殊，除了 pop 方法返回值为被删除的元素外，其他方法返回值都是 **None**，但是列表的内容会出现相应的改变，例如 m=list.sort()，那么 m 的值为 None，但是列表 list 的内容以及排序好了。

列表常用操作的代码举例如下：

(1) 创建列表：把逗号分隔的不同的数据项使用方括号括起来

```
list = [1,2,3,'James','张三']
list = [i**2 for i in range(10)]
```

(2) append()尾部新增元素

```
>>> list = [1,2,3]
>>> list.append(5)
>>> list
[1, 2, 3, 5]
```

(3) insert()插入元素：list.insert(index, object)，index 为位置，object 为要插入的对象

```
>>> list = [1,2,3,5]
>>> list.insert(3,4)
>>> list
[1, 2, 3, 4, 5]
```

(4) extend()扩展列表

```
>>> list1 = [1,2,3,4]
>>> list2 = ['a','b']
>>> list1.extend(list2)
>>> list1
[1, 2, 3, 4, 'a', 'b']
```

(5) +号用于组合列表(与 extend()类似)

```
>>> list1 = [1,2,3,4]
>>> list2 = ['a','b']
>>> list1+ list2
[1, 2, 3, 4, 'a', 'b']
```

★注意：

严格意义上来讲，使用加号不是真的为列表添加元素，而是创建一个新列表， 并将原列表中的元素和新元素依次复制到新列表的内存空间。由于涉及大量元素的复制，该操作速度较慢，在涉及大量元素添加时不建议使用该方法。

(6) * 号用于重复列表

```
>>> L1 = [1,2,3]
>>> L1*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

(7) remove()删除元素(参数如有重复元素，只会删除最靠前的)

```
>>> list = [1,3,'a','b','a']
>>> list.remove('a')
>>> list
[1, 3, 'b', 'a']              # 第一个‘a’被删除，后面的没有被删除
```

(8) pop()默认删除最后一个元素，如果有参数，参数为要删除列表元素的对应索引值

```
>>> list = [1,2,3,4,5]
>>> list.pop()  # 默认删除最后一个元素
5
>>> list
[1, 2, 3, 4]
>>> list.pop(2)  # 删除指定索引(index=2)的元素
3
>>> list
[1, 2, 4]
```

(9) reverse()列表元素反转

```
>>> list = [1,2,3,4,5]
>>> list.reverse()
>>> list
[5, 4, 3, 2, 1]
```

(10) sort()排序(sort 有三个默认参数 cmp=None,key=None,reverse=False 因此可以制定排序参数)

```
>>> a = [1,2,5,6,4]
>>> a.sort()
>>> a
```

```
[1, 2, 4, 5, 6]
#Python3.x 中，不能将数字和字符一起排序，会出现此报错
>>> a = [1,2,'c','h']
>>> a.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
#参数 reverse=False,升序排序（默认）；
#参数 reverse=True，降序排序
>>> a = [1,5,9,10,3]
>>> a.sort()
>>> a
[1, 3, 5, 9, 10]
>>> a.sort(reverse=True)
>>> a
[10, 9, 5, 3, 1]
#参数 cmp 可以指定比较的函数，后面进一步介绍
```

★注意：

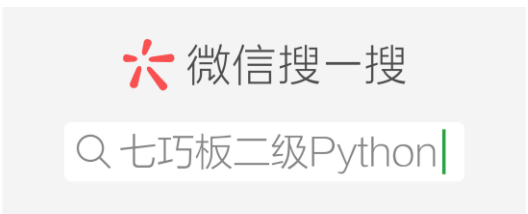
内置函数 sorted()函数与 sort()方法有一点不同，sort()会在原 list 的上重新排列并保存，而 sorted()不会改变原列表的顺序，只是生成新的排序列表

内置函数 sorted 举例：

```
>>> aa = [1,8,3,5]
>>> sorted(aa)
[1, 3, 5, 8]
>>> sorted(aa,reverse=True)
[8, 5, 3, 1]
>>> aa
[1, 8, 3, 5]
```

del 关键字（科普）

(1) 使用 del 关键字(delete) 可以删除列表中的部分元素，但是不能删除元组、字符串等不可变序列中的部分元素。



(2) del 关键字本质上是用来 **将一个变量从内存中删除的，使用 del 删除对象之后 Python 会再恰当的时机调用垃圾回收机制来释放内存。**

(3) 如果使用 del 关键字将变量从内存中删除，后续的代码就不能再使用这个变量了。

```
del name_list[1]
```

在日常开发中，要从列表删除数据，建议**使用列表提供的方法。**

关键字、函数和方法（科普）

(1) **关键字**是 Python 内置的、具有特殊意义的标识符

```
>>>import keyword
>>>print(keyword.kwlist)
>>>print(len(keyword.kwlist))
```

关键字后面不需要使用括号

(2) **函数**封装了独立功能，可以直接调用，调用格式如下:

函数名(参数)

函数需要死记硬背

(3) **方法**和函数类似，同样是封装了独立的功能

方法需要通过**对象**来调用，表示针对这个**对象**要做的操作

方法调用格式如下:

对象.方法名(参数)

在对象变量后面输入.，然后选择针对这个变量要执行的操作，记忆起来比函数要简单很多。在 Python 中字符串、列表、元组、集合和字典定义的变量都是对象。

10.2.3 循环遍历

(1) **遍历**就是**从头到尾依次**从**列表**中获取数据

- 在 **循环体内部**针对**每一个元素**，执行相同的操作

(2) 在 Python 中为了提高列表的遍历效率，专门提供的**迭代 iteration 遍历**

(3) 使用 for 就能够实现迭代遍历

for 循环遍历列表 1

```
# for 循环内部使用的变量 in 列表
for name in name_list:
    #循环内部针对列表元素进行操作
    print(name)
```

for 循环遍历列表 2(使用列表下标)

```
# for 循环内部使用的变量 in 列表
for i in range(len(name_list)):
    #循环内部针对列表下标获取元素进行操作
    print(name[i])
```

这种使用下标遍历列表的方法，适用于需要下标的情况，例如求最大值最小值的下标等。

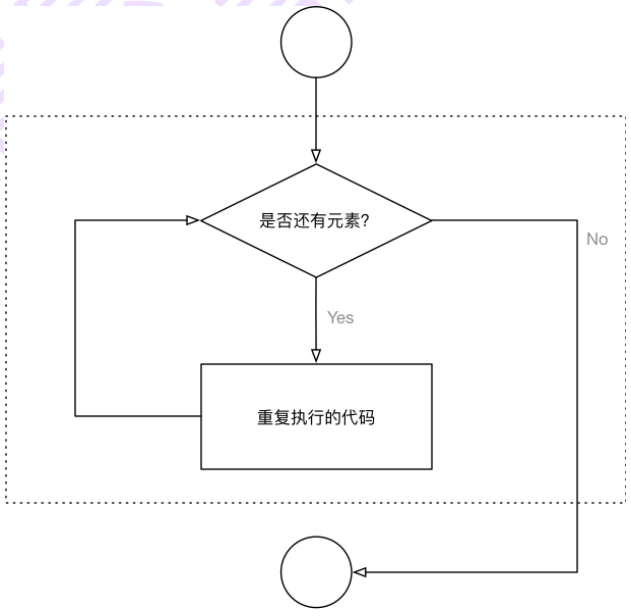


图 9-3 循环遍历

10.2.4 列表与字符串的转换

列表和字符串，前面已经学过，可以通过字符串的 join 方法和 split 方法，互相转换，举例代码如下：

(1) 使用 join()链接 list 成为字符串

```
>>> a = ['no','pain','no','gain']
>>> '_ '.join(a)
'no_pain_no_gain'
```

(2) 使用 split()分割字符串为列表

```
>>> a = 'no_pian_no_gain'
>>> a.split('_')
['no', 'pian', 'no', 'gain']
```

在日常任务处理，或者文件数据处理的时候，经常遇到固定字符分割的字符串，一般用 split()分割为字符串，而最终保存为文件或者要求组合成字符串的时候，一般用 join()把列表组合为字符串。

10.2.5 应用场景

尽管 Python 的**列表**中可以**存储不同类型的数据，** 但是在开发中，更多的应用场景是：

(1) **列表**存储相同类型的数据

(2) 通过**迭代遍历**，在循环体内部，针对列表中的每一项元素，执行相同的操作

10.3 元组

10.3.1 元组的定义

(1) Tuple（元组）与列表类似，不同之处在于元组的**元素不能修改**

1) **元组**表示多个元素组成的序列

2) **元组**在 Python 开发中，有特定的应用场景

(2) 用于存储**一串信息**，**数据**之间使用 ，分隔

(3) 元组用()定义

(4) 元组的**索引**从 0 开始

- **索引**就是数据在**元组**中的**位置编号**

定义一个元组：

```
info_tuple = ("zhangsan", 18, 1.75)
```

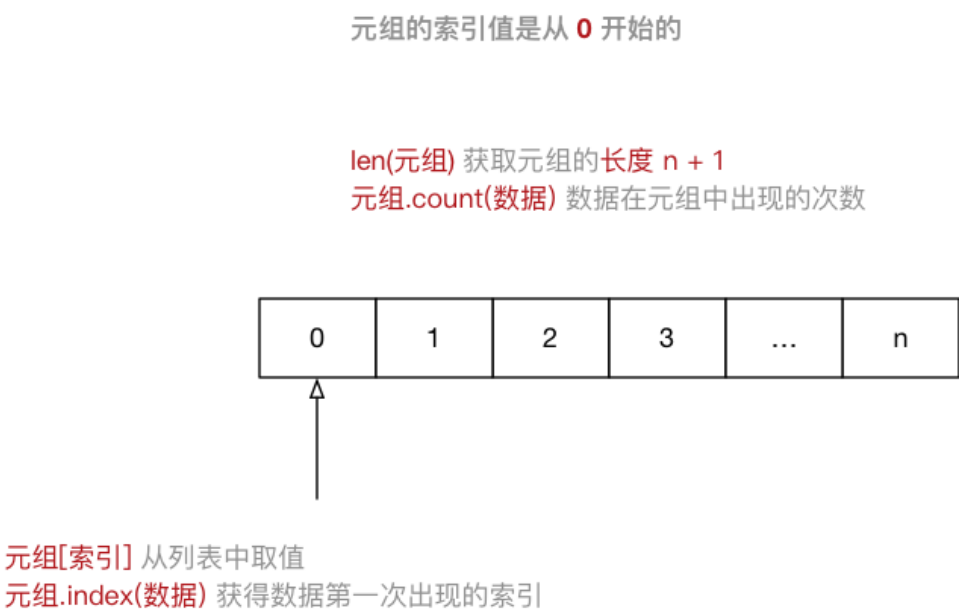


图 9-4 元组

创建空元组

```
info_tuple = ()
```

元组中只包含一个元素**时，需要**在元素后面添加逗号

```
info_tuple = (50, )
```

10.3.2 元组常用操作

在 ipython3 中定义一个**元组**，例如：info = ()

输入 info.按下 TAB 键，ipython 会提示**元组**能够使用的方法如下：

```
info.count    info.index
```

有关**元组**的**常用操作**，可以参照上图练习。

元组可以理解为“常量列表”，一旦确定，不允许对其修改，元组是不可变（immutable）类型。元组的访问和处理速度比列表更快，因此，不需要对元素进行修改、而只是遍历的情况下，建议使用元组而不是列表。

10.3.3 循环遍历

(1) **取值**：就是从**元组**中获取存储在指定位置的数据

(2) **遍历**：就是**从头到尾依次**从**元组**中获取数据

```
# for 循环内部使用的变量 in 元组
for item in info:
    循环内部针对元组元素进行操作
    print(item)
```

在 Python 中，可以使用 for 循环遍历所有非数字型类型的变量：**列表**、**元组**、**字典**以及**字符串**

- 提示：在实际开发中，除非**能够确认元组中的数据类型**，否则针对元组的循环遍历需求并不是很多。

10.3.4 封装与解构

在 Python 中，列表和元组有封装和解构的常用操作。封装和解构原理是先把等号右边的封装起来，再在左边进行复制，按照参数进行解构。这么描述，很难理解，下面详细举例说明。

(1) 封装

封装是将多个值使用逗号分隔，组合在一起，本质上，返回一个元组，只是省略掉了小括号。演练如下：

```
>>> test = 1,2,3,'a','b'
>>> test
(1, 2, 3, 'a', 'b')
>>> type(test)
<class 'tuple'>
```

(2) 解构

解构是把线性构成的元素解开，并顺序的赋给其他变量。左边接纳的变量数，要和右边解开的元素个数一致。演练如下：

```
>>> test=[1,'a','A',3]
>>> a,b,c,d = test
>>> a
1
>>> b
'a'
>>> c
'A'
>>> d
3
```

由此可以看到，前面学习过的同步赋值，实质上就是封装和解构的特殊形式。

解构的时候，使用“*变量名”接收，但不能单独使用，被“*变量名”收集后组成一个列表。演练如下：

```
>>> test=list(range(1,10))
>>> test
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> head, *mid, tail = test      #*mid 收集中间的变量组成列表
>>> head
1
>>> tail
9
>>> mid
[2, 3, 4, 5, 6, 7, 8]
```

10.3.5 应用场景

(1) 尽管可以使用 for in 遍历**元组**

(2) 但是在开发中，更多的应用场景是：

1) **函数的参数和返回值**，一个函数可以接收**任意多个参数**，或者**一次返回多个数据**

- 有关**函数的参数**和**返回值**，在后续**函数**一章给大家介绍

2) **格式字符串**，格式化字符串后面的()本质上就是一个元组

3) **让列表不可以被修改**，以保护数据安全

```
info = ("zhangsan", 18)
print("%s 的年龄是 %d" % info)
```

元组和列表之间的转换：

(1) 使用 list 函数可以把元组转换成列表

list(元组)

(2) 使用 tuple 函数可以把列表转换成元组

tuple(列表)

10.4 字典

10.3.1 字典的定义

(1) 字典（dictionary）是**除列表以外** Python 之中**最灵活**的数据类型

(2) 字典同样可以用来**存储多个数据**

- 通常用于存储**描述一个物体的相关信息**

(3) 和列表的区别

1) **列表**是**有序**的对象集合

2) **字典**是**无序**的对象集合

(4) 字典用 {} 定义

(5) 字典使用**键值对**存储数据，键值对之间使用“,”分隔，即逗号分割

1) **键** key 是索引

2) **值** value 是数据

3) **键**和**值**之间使用 ： 分隔

4) **键必须是唯一的**

5) **值可以取任何数据类型，但键只能是任意不可变数据，如字符串、数值或元组，不能使用列表、集合、字典等可变类型**

定义一个字典：

```
xiaoming = {"name": "小明",
            "age": 18,
            "gender": True,
            "height": 1.75}
```

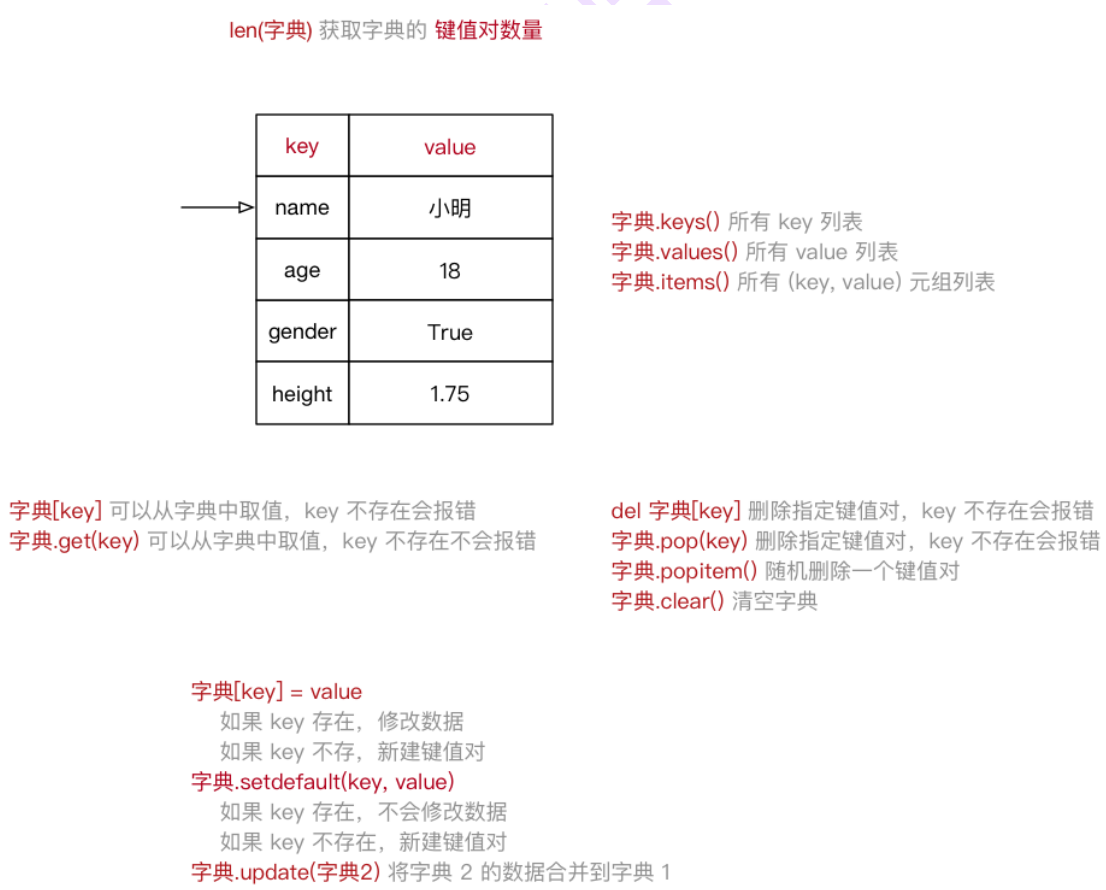


图 9-5 字典

10.4.2 字典常用操作

在 ipython3 的交互式命令行中定义一个**字典**，例如：xiaoming = {}，输入 xiaoming. 按下 TAB 键，ipython3 会提示**字典**能够使用的函数如下：

```
In [1]: xiaoming={}
In [2]: xiaoming.
xiaoming.clear          xiaoming.items          xiaoming.setdefault
xiaoming.copy           xiaoming.keys           xiaoming.update
xiaoming.fromkeys       xiaoming.pop            xiaoming.values
xiaoming.get            xiaoming.popitem
```

有关**字典**的**常用操作**，可以参照上图练习，主要包括字典的增删改查，和其他操作。

字典常用操作代码举例如下：

(1) 增加

```
>>>dic = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
>>>dic['James'] = '5124'     # 字典 dic 中有 James 这个键，新增
>>> print(dic)
{'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'James': '5124'}
>>>dic.setdefault('Jhon','1024')       # 如果在字典中存在就不做任何操作,不存在就添加
'1024'
>>> dic.setdefault(' Jhon ','9816')
'9816'
>>> print(dic)
{'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'James': '5124', 'Jhon': '1024', ' Jhon ': '9816'}
```

dic[key] = value 的功能是新增 key-value 对。dict.setdefault(key,value)的功能是，如果键在字典中存在 key 不进行任何操作,否则就添加，对此可以通过 key 查询，即 if dict[key] == None 成立，表示没有这个 key。

(2) 删除

字典删除单个 key-value 对可以使用 del 关键字、dict.pop()和 dict.popitem()，全部清除 key-value 对可以用 dict.clear()清空:

```
>>> scores = {'语文': 85, '数学': 68, '英语': 95}
>>> print(scores)
{'语文': 85, '数学': 68, '英语': 95}
>>> del scores[ '数学']
>>> print(scores)
{'语文': 85, '英语': 95}
>>> #  清空 scores 所有 key-value 对
>>> scores.clear()
>>> print(scores)
{}
>>> scores = {'Python': 85, '数学': 68, '英语': 95}
>>> scores.popitem() #随机删除  默认删除最后一个，返回值为一个元组  (key,value)
('英语', 95)
>>> print(scores)
{'Python': 85, '数学': 68}
>>> scores.pop ('Python') #返回值为 Python 键对应的值 85
85
>>> print(scores)
{'数学': 68}
```

(3) 修改

字典修改可以使用 dict [key] = 内容，修改的是 key 对应的 value。还可以使用 dict.update(字典) ，可以理解为更新，将新旧两个字典合并到一起，如果相同，新的会覆盖旧的。

```
>>> scores = {'Python': 85, '数学': 68, '英语': 95}
```

```
>>> scores ['Python'] = 95    #字典 scores 中有 Python 这个键，为修改
>>> print(scores)
{'Python': 95, '数学': 68, '英语': 95}
>>> sc = {'语文': 85, '数学': 72, '英语': 59}
>>> scores.update(sc)   #添加了语文，更新了数学和英语
>>> print(scores)
{'Python': 95, '数学': 72, '英语': 59, '语文': 85}
```

(4) 查询

字典查询可以使用 dict.get(key)和 dict [key]，也可以使用 for 循环遍历。如果字典 dict 中不存在 key，可以使用 dict.get(key,default)，即给出第二个参数，那么就返回 default 值，否则使用 dict.get(key)，即不给出第二个参数，返回 None。

```
>>> scores = {'Python': 85, '数学': 68, '英语': 95}
>>> print(scores ['Python'])
85
>>> print(scores.get('一个不存在的 key','你傻啊,没有!'))
你傻啊,没有!
>>> print(scores.get('一个不存在的 key'))   #没有第二个参数默认值
None
```

经常使用字典的 get 方法作为计数器使用。如下面的例子：

```
scores = [96,23,78,98,78,62,56]
result = {}
for score    in scores:
    if score>=60:
        result['及格'] = result.get('及格',0)+1
    else:
        result['不及格'] = result.get('不及格',0)+1
```

```
print(result)
```

运行结果：

```
{'及格': 5, '不及格': 2}
```

(5) 其他操作

字典其他操作有 dict.keys()获取到所有的键，存在一个列表中； dict.values()获取到所有的值存在一个列表中；dict.items()获取到所有的键值对，以元祖的形式存在一个列表中:

```
>>> scores={'Python': 95, '数学': 72, '英语': 59, '语文': 85}
>>> scores.keys()
dict_keys(['Python', '数学', '英语', '语文'])  # (高仿列表) 列表的操作都可以在其上面
>>> scores.values()
dict_values([95, 72, 59, 85])  # (高仿列表) 列表的操作都可以在其上面
>>> scores.items()
dict_items([('Python', 95), ('数学', 72), ('英语', 59), ('语文', 85)])
>>> for k in scores.keys():
        print(k)
Python
数学
英语
语文
>>> for v in scores.values():
        print(v)
95
72
59
85
>>> for k,v in scores.items():
```

```
        print(k,'::',v)
Python :: 95
数学  :: 72
英语  :: 59
语文  :: 85
```

10.4.3 循环遍历

遍历就是**依次**从**字典**中获取所有键值对

```
# for 循环内部使用的 `key 的变量` in 字典
for k in xiaoming:
    print("{}:{}" .format(k, xiaoming[k]))
```

- 提示：在实际开发中，由于字典中每一个键值对保存数据的类型是不同的，所以针对字典的循环遍历需求并不是很多。

10.4.4 应用场景

(1) 尽管可以使用 for in 遍历**字典**

(2) 但是在开发中，更多的应用场景是：

1) 使用**多个键值对**，存储**描述一个物体的相关信息**——描述更复杂的数据信息

2) 将**多个字典**放在**一个列表**中，再进行遍历，在循环体内部针对每一个字典进行**相同的处理**

```
info_list = [{"name": "张三",
              "qq": "12345",
              "phone": "110"},
             {"name": "李四",
              "qq": "54321",
              "phone": "10086"}
             ]
```

10.5 集合简介

集合（set）是一个无序的不重复元素序列。可以使用大括号 { } 或者 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 { } 是用来创建一个空字典。

10.5.1 集合运算

集合之间也可进行数学集合运算（例如：并集、交集等），可用相应的操作符或方法来实现。

10.5.1.1 子集

子集，为某个集合中一部分的集合，故亦称部分集合。使用操作符 < 执行子集操作，同样地，也可使用方法 issubset() 完成。

```
>>> A = set('abcd')
>>> B = set('cdef')
>>> C = set("ab")
>>> C < A
True                # C 是 A 的子集
>>> C < B
False
>>> C.issubset(A)
True
```

10.5.1.2 并集

一组集合的并集是这些集合的所有元素构成的集合，而不包含其他元素。使用操作符 | 执行并集操作，同样地，也可使用方法 union() 完成。

```
>>> A | B
{'c', 'b', 'f', 'd', 'e', 'a'}
>>> A.union(B)
{'c', 'b', 'f', 'd', 'e', 'a'}
```

10.5.1.3 交集

两个集合 A 和 B 的交集是含有所有既属于 A 又属于 B 的元素，而没有其他元素的集合。使用 & 操作符执行交集操作，同样地，也可使用方法 intersection() 完成。

```
>>> A & B
{'c', 'd'}
>>> A.intersection(B)
{'c', 'd'}
```

10.5.1.4 差集

A 与 B 的差集是所有属于 A 且不属于 B 的元素构成的集合。使用操作符 - 执行差集操作，同样地，也可使用方法 difference() 完成。

```
>>> A - B
{'b', 'a'}
>>> A.difference(B)
{'b', 'a'}
```

10.5.1.5 对称差

两个集合的对称差是只属于其中一个集合，而不属于另一个集合的元素组成的集合。使用 ^ 操作符执行差集操作，同样地，也可使用方法 symmetric_difference() 完成。

```
>>> A ^ B
{'b', 'f', 'e', 'a'}
>>> A.symmetric_difference(B)
{'b', 'f', 'e', 'a'}
```

10.5.2 集合方法

```
>>>seta =set()
>>>seta.
seta.add      seta.clear      seta.copy      seta.difference      seta.difference_update
```

```
seta.discard      seta.intersection      seta.intersection_update      seta.isdisjoint
seta.issubset     seta.issuperset        seta.pop      seta.remove     seta.symmetric_difference
seta.symmetric_difference_update      seta.union      seta.update
```

其中 add 是给集合增加元素，clear 为清空集合，pop 为随机删除元素，remove 为删除元素，discard 也为删除一个元素(但没该元素也不发生异常），update 是更新元素。

10.6 高级数据的其他用法

10.6.1 内置函数(公共方法)

Python 包含了以下高级数据的内置函数：

表 9-2 高级数据常用内置函数

函数	描述	备注
len(item)	计算容器中元素个数	
max(item)	返回容器中元素最大值	如果是字典，只针对 key 比较
min(item)	返回容器中元素最小值	如果是字典，只针对 key 比较
del item	删除变量	del 有两种方式
sorted(item)	对容器中元素进行排序	参数只能是可迭代的，如列表等
zip(item1, item2)	生成两个容器中元素一一对应的配对的元组	生成器，每个元组形如：(a,1)
map(func, item1)	对容器中每个元素都运行 func 函数	
filter(func, item1)	func 函数对容器中每个元素都过滤，为真保留	
enumerate (item1)	对容器中每个元素都组合为一个索引序列	
all(item)	容器中每个元素都为非 0 才为真	
any(item)	容器中只要有一个元素为非 0 就为真	
list(item)	转换函数，把容器对象转换为列表	
tuple(item)	转换函数，把容器对象转换为元组	
dict(item)	转换函数，把容器对象转换为字典	

函数	描述	备注
set(item)	转换函数，把容器对象转换为集合	
frozenset(item)	转换函数，把容器对象转换为不可变集合	

★注意：

- **字符串**比较符合以下规则： "0" < "A" < "a"

高级数据常用内置函数演练如下：

(1) len、min、max 函数

```
>>> len([1,2,3,4,0])
5
>>> len({2:3,3:4})
2
>>> min((1,9,8,0,-7,10))
-7
>>> min({2:3,3:4})
2
>>> max([1,9,8,0,-7,10])
10
>>> max({2:'a',3:'b',5:'d'})
5
```

(2) del 操作

```
#删除变量
>>> x=2
>>> y=3
>>> del x
>>> y
3
```

```
>>> x
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    x
NameError: name 'x' is not defined
#删除元素
>>> s=[1,2,3,4,5]
>>> del s[2]
>>> s
[1, 2, 4, 5]
>>> d={2:'a',3:'b',5:'d'}
>>> del d[3]
>>> d
{2: 'a', 5: 'd'}
```

(3) zip 函数

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> zip(a,b)
<zip object at 0x000001E17BCFDF88>
>>> list(zip(a,b))
[(1, 4), (2, 5), (3, 6)]
>>> for k,v in zip(a,b):
        print('{}: {}'.format(k,v))

1: 4
2: 5
```

```
3: 6
>>> names = ['张三','李四','王五']
>>> sexs = ['男','男','女']
>>> scores = [54, 99]
>>> for name, sex, score in zip(names,sexs,scores):
        print('{}, {}, {}'.format(name, sex, score))

张三, 男, 54
李四, 男, 99
```

(4) enumerate 函数

```
>>> lst = [1, 2, 3, 4, 10, 5]
>>> enumerate(lst)
<enumerate object at 0x0000022857C775A0>
>>> num = ['one','two','three','four','five','six']
>>> for index, value in enumerate(num):
        print('{}:{}'.format(index,value))

0:one
1:two
2:three
3:four
4:five
5:six
#指定索引起始值为 1
>>> for index,value in enumerate(lst,1):
        print('{}:{}'.format(index,value))
```

```
1:1
2:2
3:3
4:4
5:10
6:5
#字典
>>> d1 = {1:11,2:22,3:33}
>>> d2 = {5:55,6:66,7:77}
>>> list(zip(d1,d2))
[(1, 5), (2, 6), (3, 7)]
```

(5) map 函数

```
>>> map(lambda x, y: x+y,[1,3,5,7,9],[2,4,6,8,10])
<map object at 0x000001EF9BA80128>
>>> list(map(lambda x, y: x+y,[1,3,5,7,9],[2,4,6,8,10]))
[3, 7, 11, 15, 19]
>>> def f(x):
        return x*x

>>> map(f,(0,1,2,3,4,5))
<map object at 0x000001EF9BA85B38>
>>> list(map(f,(0,1,2,3,4,5)))
[0, 1, 4, 9, 16, 25]
>>> list(map(int,'012345'))
[0, 1, 2, 3, 4, 5]
```

```
>>> list(map(int, {1:2,2:3,3:4}))
[1, 2, 3]
```

(6) filter 函数

```
>>> def odd(x):
        return x%2==0

>>> list(filter(odd, [1,2,3,4]))
[2, 4]
>>> filter(odd,[1,2,3,4])
<filter object at 0x0000013D48EB4278>
>>> scores={'张三':45,'李四':99,'王五':62,'李晓丽':82,'陈福明':60,'马晓华':69}
>>> r = filter(lambda score: score>=60,scores.values())
>>> r
<filter object at 0x0000013D48EB4F28>
>>> list(r)
[99, 62, 82, 60, 69]
```

(7) all 和 any 函数

```
>>> all((1,1,0))
False
>>> all({"", 1, 1})
False
>>> all([" ", 1, 3])
True
>>> all((" "))
True
>>> all(("))
```

```
True
>>> all([True,True,True,True])
True
>>> all([True,True,True,False])
False
>>> all(['a', 'b', '', 'd'])
False
>>> all([0, 1,2, 3])
False
>>> all([])
True
>>> all(())
True
>>> any({"", 1, 1})
True
>>> any((1,1,0))
True
>>> any(("",))
False
>>> any((" "))
True
>>> any([True,True,True,False])
True
>>> any("")
False
>>> any('123')
True
```

(8) tuple 函数

```
>>>tuple([1,2,3,4])
(1, 2, 3, 4)
>>> tuple({1:2,3:4})       #对于字典，会返回字典的 key 组成的 tuple
(1, 3)
```

(9) dict 函数

```
>>> # 创建空字典
>>> dict()
{}
>>> dict(a='a', b='b', c='c')
{'a': 'a', 'b': 'b', 'c': 'c'}
>>> dict(zip([1, 2, 3],['one', 'two', 'three'] ))
{1: 'one', 2: 'two', 3: 'three'}
>>> dict([('a', 1), ('b', 2), ('c', 3)])
{'a': 1, 'b': 2, 'c': 3}
```

10.6.2 序列切片

表 9-3 切片

描述	Python 表达式	结果	支持的数据类型
切片	"0123456789"[::-2]	"97531"	字符串、列表、元组

(1) **切片**：使用**索引值**来限定范围，从一个大的**字符串**中**切出**小的**字符串**

(2) **列表**和**元组**都是**有序**的集合，都能够**通过索引值**获取到对应的数据，也可以切片

(3) **集合**是**无序**的，不可以切片

(4) **字典**是一个**无序**的集合，是使用**键值对**保存数据，也不可以切片

列表与元组的切片类似于字符串，下面对列表切片进行演练：

```
>>> lst=[1,2,3,4,5,6,7,8,9,0]
>>> lst[:2]
```

```
[1, 2]
>>> lst[:-1]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lst[1:-1]
[2, 3, 4, 5, 6, 7, 8, 9]
>>> lst[::-1]
[0, 9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> lst[::-2]
[0, 8, 6, 4, 2]
```

10.6.3 高级数据运算符

表 9-4 运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	["Hi!"] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典
not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典
> >= == < <=	(1, 2, 3) < (2, 2, 3)	True	元素比较	字符串、列表、元组

★注意：

(1) in 在对**字典**操作时，判断的是**字典的键**

(2) in 和 not in 被称为**成员运算符**

成员运算符

成员运算符用于**测试**序列中是否包含指定的**成员**

表 9-5 成员运算符

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False	3 in (1, 2, 3) 返回 True
not in	如果在指定的序列中没有找到值返回 True，否则返回 False	3 not in (1, 2, 3) 返回 False

★注意：在对**字典**操作时，判断的是**字典的键**

高级数据运算符演练如下：

(1) +运算符

```
>>> lst1=[1,2,3,4]
>>> lst2=[7,8,9]
>>> lst1+lst2
[1, 2, 3, 4, 7, 8, 9]
```

(2) *运算符

```
>>> lst=[1,2,3,4]
>>> lst*2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> tpl=(1,2,3,4)
>>> tpl*2
(1, 2, 3, 4, 1, 2, 3, 4)
```

(3) in 和 not in 运算符

```
>>> lst=[1,2,3,4]
>>> 2 in lst
True
>>> 9 in lst
```

```
False
>>> 'a' in lst
False
>>> 'b' not in lst
True
>>> lst2=['a','b','c','e']
>>> 'a' in lst2
True
>>> 'd' in lst2
False
>>> 'cc' in lst2
False
#字典
>>> d={1:11,2:22,3:33,5:55,9:99}
>>> 22 in d
False
>>> 2 in d
True
>>> 33 not in d
True
>>> 7 not in d
True
>>> 55 not in d
True
```

(4) 比较运算符

```
>>> lst1=[1,2,3,5]
```

```
>>> lst2=[1,2,3,7]
>>> lst3=[1,2]
>>> lst1>lst2
False
>>> lst1<lst2
True
>>> lst1>lst3
True
>>> lst2>lst3
True
>>> tpl1=(1,2,3,5)
>>> tpl2=(1,2,3,7)
>>> tpl3=(1,2)
>>> tpl1>tpl2
False
>>> tpl1<tpl2
True
>>> tpl1>tpl3
True
>>> tpl2>tpl3
True
>>> lst4=[1,2,3,5]
>>> lst1==lst4
True
>>> tpl4=(1,2,3,5)
>>> lst1==list(tpl4)
True
```

10.6.4 推导式与生成器

Python 高级数据类型中，包含列表推导式，字典推导式和集合推导式，下面分别介绍列表推导式、字典推导式和集合推导式：

10.6.4.1 列表推导式

(1) 列表推导式书写形式

[表达式 for 变量 in 列表]　或者　[表达式 for 变量 in 列表 if 条件]

(2) 列表推导式举例说明

```
li = [1,2,3,4,5,6,7,8,9]
print( [x**2 for x in li] )
print([x**2 for x in li if x>5]
print(dict([(x,x*10) for x in li]) )
print( [ (x, y) for x in range(10) if x % 2 if x > 3 for y in range(10) if y > 7 if y != 8 ] )

vec=[2,4,6]
vec2=[4,3,-9]
sq = [vec[i]+vec2[i] for i in range(len(vec))] )
print(sq)

print( [x*y for x in [1,2,3] for y in    [1,2,3]] )

testList = [1,2,3,4]
def mul2(x):
    return x*2
print ([mul2(i) for i in testList] )
```

运行结果：

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
[36, 49, 64, 81]
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 7: 70, 8: 80, 9: 90}
[(5, 9), (7, 9), (9, 9)]
[6, 7, -3]
[1, 2, 3, 2, 4, 6, 3, 6, 9]
[2, 4, 6, 8]
```

10.6.4.2 字典推导式

(1) 字典推导式书写形式

{表达式 1:表达式 2 for 变量 1 in 字典或者列表}

或者 {表达式 1:表达式 2 for 变量 1 in 字典或者列表 if 条件}

(2) 字典推导式举例说明

快速更换 key 和 value：

```
mcase = {'a': 10, 'b': 34}
mcase_frequency = {v: k for k, v in mcase.items()}
print(mcase_frequency)
```

运行结果：

```
{10: 'a', 34: 'b'}
```

10.6.4.3 集合推导式

(1) 集合推导式书写形式

{表达式 for 变量 in 字典或者列表}

或者 [表达式 for 变量 in 字典或者列表 if 条件]

(2) 集合推导式举例说明

```
squared = {x**2 for x in [1, 3, 2]}
print(squared)
```

运行结果：

```
{1, 4, 9}
```

10.6.4.4 生成器

我们前面学过的 for i in range(100)，在 Python3.x 中 range(100)就是生成器，每取一次生成一个数，因此在 Python3.x 中用 print(range(100))是打印不出来 0 到 99 的(可以用 print(list(range(100))))。

生成器 generator 是为了节约内存设计的，比如 range 内的参数为 100 万，那么直接生成 100 万个数，内存会严重浪费的。

要创建一个生成器，有很多种方法，第一种方法很简单，只要把一个列表推导式的[]中括号改为()小括号，就创建一个生成器，或者用 iter()内置函数也可以通过列表创建一个生成器。下面演示了生成器的简单用法:

```
#生成器
generator_ex = (i for i in range(10))
print(next(generator_ex))
print(next(generator_ex))
print(next(generator_ex))
print(next(generator_ex))
```

运行结果：

```
0
1
2
3
```

用 iter()内置函数也可以把列表转换为生成器，演示如下：

```
>>> lst=[1,2,3,4,0]
>>> itr = iter(lst)
```

```
>>> next(itr)
1
>>> next(itr)
2
>>> next(itr)
3
>>> next(itr)
4
>>> next(itr)
0
>>> next(itr)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    next(itr)
StopIteration
```

一般我们都用 for 循环访问生成器，如 for gen in generator_ex: print(gen)，类似与 range 的使用。

生成器是一个特殊的程序，可以被用作控制循环的迭代行为，Python 中生成器是迭代器的一种。使用 yield 返回值函数，每次调用 yield 会暂停，而可以使用 next()函数和 send()函数恢复生成器。

因此，创建一个生成器第二种方法：yield 关键字。把 yield 放在函数中，那么带有 yield 的函数不再是一个普通函数，而是一个**生成器**。**yield 相当于 return 返回一个值，并且记住这个返回的位置，下次迭代时，代码从 yield 的下一条语句开始执行**。下面演示了 yield 生成器的简单用法:

```
def yield_test(n):
    for i in range(n):
```

```
            yield i
            print("i=",i)
        print("Done.")
for i in yield_test(3):
        print(i,",")
```

结果：

```
0 ,
i= 0
1 ,
i= 1
2 ,
i= 2
Done.
```

10.6.5 函数的可变参数

在函数一章中，大家知道了 Python 支持定义参数个数可变的函数。实现参数个数可变可以采用三种方式：参数默认值、可变参数*args 和字典参数**kargs。可变参数*args 方法是把这些参数包装进一个元组里(包装在列表里也可以，只是调用的时候，会自动转换为元组)。在这些可变个数的参数之前，可以有零到多个普通的参数。 此外参数也可以是字典，如果是字典，要在字典变量名前加两个星号，即** dicts，其中 dicts 为字典。普通参数，元组参数(列表参数)和字典参数在一起的时候，Python 要求普通参数在前，元组参数(列表参数)在中间，字典参数在最后，例如 def func(x,y,*tuples,**dicts)。

可变参数演示：

```
def multi_sum( x, *args ):
        s = x * sum( args )
        return s
print (multi_sum (2, 3, 4, 5) )
```

运行结果：

```
24
```

其中参数 3，4，5 就是可变参数。

下面的综合例子演示了使用列表或者元组作为实参调用函数：

```
>>> l=[1,2,3]
>>> t=(4,5,6)
>>> d={'a':7,'b':8,'c':9}
>>> def test(a,*args):
        print(a,args)
>> test(1,l)
1 ([1, 2, 3],)
>>> test(2,*l)
2 (1, 2, 3)
>>> test(3,t)
3 ((4, 5, 6),)
>>> test(4,*t)
4 (4, 5, 6)
```

下面进一步演示了普通参数，列表或者元组参数以及字典参数的混合使用与调用：

```
# 带一个*的是元组，带两个星号的是字典。
def func(x=5,*tuples,**dicts):
    print('x=',x)
    for item in tuples:
        print('元组元素',item)
    for first_part,second_part in dicts.items():
        print(first_part,second_part)
func(20,5,6,7,Jack=1523,James=1235,Alice=1760)
```

运行结果：

```
x= 20
元组元素: 5
元组元素: 6
元组元素: 7
Jack 1523
```

```
James 1235
Alice 1760
```

上面的演示可以看出，也可以用**dicts 这样的字典参数，作为函数的默认值参数使用。

习 题

一、填空题

1、列表、元组、字符串是 Python 的_____(有序？无序)序列。

2、使用运算符测试集合包含集合 A 是否为集合 B 的真子集的表达式可以写作_____。

3、表达式[1, 2, 3]*3 的执行结果为_____。

4、list(map(str, [1, 2, 3]))的执行结果为_____。

5、语句 x = 3==3, 5 执行结束后，变量 x 的值为_____。

6、表达式“[3] in [1, 2, 3, 4]”的值为_____。

7、列表对象的 sort()方法用来对列表元素进行原地排序，该函数返回值为_____。

8、假设列表对象 aList 的值为[3, 4, 5, 6, 7, 9, 11, 13, 15, 17]，那么切片 aList[3:7]得到的值是_____。

9、使用列表推导式生成包含 10 个数字 5 的列表，语句可以写为_____。

10、假设有列表 a = ['name', 'age', 'sex']和 b = ['Dong', 38, 'Male']，请使用一个语句将这两个列表的内容转换为字典，并且以列表 a 中的元素为“键”，以列表 b 中的元素为“值”，这个语句可以写为_____。

11、任意长度的 Python 列表、元组和字符串中最后一个元素的下标为_____。

12、Python 语句".join(list('hello world!'))执行的结果是_____。

13、Python 语句 list(range(1,10,3))执行结果为_____。

14、表达式 list(range(5)) 的值为_____。

15、_____命令既可以删除列表中的一个元素，也可以删除整个列表。

16、已知 a = [1, 2, 3]和 b = [1, 2, 4]，那么 id(a[1])==id(b[1])的执行结果为_____。

17、切片操作 list(range(6))[::2]执行结果为_____。

18、使用切片操作在列表对象 x 的开始处增加一个元素 3 的代码为_____。

19、语句 sorted([1, 2, 3], reverse=True) == reversed([1, 2, 3])执行结果为_____。

20、表达式 sorted([111, 2, 33], key=lambda x: len(str(x))) 的值为_____。

21、达式 sorted([111, 2, 33], key=lambda x: -len(str(x))) 的值为_____。

22、Python 内置函数_____可以返回列表、元组、字典、集合、字符串以及 range 对象中元素个数。

23、Python 内置函数_____用来返回序列中的最大元素。

24、Python 内置函数_____用来返回序列中的最小元素。

25、Python 内置函数_____用来返回数值型序列中所有元素之和。

26、已知列表对象 x = ['11', '2', '3']，则表达式 max(x) 的值为_____。

27、表达式 min(['11', '2', '3']) 的值为_____。

28、已知列表对象 x = ['11', '2', '3']，则表达式 max(x, key=len) 的值为_____。

29、字典中多个元素之间使用_____分隔开，每个元素的“键”与“值”之间使用_____分隔开。

30、字典对象的_____方法可以获取指定“键”对应的“值”，并且可以在指定“键”不存在的时候返回指定值，如果不指定则返回 None。

31、字典对象的_____方法返回字典中的“键-值对”列表。

32、字典对象的_____方法返回字典的“键”列表。

33、字典对象的_____方法返回字典的“值”列表。

34、已知 x = {1:2}，那么执行语句 x[2] = 3 之后，x 的值为_____。

35、表达式 {1, 2, 3, 4} - {3, 4, 5, 6}的值为_____。

36、表达式 set([1, 1, 2, 3])的值为_____。

37、使用列表推导式得到 100 以内所有能被 13 整除的数的代码可以写作_____。

38、已知 x = {'a':'b', 'c':'d'}，那么表达式 'a' in x 的值为_____。

39、已知 x = {'a':'b', 'c':'d'}，那么表达式 'b' in x 的值为_____。

40、已知 x = {'a':'b', 'c':'d'}，那么表达式 'b' in x.values() 的值为_____。

41、已知 x = [3, 5, 7]，那么表达式 x[10:]的值为_____。

42、已知 x = [3, 5, 7]，那么执行语句 x[len(x):] = [1, 2]之后，x 的值为_____。

43、已知 x = [3, 7, 5]，那么执行语句 x.sort(reverse=True)之后，x 的值为_____。

44、已知 x = [3, 7, 5]，那么执行语句 x = x.sort(reverse=True)之后，x 的值为_____。

45、已知 x = [1, 11, 111]，那么执行语句 x.sort(key=lambda x: len(str(x)), reverse=True) 之后，x 的值为_____。

46、表达式 list(zip([1,2], [3,4])) 的值为_____。

47、已知 x = [1, 2, 3, 2, 3]，执行语句 x.pop() 之后，x 的值为_____。

48、表达式 list(map(list,zip(*[[1, 2, 3], [4, 5, 6]]))) 的值为_____。

49、表达式 [x for x in [1,2,3,4,5] if x<3] 的值为_____。

50、表达式 [index for index, value in enumerate([3,5,7,3,7]) if value == max([3,5,7,3,7])] 的值为_____。

51、已知 x = [3,5,3,7]，那么表达式 [x.index(i) for i in x if i==3] 的值为_____。

52、已知列表 x = [1, 2]，那么表达式 list(enumerate(x)) 的值为_____。

53、已知 vec = [[1,2], [3,4]]，则表达式 [col for row in vec for col in row] 的值为_____。

54、已知 vec = [[1,2], [3,4]]，则表达式 [[row[i] for row in vec] for i in range(len(vec[0]))] 的值为_____。

55、已知 x = list(range(10))，则表达式 x[-4:] 的值为_____。

56、已知 path = r'c:\test.html'，那么表达式 path[:-4]+'htm' 的值为_____。

67、已知 x = [3, 5, 7]，那么执行语句 x[1:] = [2]之后，x 的值为_____。

58、已知 x = [3, 5, 7]，那么执行语句 x[:3] = [2]之后，x 的值为_____。

59、已知 x 为非空列表，那么执行语句 y = x[:]之后，id(x[0]) == id(y[0])的值为_____。

二、判断题

1、Python 支持使用字典的“键”作为下标来访问字典中的值。（ ）

2、列表可以作为字典的“键”。（ ）

3、元组可以作为字典的“键”。（ ）

4、字典的“键”必须是不可变的。（ ）

5、在 Python 3.5 中运算符+不仅可以实现数值的相加、字符串连接，还可以实现列表、元组的合并和集合的并集运算。（ ）

6、已知 x 为非空列表，那么表达式 sorted(x, reverse=True) == list(reversed(x)) 的值一定是 True。（ ）

7、已知 x 为非空列表，那么 x.sort(reverse=True)和 x.reverse()的作用是等价的。（ ）

8、生成器推导式比列表推导式具有更高的效率，推荐使用。（ ）

9、Python 集合中的元素不允许重复。（ ）

10、Python 集合可以包含相同的元素。（ ）

11、Python 字典中的“键”不允许重复。（ ）

12、Python 字典中的“值”不允许重复。（ ）

13、Python 集合中的元素可以是元组。（ ）

14、Python 集合中的元素可以是列表。（ ）

15、Python 字典中的“键”可以是列表。（ ）

16、Python 字典中的“键”可以是元组。（ ）

17、Python 列表中所有元素必须为相同类型的数据。（ ）

18、Python 列表、元组、字符串都属于有序序列。（ ）

19、已知 A 和 B 是两个集合，并且表达式 A<B 的值为 False，那么表达式 A>B 的值一定为 True。（ ）

20、列表对象的 append()方法属于原地操作，用于在列表尾部追加一个元素。（ ）

三、程序题

1、编写程序，生成一个包含 20 个随机整数的列表，然后对其中偶数下标的元素进行降序排列，奇数下标的元素不变。（提示：使用切片。）

2、从键盘输入一个列表，计算输出列表元素的平均值。示例如下：

输入：[2,3,5,7] **输出：**平均值为： 4.25

3、编写函数，模拟 Python 内置函数 sorted()。

4、编写函数，给定任意字符串，找出其中只出现一次的字符，如果有多个这样的字符，就全部找出。

5、一个班选举班长，选举票放在一个字符串中，如：votes="张力,王之夏,Jams,韩梅梅,John,韩梅梅,john ,韩梅梅"，请编程统计每个人的得票数与字典中，并打印结果。提示：字典样例如：result={"张力":3, "王之夏":3, " Jams ":3, "韩梅梅":10, " John ":8}。

6、写出下面代码的执行结果_____。

```
def Join(List, sep=None):
    return (sep or ',').join(List)
print(Join(['a', 'b', 'c']))
print(Join(['a', 'b', 'c'],':'))
```

7、下面程序的执行结果是_____。

```
s = 0
for i in range(1,101):
    s += i
    if i == 50:
        print(s)
        break
else:
    print(1)
```

第 11 章 文件的使用

知识点

(1) 文件操作

(2) 内置库 os, os.path

(3) 文件数据处理

11.1 文件操作

为了长期保存数据以便重复使用、修改和共享，必须将数 据以文件的形式存储到外部存储介质(如磁盘、U 盘、光盘 或云盘、网盘、快盘等)中。文件操作在各类应用软件的开发中均占有重要的地位。

11.1.1 文件分类

按文件中数据的组织形式把文件分为文本文件和二进制文件两类。

(1) 文本文件：文本文件存储的是常规字符串，由若干文本行组成，通常每行以换行符'\n'结尾。**常规字符串**是指记事本或其他文本编辑器能正常显示、编辑并且人类能够直接阅读和理解的字符串，如英文字母、汉字、数字字符串。文本文件可以使用字处理软件如 gedit、记事本进行编辑。

(2) 二进制文件：二进制文件把对象内容以**字节串**(bytes)进行存储，无法用记事本或其他普通字处理软件直接进行编辑，通常也无法被人类直接阅读和理解，需要使用专门的软件进行解码后读取、显示、修改或执行。常见的 如图形图像文件、音视频文件、可执行文件、资源文件、各种数据库文件、各类 office 文档等都属于二进制文件。

11.1.2 文件内容操作

文件内容操作三个步骤：打开、读写、关闭。文件打开使用内置函数 open，open 函数的详细格式如下：

open(file, mode='r', buffering=1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

(1) **文件名** file 指定了被打开的文件名称。

(2) **打开模式** mode 指定了打开文件后的处理方式。

(3) **缓冲区** buffering 指定了读写文件的缓存模式。0 表示不缓存，1 表示缓存，如大于 1 则表示缓冲区的大小。默认值是缓存模式。

(4) 参数 **encoding** 指定对文本进行编码和解码的方式，只适用于文本模式，可以使用 Python 支持的任何格式，如 GBK、utf8、CP936 等等。

(5) 如果执行正常，open()函数返回 1 个**可迭代的文件对象**， 通过该文件对象可以对文件进行读写操作。如果指定**文件不存在**、**访问权限不够**、**磁盘空间不够**或其他原因导致创 建文件对象失败则**抛出异常**。

下面的代码分别以读、写方式打开了两个文件并创建了与之对应的文件对象。

```
f1 = open( 'file1.txt', 'r' )
f2 = open( 'file2.txt', 'w')
```

当对文件内容操作完以后，一定要关闭文件对象，这样才能保证所做的任何修改都确实被保存到文件中。即要有：

```
f1.close()
f2.close()
```

需要注意的是，即使写了关闭文件的代码，也无法保证文件一定能够正常关闭。例如，如果在打开文件之后和关闭 文件之前发生了错误导致程序崩溃，这时文件就无法正常 关闭。在管理文件对象时推荐使用 **with 关键字**，可以有效地避免这个问题。

(1) 用于文件内容读写时，with 语句的用法如下：

```
with open(filename, mode, encoding) as fp:
    #这里写通过文件对象 fp 读写文件内容的语句
```

(2) 另外，上下文管理语句 with 还支持下面的用法，进一步简化了代码的编写。

```
with open('test.txt', 'r') as src, open('test_new.txt', 'w') as dst:
    dst.write(src.read())
```

11.1.3 文件打开方式

open 函数的第二个参数是 mode，这个参数，就是表明文件的打开方式的，下表给出了 mode 参数可以使用的值及其含义。

表 11-1 文件打开方式

模式	说明
r	读模式（**默认模式**，可省略），如果文件不存在则抛出异常
w	写模式，如果文件已存在，先清空原有内容
x	写模式，创建新文件，如果文件已存在则抛出异常
a	追加模式，不覆盖文件中原有内容
b	二进制模式（可与其他模式组合使用）
t	文本模式（**默认模式**，可省略）
+	读、写模式（可与其他模式组合使用）

11.1.4 文件对象的属性

一个文件被打开后，有一个 file 对象，通过它可以得到有关该文件的各种信息。以下是和 file 对象相关的所有属性的列表：

表 11-2　文件(file)对象的属性

属性	描述
file.closed	返回 true,如果文件已关闭，则返回 false
file.mode	返回被打开文件的访问模式
file.name	返回文件的名称
file.softspace	如果用 print 输出后，必须跟一个空格符，返回 true，否则返回 false

查看一个文件对象的基本的属性的演示：

```
# 打开一个文件
fo = open("pythonlearning.txt", "w")
print( "文件名: ", fo.name)
print ("是否已关闭 : ", fo.closed)
print ("访问模式 : ", fo.mode)
print ("末尾是否强制加空格 : ", fo.softspace)
fo.close()
```

运行结果：

```
文件名: pythonlearning.txt
是否已关闭: False
访问模式: w
末尾是否强制加空格: 0
```

上面例子中的最后一行是文件对象 close()方法，这个方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。当一个文件对象的引用被重新指定给另一个文件时，Python 会自动关闭之前的文件。用 close()方法关闭文件是一个很好的习惯。下面进一步介绍文件对象的常用方法。

11.1.4 文件对象常用方法

表 11-3 文件(file)对象常用方法

方法	功能说明
close()	把缓冲区的内容写入文件，同时关闭文件，并释放文件对象
detach()	分离并返回底层的缓冲，底层缓冲被分离后，文件对象不再可用，不允许做任何操作
flush()	把缓冲区的内容写入文件，但不关闭文件
read([size])	从文本文件中读取 size 个字符（Python 3.x）的内容作为结果返回，或从二进制文件中读取指定数量的字节并返回，如果省略 size 则表示读取所有内容
readable()	测试当前文件是否可读
readline()	从文本文件中读取一行内容作为结果返回
readlines()	把文本文件中的每行文本作为一个字符串存入列表中，返回该列表，对于大文件会占用较多内存，不建议使用
seek(offset[, whence])	把文件指针移动到新的位置，offset 表示相对于 whence 的位置。whence 为 0 表示从文件头开始计算，1 表示从当前位置开始计算，2 表示从文件尾开始计算，默认为 0

seekable()	测试当前文件是否支持随机访问，如果文件不支持随机访问，则调用方法 seek()、 tell()和 truncate()时会抛出异常
tell()	返回文件指针的当前位置
truncate([size])	删除从当前指针位置到文件末尾的内容。如果指定了 size，则不论指针在什么位置都 只留下前 size 个字节，其余的一律删除
write(s)	把 s 的内容写入文件
writable()	测试当前文件是否可写
writelines(s)	把字符串列表写入文本文件，不添加换行符

下面对文件对象的最常用方法，举例说明：

文件的打开与关闭的演示：

```
# 打开一个文件
fo = open("pythonlearning.txt", "w")
print ("文件名: ", fo.name)
# 关闭打开的文件
fo.close()
```

write()方法可将任何字符串写入一个打开的文件，Python 字符串可以是二进制数据，或者是文字。write()方法不会在字符串的结尾自动添加换行符('\n')：

写入文件(write) 的演示：

```
# 打开一个文件
fo = open("pythonlearning.txt", "w")
fo.write( "www.pythonlearning.com!\nVery good site for learing python!\n")
# 关闭打开的文件
fo.close()
```

read()方法从一个打开的文件中读取一个字符串，Python 字符串可以是二进制数据，或者是文字。

读出文件的演示：

```
# 打开一个文件
```

```
fo = open("pythonlearning.txt", "r+")
str = fo.read(22)
print ("读取的字符串是 : {}" .format(str) )
# 关闭打开的文件
fo.close()
```

tell()方法返回文件指针的当前位置，即指出文件对象操作文件的当前位置，换句话说，下一次的读写会发生在文件开头这么多字节之后。

seek(offset[,whence])方法把文件指针移动到新的位置，即改变当前文件操作位置。offset 变量表示要移动的字节数。whence 变量指定开始移动字节的参考位置。 如果 whence 被设为 0，这意味着将文件的开头作为移动字节的参考位置。如果设为 1，则使用当前的位置作为参考位置。如果它被设为 2，那么该文件的末尾将作为参考位置。

文件位置重新定位的演示：

```
# 打开一个文件
fo = open("pythonlearning.txt", "r+")
str = fo.read(22)
print ("读取的字符串是 : {}".format(str) )
# 查找当前位置
position = fo.tell()
print ("当前文件位置 : ", position)
# 把指针再次重新定位到文件开头
position = fo.seek(0, 0)
str = fo.read(22)
print ( "重新读取字符串 : ", str)
# 关闭打开的文件
fo.close()
```

运行结果：

```
读取的字符串是 :  www.pythonlearning.com
```

```
当前文件位置 ： 22
重新读取字符串 ： www.pythonlearning.com
```

11.2 文件的内置库

11.2.1 os 模块常用的文件操作函数

表 11-4 os 模块常用的文件操作函数

方法	功能说明
rename(path1, path2)	文件或文件夹重命名，path1 改名为 path2
remove(file)	删除指定文件 file
access(path, mode)	测试是否可以按照 mode 指定的权限访问文件
chmod(path, mode, *, dir_fd=None, follow_symlinks=True)	改变文件的访问权限
name	判断现在正在实用的平台，Windows 返回 ‘nt'; Linux 返回’posix'
linesep	获取操作系统的换行符号，window 为 \r\n，linux/unix 为 \n
stat(file)	获得文件 file 属性
system(str)	运行 shell 命令，str 为命令字符串

重命名一个已经存在的文件的演示：

```
import os
# 将文件名“test1.txt”改为“test2.txt”
os.rename( "test1.txt", "test2.txt" )
```

删除一个已经存在的文件

```
import os
# 删除一个已经存在的文件 test2.txt
```

```
os.remove("test2.txt")
```

11.2.2 os 模块常用的文件夹操作函数

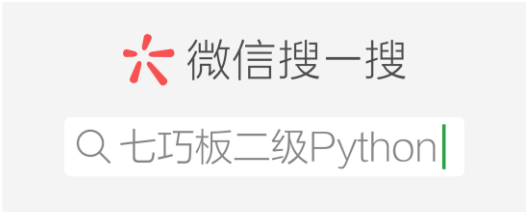
表 11-5 os 模块常用的文件夹操作函数

函数名称	使用说明
mkdir(path[, mode=0o777])	创建文件夹，要求上级文件夹必须存在
makedirs(path1/path2…, mode=511)	创建多级文件夹，会根据需要自动创建 中间缺失的文件夹
rmdir(path)	删除文件夹，要求该文件夹中不能有文件或子文件夹
removedirs(path1/path2…)	删除多级文件夹
listdir(path)	返回指定文件夹下所有文件信息
getcwd()	返回当前工作文件夹
chdir(path)	把 path 设为当前工作文件夹
curdir	当前文件夹
sep	取系统路径间隔符号，window 为\，linux 为/
walk(top, topdown=True, onerror=None)	遍历文件夹树，该方法返回一个元组，包括 3 个元素：所有路径名、所有文件夹列表与文件列表

使用 listdir 查看某个文件夹下的所有文件和文件夹的演示：

```
import os, sys
# 设置文件路径变量 path
path = "../"   #父文件夹
dirs = os.listdir( path )
# 输出所有文件和文件夹
for file in dirs:
    print file
```

获取当前路径及路径下全部文件的演示：



```
import os
os.getcwd()     # 获取当前路径
os.listdir(os.getcwd()) # 列举当前路径下的文件
```

使用 os.walk 遍历当前文件夹的演示：

```
import os
for path, dirs, files in os.walk(".", topdown=False):
    print("文件:")
    for name in files:
        print(os.path.join(path, name))
    print("文件夹:")
    for name in dirs:
        print(os.path.join(path, name))
```

11.2.3 os.path 常用的文件和文件夹操作函数

表 11-5 os.path 常用的文件操作函数

方法	功能说明
abspath(path)	返回给定路径的绝对路径
basename(path)	返回指定路径的最后一个组成部分
commonpath(paths)	返回给定的多个路径的最长公共路径
commonprefix(paths)	返回给定的多个路径的最长公共前缀
dirname(path)	返回给定路径的文件夹部分
exists(path)	判断文件是否存在
getatime(filename)	返回文件的最后访问时间
getctime(filename)	返回文件的创建时间
getmtime(filename)	返回文件的最后修改时间
getsize(filename)	返回文件的大小
isabs(path)	判断 path 是否为绝对路径
isdir(path)	判断 path 是否为文件夹
isfile(path)	判断 path 是否为文件
join(path, *paths)	连接两个或多个 path

realpath(path)	返回给定路径的绝对路径
relpath(path)	返回给定路径的相对路径，不能跨越磁盘驱动器或分区
samefile(f1, f2)	测试 f1 和 f2 这两个路径是否引用的同一个文件
split(path)	以路径中的最后一个斜线为分隔符把路径分隔成两部分，以元组形式返回
splitext(path)	从路径中分隔文件的扩展名
splitdrive(path)	从路径中分隔驱动器的名称

Linux 下获取文件名和文件路径使用示例的演示：

```
import os
print( os.path.basename('/root/陈福明.txt') )     # 返回文件名
print( os.path.dirname('/root/陈福明.txt') )      # 返回文件夹路径
```

Linux 下输出结果为：

```
陈福明.txt
/root
```

Linux 下文件路径拆分与合并的演示：

```
print( os.path.split('/root/陈福明.txt') )          # 分割文件名与路径
print( os.path.join('root','test','陈福明.txt') )   # 将文件夹和文件名合成一个路径
```

Linux 下输出结果为：

```
 ('/root', '陈福明.txt')
root/test/陈福明.txt
```

Window 下可以改写上面的完整的文件名和路径，然后运行查看输出结果。

文件各种时间属性的获取的演示：

```
os.path.getctime(r'D:\Pythontest\ostest\hello.py') #  文件的创建时间
os.path.getmtime(r'D:\Pythontest\ostest\hello.py')   #文件的最后修改时间
os.path.getatime(r'D:\Pythontest\ostest\hello.py')   #文件的最后访问时间
```

执行结果为：

```
1481687717.8506615
1481695651.857048
1481687717.8506615
```

查看一个文件的字节数的演示：

```
os.path.getsize(r'D:\Pythontest\ostest\hello.py')
```

执行结果为：

```
58
```

判断文件是否存在的演示：

```
os.path.exists(r'D:\Pythontest\ostest\hello.py')
os.path.exists(r'D:\Pythontest\ostest\hello1.py')
```

执行结果为：

```
True
False
```

11.3 文件数据处理

11.3.1 有规则的文本文件的数据处理

11.3.1.1 用分隔符分隔的字符文件数据处理

用固定分隔符分隔的**字符**文件数据，读取出来之后，可以直接用字符串的 split 方法，分割为列表。假如 strs.txt 文件中有 utf-8 编码的数据：陈福明,李晓丽,高兴,James,Tom，下例可以获得列表数据：

固定分隔符分隔的字符文件数据处理的演示：

```
with open("strs.txt","r",encoding="utf-8") as f:
    names = f.read()
    name_list = names.split(",")
```

```
print(name_list)
```

运行结果:

```
['陈福明', '李晓丽', '高兴', 'James', 'Tom']
```

获得列表数据后可以进一步数据处理。例如 csv 格式的文件就是常见的分隔符分隔的字符文件。

此外对于用空格、Tab 键、回车以及换行键分割的文件数据，从文件中读取出字符串之后，可以直接用字符串的不带参数的 split 方法，直接分割为列表。

11.3.1.2 用英文逗号分隔的数字文件数据处理

用英文逗号分隔的**数字**文件，读取出来之后，可以直接用字符串的 eval 函数，转换为元组。假如 numbers.txt 文件中有数字数据: 92,63,34,77,82，下例可以获得元组数据：

用英文逗号分隔的数字文件数据处理的演示：

```
with open("numbers.txt","r",encoding="utf-8") as f:
    numbers = f.read()
    number_tuple = eval(numbers)
    print(number_tuple)
```

运行结果:

```
(92, 63, 34, 77, 82)
```

获得元组数据后可以进一步数据处理。

11.3.2 高级数据的文件存取

11.3.2.1 使用 str 和 eval 函数

对于高级数据，保存为文件的时候，一种方法是直接用内置函数 str()把高级数据转换为字符串，然后写入文件；读取时，读取出来的数据，用 eval 函数转换为相应的数据类型即可。如下例：

使用 str 和 eval 函数通过文件存取高级数据的演示：

```
listdict = [
            2,
          {'姓名':"陈福明","性别":"男"},
```

```
            {'姓名':"李晓丽","性别":"女"}
        ]
def readListDict(filename):
        with open(filename,"r",encoding="utf-8") as f:
            listdict = eval(f.read())
            return listdict
def writeListDict(filename,listdict):
        with open(filename,"w",encoding="utf-8") as f:
            f.write(str(listdict))
filename = "listdict.txt"
writeListDict(filename,listdict)
listd = readListDict(filename)
print(listd)
```

输出结果:

```
[2, {'姓名': '陈福明', '性别': '男'}, {'姓名': '李晓丽', '性别': '女'}]
```

打开文件 listdict.txt，文件内容也是：[2, {'姓名': '陈福明', '性别': '男'}, {'姓名': '李晓丽', '性别': '女'}]

11.3.2.2 使用 pickle 模块

pickle 是 Python 语言的一个标准模块，安装 Python 后已包含 pickle 库，不再需要单独安装。

pickle 模块实现了基本的数据序列化和持久化、反序列化。通过 pickle 模块的序列化操作我们能够将程序中运行的对象(Python 中就是字符串、高级数据等)信息保存到文件中去，永久存储；通过 pickle 模块的反序列化操作，我们能够从文件中创建上一次程序保存的对象。

所谓序列化过程，就是将对象(Python 中就是字符串、高级数据等)信息转变为二进制数据流，更容易存储在硬盘文件之中；当需要从硬盘上读取文件时，可将其反序列化得到原始的数据。有时我们需要将程序中的一些字符串、列表、字典等数据，长久的保存下来，而不是简单的放入内存中，关机断电也不会丢失数据，方便以后使用。pickle 模块就派上用场了，它可以将对象转换为一种可以传输或存储的文件格式。

使用 pickle 存取高级数据的演示：

```
#保存到 db 文件(dump)：
li = [11,22,33]
pickle.dump(li,open('db','wb'))
#读取保存的列表(load)：
ret = pickle.load(open('db','rb'))
print(ret)
```

pickle 使用注意事项：

(1) pickle 只能在 Python 中使用，只支持 Python 的基本数据类型。

(2) 序列化的时候，pickle 只是序列化了整个序列对象，而不是内存地址。

11.3.2.3 使用 json 模块

json 是 Python 语言的一个标准模块，安装 Python 后已包含 json 库，不再需要单独安装。

Python 允许使用称为 json（JavaScript Object Notation）的数据交换格式，用户不用对复杂的数据类型进行特殊转换后保存到文件中。json 可以将数据层次结构化，并将它们转换为字符串表示形式，这个过程叫做序列化；从字符串表示重建数据称为反序列化。在序列化和反序列化之间，表示对象的字符串已存储在文件中，也可以通过网络连接可以发送到远程服务器。在 Python 中，列表，字典可以用 json 序列化。

json 数据格式的使用(dumps 和 loads)的演示：

```
import json
data = {"spam" : "foo", "parrot" : 42}
in_json = json.dumps(data) #编码
in_json
```

运行结果：

```
'{"parrot": 42, "spam": "foo"}' #字符串形式
```

接前面的程序：

```
data=json.loads(in_json)        # 解码成一个对象
data
```

运行结果：

```
{"spam" : "foo", "parrot" : 42} #对象形式
```

json 数据文件的使用(dump 和 load) 的演示：

```
import json
data = {"name" : "Python", "count" : 46, "mark": [["A ", 7], ["B ", 16], ["C ", 14] , ["D ", 6] ,
["E ", 3]] }
#保存数据到文件 record.json
with open("record.json","w") as dump_f:
    json.dump(data,dump_f)
#从文件 record.json 读取数据
with open("record.json",'r') as load_f:
    load_ data = json.load(load_f)
    print(load_ data)
load_ data ['pass'] = 43
print(load_ data)
```

运行结果：

```
{'name': 'Python', 'count': 46, 'mark': [['A ', 7], ['B ', 16], ['C ', 14], ['D ', 6], ['E ', 3]]}
{'name': 'Python', 'count': 46, 'mark': [['A ', 7], ['B ', 16], ['C ', 14], ['D ', 6], ['E ', 3]], 'pass': 43}
```

11.3.3 其他类型文件的数据处理

其他类型文件的数据，基本上都需要安装三方模块进行数据存取和处理，比如 Excel 文件可以使用三方模块 xlrd 和 xlwt 或者 openpyxl 进行存取，也可以用 pandas 存取和数据处理。其中值得推荐的是 pandas 可以对多种文件的数据进行存取，而且数据处理能力强大。有兴趣的同学可以查找相关资料学习。

习题

一、填空题

1、对文件进行写入操作之后，_____方法用来在不关闭文件对象的情况下将缓冲区内容写入文件。

2、Python 内置函数_____用来打开或创建文件并返回文件对象。

3、使用上下文管理关键字_____可以自动管理文件对象，不论何种原因结束该关键字中的语句块，都能保证文件被正确关闭。

3、Python 标准库 os 中用来列出指定文件夹中的文件和子文件夹列表的方式是_____。

4、Python 标准库 os.path 中用来判断指定文件是否存在的方法是_____。

5、Python 标准库 os.path 中用来判断指定路径是否为文件的方法是_____。

6、Python 标准库 os.path 中用来判断指定路径是否为文件夹的方法是_____。

7、Python 标准库 os.path 中用来分割指定路径中的文件扩展名的方法是_____。

二、判断题

1、扩展库 os 中的方法 remove()可以删除带有只读属性的文件。（ ）

2、使用内置函数 open()且以”w”模式打开的文件，文件指针默认指向文件尾。（ ）

3、使用内置函数 open()打开文件时，只要文件路径正确就总是可以正确打开的。（ ）

4、假设 os 模块已导入，那么列表推导式[filename for filename in os.listdir('C:\\Windows') if filename.endswith('.exe')]的作用是列出 C:\Windows 文件夹中所有扩展名为.exe 的文件。（ ）

5、二进制文件不能使用记事本程序打开。（ ）

6、使用普通文本编辑器软件也可以正常查看二进制文件的内容。（ ）

7、二进制文件也可以使用记事本或其他文本编辑器打开，但是一般来说无法正常查看其中的内容。（ ）

8、Python 标准库 os 中的方法 isfile()可以用来测试给定的路径是否为文件。（ ）

9、Python 标准库 os 中的方法 exists()可以用来测试给定路径的文件是否存在。（ ）

10、Python 标准库 os 中的方法 isdir()可以用来测试给定的路径是否为文件夹。（ ）

11、Python 标准库 os 中的方法 listdir()返回包含指定路径中所有文件和文件夹名称的列表。（ ）

12、标准库 os 的 rename()方法可以实现文件移动操作。（ ）

13、标准库 os 的 listdir()方法默认只能列出指定文件夹中当前层级的文件和文件夹列表，而不能列出其子文件夹中的文件。（ ）

14、文件对象是可以迭代的。（ ）

15、文件对象的 tell()方法用来返回文件指针的当前位置。（ ）

16、以写模式打开的文件无法进读操作。（ ）

17、假设已成功导入 os 和 sys 标准库，那么表达式 os.path.dirname(sys.executable) 的值为 Python 安装文件夹。（ ）

三、程序题

1、编写程序，在 D 盘根文件夹下创建一个文本文件 test.txt，并向其中写入字符串 hello world。

答：

```
fp = open(r'D:\test.txt', 'a+')
print('hello world', file=fp)
fp.close()
```

2、一个 scores.txt 文件中存放了成绩列表，形如:[100,90,21,49,61,80,71,42,66…]，请统计优(大于等于 90)，良(80 到 90) ，中(70 到 80) ，及格(60 到 70)和不及格(小于 60)的人数到一个字典中，如：{'优':5, '良':35, '中':15, '及格':6, '不及格':5}，并打印字典结果。

提示：列表字符串转列表可用 eval 函数。

3、把一个数字的 list 从小到大排序，然后写入文件，然后从文件中读取出来文件内容，然后反序，在追加到文件的下一行。

答：

```
import random
list_num = [random.randint(1, 100) for i in range(1, 20)]
list_num.sort()
txt = ','.join(list(map(str, list_num)))
with open('num.txt', 'w', encoding='utf-8') as f:
    f.write(txt)
with open('num.txt', 'r+', encoding='utf-8') as f:
```

```
    list_num2 = f.read().split(',')
    list_num2.reverse()
    txt2 = ','.join(list(map(str, list_num2)))
    f.write('\n')
    f.write(txt2)
```


第 12 章 日期、时间和 Turtle 库

知识点

(1) time 库

(2) datetime 库

(3) calendar 库

(4) turtle 库

12.1 日期和时间简介

软件开发过程中转换日期和时间格式是一个常见的功能，Python 程序有很多种方式处理日期和时间。Python 的内置库提供了 time 、datetime 和 calendar 模块用于格式化日期和时间，时间间隔是以秒为单位的浮点小数。和大多数语言一样，Python 中的每个时间戳都以从公元 1970 年 1 月 1 日午夜零时、零分、零秒经过了多长时间来表示。时间戳单位适于做日期运算。但是 1970 年之前的日期就无法以此表示了。太遥远的日期也不行，UNIX 和 Windows 只支持到 2038 年。

12.2 time 模块

time 模块中时间表现的格式主要有三种：

(1) 时间戳(timestamp)，时间戳表示的是从 1970 年 1 月 1 日 00:00:00 开始按秒计算的偏移量

(2) 时间元组(struct_time)，共有九个元素。

(3) 格式化时间(format time)，已格式化的结构使时间更具可读性。包括自定义格式和固定格式。

这三种格式转换图如下：

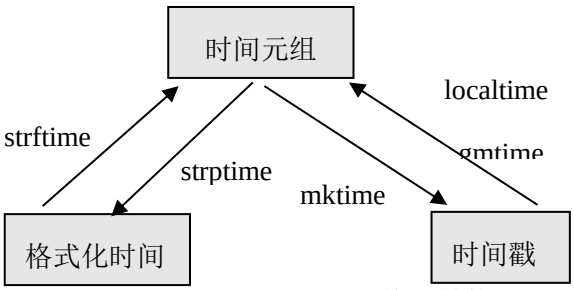


图 15-1 时间格式转换图

举例如下：

(1) 获取以秒为单位的浮点时间 time ():

```
import time
print (time.time()) #获取当前时间戳的值，单位为秒
```

运行结果:

```
1576764132.213746
```

(2) 使用 ctime()函数获取可以直观理解的当前时间:

```
import time
print ( time.ctime() )
```

运行结果:

```
Thu Dec 19 22:03:18 2019
```

(3) 将已有时间戳转化为直观时间:

```
import time
t = time.time () #时间戳
print (t)
print ( time.ctime(t)) #时间戳转化为直观时间
```

运行结果:

```
1576764198.733124
Thu Dec 19 22:03:18 2019
```

(4) 时间元组:

表 12-1 时间元组格式

字段	属性	值
年（4 位数字）	tm_year	2017
月	tm_mon	1 到 12
日	tm_mday	1 到 31
小时	tm_hour	0 到 23

分钟	tm_min	0 到 59
秒	tm_sec	0 到 61（60 或 61 是润秒）
一周的第几日	tm_wday	0 到 6 (0 是周一)
一年的第几日	tm_yday	1 到 366，一年中的第几天
夏令时	tm_isdst	是否为夏令时（值 1：夏令时，值 0：不是夏令时，默认为 0）

时间元组是用 9 个数字封装起来的一个元组，表示固定格式的时间（日期），如表 15-1。可以用 localtime()打印时间元祖，例子如下：

```
print (time.localtime (time.time() ) )
```

运行结果:

```
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=22, tm_min=4,
tm_sec=53, tm_wday=3, tm_yday=353, tm_isdst=0)
```

(5) 获取格林尼治时间 UTC（Coordinated Universal Time）:

```
print( time.gmtime()) #获取 UTC 格式的当前时间(时间元组)
```

运行结果:

```
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=22, tm_min=4,
tm_sec=53, tm_wday=3, tm_yday=353, tm_isdst=0)
```

一个 UTC 时间有 9 项，也用元组时间格式表示。

(6) 时区时间和时间戳的转换:

通过 mktime()函数可以实现时区时间(时间元组)转换为时间戳，如下：

```
gmt = time.gmtime()          #UTC 格式的时间(时间元组)
print (gmt)
ftime1=time.mktime(gmt)
print (ftime1)               #将 UTC 格式的时间(时间元组)转化为时间戳
lt = time.localtime()        #所在时区当前时间(时间元组)
print (gmt)
ftime2=time.mktime(lt)
print (ftime2)               #将所在时区当前时间(时间元组)转化为时间戳
```

运行结果:

```
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=14, tm_min=19,
tm_sec=15, tm_wday=3, tm_yday=353, tm_isdst=0)
1576736355.0
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=14, tm_min=19,
tm_sec=15, tm_wday=3, tm_yday=353, tm_isdst=0)
1576765155.0
```

(7) 时间戳转化为 UTC 格式时间:

```
t = time.time()                 #时间戳
print (time.gmtime(t))          #将时间戳转化为 UTC 格式的时间
print (time.localtime(t))       #将时间戳转化为当前时区时间
```

运行结果:

```
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=14, tm_min=20,
tm_sec=28, tm_wday=3, tm_yday=353, tm_isdst=0)
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=22, tm_min=20,
tm_sec=28, tm_wday=3, tm_yday=353, tm_isdst=0)
```

分析结果发现 tm_hour=14 对应 tm_hour=22，即 UTM 时间比当前时区时间晚 8 个小时。。

(8) 格式化时间：

```
lt = time.gmtime()                                  #UTC 格式当前时区时间
st = time.strftime("%b %d %Y %H:%M:%S", lt)
print(st)
```

运行结果:

```
Dec 19 2019 14:22:32
```

表 12-2 基本的时间格式

标识	含义	举例
%b	月份简写	Mar
%B	本地完整月份名称	3 月
%d	一个月的第几天，取值范围： [01,31].	20

%y	去掉世纪的年份（00 - 99）	13
%Y	完整的年份	2013
%H	24 小时制的小时，取值范围[00,23].	17
%M	分钟，取值范围 [00,59].	50
%S	秒，取值范围 [00,61].	30

12.3 datatime 模块

datatime 模块重新封装了 time 模块，提供更多接口，提供的类有：date, time, datetime, timedelta 等。date 类是日期对象,常用的属性有 year, month, day；time 类是时间对象；datetime 类是日期时间对象,常用的属性有 hour, minute, second, microsecond；timedelta 类是时间间隔，即两个时间点之间的长度。

1. datetime 模块的一些基本操作

(1) 打印当前的年月日：

```
import datetime
# 打印当前，年，月，日
print(datetime.date.today() )
```

运行结果:

```
2019-12-19
```

(2) 打印当前时间，精确到微秒：

```
import datetime
#打印当前时间，精确到微秒
current_time = datetime.datetime.now()
print(current_time)
```

运行结果:

```
2019-12-19 22:55:05.199632
```

(3) 时间的计算：

通过 timedelta ( ) 调整当前日期和时间，具体如下：

```
import datetime
```

```
#加十天
print (datetime.datetime.now() +datetime.timedelta (days=10) )
#减十天
print(datetime.datetime.now() +datetime.timedelta (days=-10))
#减十个小时
print(datetime.datetime.now() +datetime.timedelta (hours=-10))
#加 120s
print(datetime.datetime.now() +datetime.timedelta (seconds=120))
```

运行结果:

```
2019-12-29 22:57:45.214486
2019-12-09 22:57:45.230114
2019-12-19 12:57:45.230114
2019-12-19 22:59:45.245734
```

(4) 打印当天开始和结束时间(00:00:00 23:59:59):

```
>>> datetime.datetime.combine(datetime.date.today(), datetime.time.min)
datetime.datetime(2019, 12, 19, 0, 0)
>>> datetime.datetime.combine(datetime.date.today(), datetime.time.max)
datetime.datetime(2019, 12, 19, 23, 59, 59, 999999)
```

(5) 打印两个 datetime 的时间差:

```
>>> (datetime.datetime(2019,12,23,12,0,0) - datetime.datetime.now()).total_seconds()
305827.552091
```

2. datetime 模块用于时间格式转换

datetime 模块中 datetime 对象、格式化时间(string time)、时间戳(timestamp)和时间元组(tumetuple)可以互相转换。

(1) datetime 对象转为格式化时间(string time):

```
>>> import datetime
>>> datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
'2019-12-19 23:11:18' 305827.552091
```

(2) 格式化时间转为 datetime 对象:

```
>>> import datetime
>>> datetime.datetime.strptime("2019-12-31 18:20:10", "%Y-%m-%d %H:%M:%S")
datetime.datetime(2019, 12, 31, 18, 20, 10)
```

(3) datetime 对象转为时间元组(timetuple)：

```
>>> import datetime
>>> datetime.datetime.now().timetuple()
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=19, tm_hour=23, tm_min=13,
tm_sec=52, tm_wday=3, tm_yday=353, tm_isdst=-1)
```

(4) datetime 对象转为 date 对象：

```
>>> import datetime
>>> datetime.datetime.now().date()
datetime.date(2019, 12, 19)
```

(5) date 对象转为 datetime 对象：

```
>>> import datetime
>>> datetime.date.today()
datetime.date(2019, 12, 19)
>>> today = datetime.date.today()
>>> datetime.datetime.combine(today, datetime.time())
datetime.datetime(2019, 12, 19, 0, 0)
>>> datetime.datetime.combine(today, datetime.time.min)
datetime.datetime(2019, 12, 19, 0, 0)
```

(6) datetime 对象转为时间戳：

```
>>> import datetime,time
>>> now = datetime.datetime.now()
>>> timestamp = time.mktime(now.timetuple())
>>> timestamp
1576769321.0
```

(7) 时间戳转为 datetime 对象：

```
>>> import datetime
```

```
>>> datetime.datetime.fromtimestamp(1576769321.0)
datetime.datetime(2019, 12, 19, 23, 28, 41)
```

12.4 calendar 模块

calendar 模块主要用来处理年历和月历等，有很广泛的应用。

(1) 打印一年的日历：

```
import calendar
#打印每月日期
c = calendar.calendar(2021)
print(c)
```

运行结果:

```
                                  2021

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2  3       1  2  3  4  5  6  7       1  2  3  4  5  6  7
 4  5  6  7  8  9 10       8  9 10 11 12 13 14       8  9 10 11 12 13 14
11 12 13 14 15 16 17      15 16 17 18 19 20 21      15 16 17 18 19 20 21
18 19 20 21 22 23 24      22 23 24 25 26 27 28      22 23 24 25 26 27 28
25 26 27 28 29 30 31                                29 30 31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
          1  2  3  4                   1  2          1  2  3  4  5  6
 5  6  7  8  9 10 11       3  4  5  6  7  8  9       7  8  9 10 11 12 13
12 13 14 15 16 17 18      10 11 12 13 14 15 16      14 15 16 17 18 19 20
19 20 21 22 23 24 25      17 18 19 20 21 22 23      21 22 23 24 25 26 27
26 27 28 29 30            24 25 26 27 28 29 30      28 29 30
                          31
.......
```

(2) 打印一个月的日历：

```
cm=calendar.month(2021,9)
print(cm)
```

运行结果:

```
    September 2021
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

3) 根据指定的年月日计算星期几：

```
import calendar
cm=calendar.month(2021,9)
print(cm)
```

运行结果:

```
0
```

(4) 将时间元组转化为时间戳：

```
import calendar
tps = (2021,9,10,11,35,0,0,0)
result = calendar.timegm(tps)
print(result)
```

运行结果:

```
1631273700
```

(5) 将时间元组转化为时间戳：

```
import calendar
tps = (2021,9,10,11,35,0,0,0)
stamp= calendar.timegm(tps) #和 time.gmtime 功能相反
print(stamp)
```

运行结果:

```
1631273700
```

12.5 turtle 库

turtle（海龟）是 Python 重要的标准库之一，它能够进行基本的图形绘制。turtle 图形绘制的概念诞生于 1969 年，成功应用于 LOGO 编程语言。

turtle 库绘制图形有一个基本框架：一个小海龟在坐标系中爬行，其爬行轨迹形成了绘制图形。刚开始绘制时，小海龟位于画布正中央，此处坐标为（0,0），前进方向为水平右方。

12.5.1 turtle 库的导入

turtle 库中的函数不能直接使用，需要使用 import 导入该库：

```
import turtle as tt
```

或者：

```
from turtle import *
```

turtle 库包含 100 多个功能函数，主要包括画布函数、画笔状态函数和画笔运动函数 3 类。

12.5.2 turtle 画布

画布就是 turtle 展开用于绘图的区域，大家可以设置它的大小和初始位置。

设置画布大小函数 screensize：

screensize(canvwidth=None, canvheight=None, bg=None)

参数分别为画布的宽(单位像素)，高，背景颜色。如：

```
import turtle as tt
tt.screensize(800,600, "green")  #画布的宽(单位像素)，高，背景颜色
```

再如：

```
tt.screensize() #返回默认大小(400, 300)
```

设置画布大小函数 setup：

setup(width=0.5, height=0.75, startx=None, starty=None)，

参数：width, height: 输入宽和高为整数时，表示像素；为小数时，表示占据电脑屏幕的比例；(startx, starty): 这一坐标表示矩形窗口左上角顶点的位置，如果为空,则窗口位于屏幕中心。如：

```
import turtle as tt
tt.setup(width=0.6,height=0.6)
```

再如：

```
tt.setup(width=800,height=800, startx=100, starty=100)
```

12.5.3 turtle 画笔的状态与属性

1. 画笔的状态

在画布上，默认有一个坐标原点为画布中心的坐标轴，坐标原点上有一只面朝 x 轴正方向小乌龟。这里描述小乌龟时使用了两个词语：坐标原点(位置)，面朝 x 轴正方向(方向)， turtle 绘图中，就是使用位置方向描述小乌龟(画笔)的状态。

2. 画笔的属性

画笔（画笔的属性，颜色、画线的宽度等）

(1) pensize()：设置画笔的宽度；

(2) pencolor()：没有参数传入，返回当前画笔颜色，传入参数设置画笔颜色，可以是字符串如"green", "red",也可以是红绿蓝 RGB 3 元组。

(3) speed(speed)：设置画笔移动速度，画笔绘制的速度范围[0,10]整数，数字越大越快。

12.5.4 turtle 绘图

操纵海龟绘图有着许多的命令，这些命令可以划分为 3 种：一种为画笔运动命令，一种为画笔控制命令，还有一种是全局控制命令。

1. 画笔运动命令

表 12-3 画笔运动命令

命令	说明
forward(distance)	向当前画笔方向移动 distance 像素长度，简写为 fd()

backward(distance)	向当前画笔相反方向移动 distance 像素长度
right(degree)	顺时针移动 degree°
left(degree)	逆时针移动 degree°
pendown()	移动时绘制图形，缺省时也为绘制
goto(x,y)	将画笔移动到坐标为 x,y 的位置
penup()	提起笔移动，不绘制图形，用于另起一个地方绘制
circle()	画圆，半径为正(负)，表示圆心在画笔的左边(右边)画圆
setx( )	将当前 x 轴移动到指定位置
sety( )	将当前 y 轴移动到指定位置
setheading(angle)	设置当前朝向为 angle 角度
home()	设置当前画笔位置为原点，朝向东。
dot(r)	绘制一个指定直径和颜色的圆点

2. 画笔控制命令

表 12-4 画笔控制命令

命令	说明
fillcolor(colorstring)	绘制图形的填充颜色
color(color1, color2)	同时设置 pencolor=color1，fillcolor=color2
filling()	返回当前是否在填充状态
begin_fill()	准备开始填充图形
end_fill()	填充完成
hideturtle()	隐藏画笔的 turtle 形状
showturtle()	显示画笔的 turtle 形状

3. 全局控制命令

表 12-5 全局控制命令

命令	说明
clear()	清空 turtle 窗口，但是 turtle 的位置和状态不会改变

reset()	清空窗口，重置 turtle 状态为起始状态
undo()	撤销上一个 turtle 动作
isvisible()	返回当前 turtle 是否可见
stamp()	复制当前图形
write(s, move=False, align='left', font=('Arial', 8, 'normal'))	写文本，s 为文本内容，align 为对齐方式，font 是字体元组，分别为字体名称，大小和类型

4. 其他命令

表 12-6 其他命令

命令	说明
mainloop()或 done()	启动事件循环 -调用 Tkinter 的 mainloop 函数。 必须是乌龟图形程序中的最后一个语句。
mode(mode=None)	设置乌龟模式（“standard”，“logo”或“world”）并执行重置。如果没有给出模式，则返回当前模式。 standard：向右（东），逆时针 logo：向上（北）顺时针
delay(delay=None)	设置或返回以毫秒为单位的绘图延迟。
begin_poly()	开始记录多边形的顶点。当前的乌龟位置是多边形的第一个顶点。
end_poly()	停止记录多边形的顶点。当前的乌龟位置是多边形的最后一个顶点。将与第一个顶点相连。
get_poly()	返回最后记录的多边形。

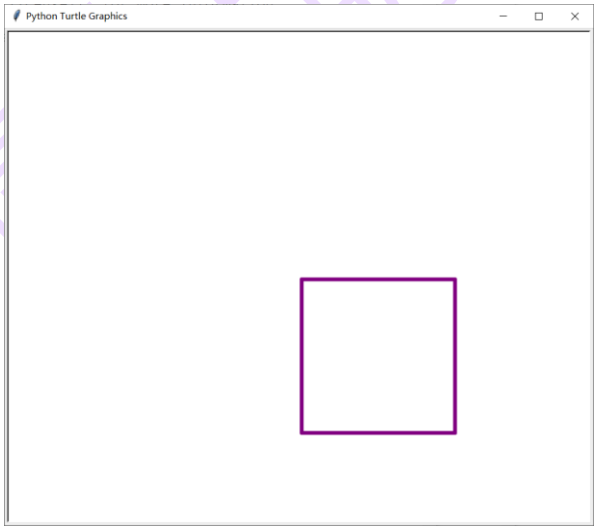
12.5.5 turtle 绘图举例

1. 绘制正方形

```
#绘制正方形
import turtle as tt
#定义绘制时画笔的颜色
tt.color("purple")
```

```
#定义绘制时画笔的线条的宽度
tt.pensize(5)
#定义绘图的速度
tt.speed(2)
#以 0,0 为起点进行绘制
tt.goto(0,0)
#绘出正方形的四条边
for i in range(4):
    tt.forward(200)
    tt.right(90)
#隐藏画笔
tt.hideturtle()
#结束绘图
tt.done()
```

运行结果：



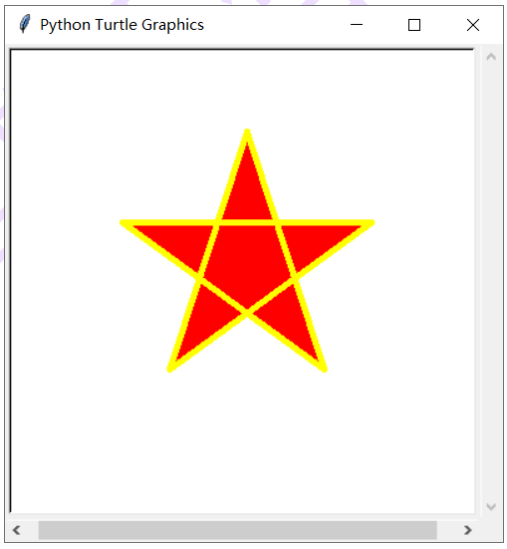
2.绘制五角星

下面是绘制五角星的代码，大家试着解释每一代码块的含义。

```
# 绘制五角星
```

```
import turtle as tt
tt.setup(400,400)
tt.penup()
tt.goto(-100,50)
tt.pendown()
tt.color("red")
tt.begin_fill()
for i in range(5):
    tt.forward(200)
    tt.right(144)
tt.end_fill()
tt.hideturtle()
tt.done()
```

运行结果：

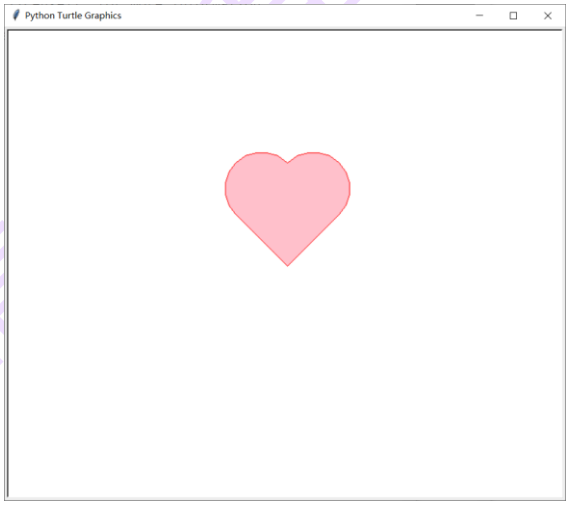


3. 绘制心形

```
# 绘制心形
import turtle as tt
tt.color('red','pink')
```

```
tt.begin_fill()
tt.left(135)
tt.fd(100)
tt.right(180)
tt.circle(50,-180)
tt.left(90)
tt.circle(50,-180)
tt.right(180)
tt.fd(100)
tt.end_fill()
tt.hideturtle()
tt.done()
```

运行结果：



4. 绘制时钟

```
#绘制时钟
# coding=utf-8
import turtle as tt
from datetime import *
```

```
# 当前日期属于一周的第几天
def Week(t):
    week = ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期日"]
    return week[t.weekday()]

# 获取当前时间
def Date(t):
    y = t.year
    m = t.month
    d = t.day
    cur_hour = t.hour;
    cur_min = t.minute;
    cur_sec = t.second;
    return "%s-%d-%d %d:%02d:%02d" % (y, m, d, cur_hour, cur_min, cur_sec)

# 移动画笔,距离为 distance
def movePen(distance):
    tt.penup()
    tt.pensize(5)
    tt.pencolor("blue")
    tt.fd(distance)
    tt.pendown()

# 绘制表针
def makeHands(name, length):
    # 清空窗口，重置 turtule 状态为初始状态
```

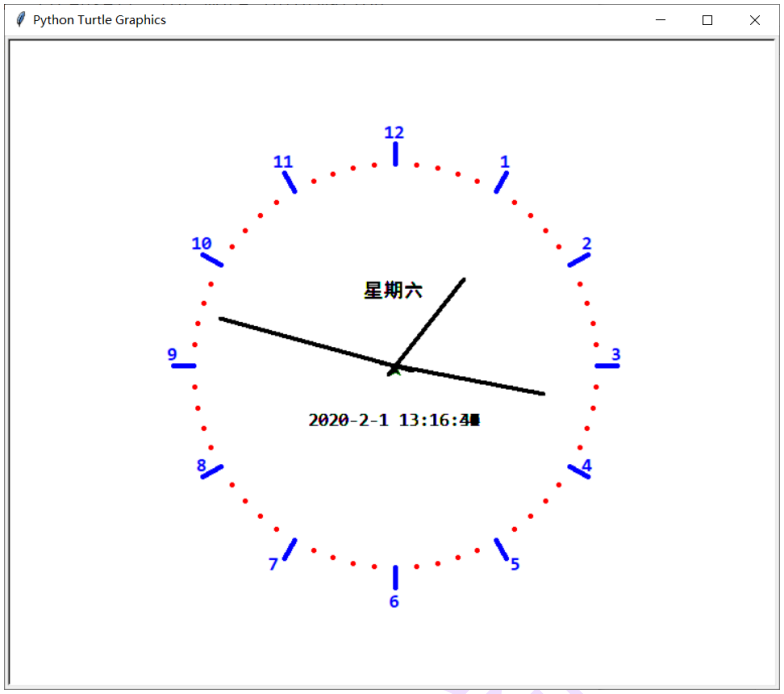
```
    tt.reset()
    movePen(-length * 0.1)
    # 开始记录多边形的顶点
    tt.begin_poly()
    tt.fd(length * 1.1)
    # 停止记录多边形的顶点
    tt.end_poly()
    # 返回记录的多边形
    handForm = tt.get_poly()
    tt.register_shape(name, handForm)

# 初始化
def initial():
    global secHand, minHand, hurHand, printer
    # 重置方向向北(上)，正角度为顺时针
    tt.mode("logo")
    # 建立并初始化表针
    makeHands("secHand", 180)
    makeHands("minHand", 150)
    makeHands("hurHand", 110)
    secHand = tt.Turtle()
    secHand.shape("secHand")
    minHand = tt.Turtle()
    minHand.shape("minHand")
    hurHand = tt.Turtle()
    hurHand.shape("hurHand")
```

```
    for hand in secHand, minHand, hurHand:
        hand.shapesize(1, 1, 4)
        hand.speed(0)

    # 输出文字
    printer = tt.Turtle()
    # 隐藏画笔
    printer.hideturtle()
    printer.penup()

# 绘制表盘外框
def drawClock(R):
    # 清空窗口，重置 turtule 状态为初始状态
    tt.reset()
    # 画笔尺寸
    tt.pensize(5)
    for i in range(60):
        movePen(R)
        if i % 5 == 0:
            tt.fd(20)
            movePen(-R - 20)

            movePen(R + 20)
            if i == 0:
                # 写文本
                tt.write(int(12), align="center", font=("Consolas", 14, "bold"))
            elif i == 30:
```

```
            movePen(25)
            tt.write(int(i / 5), align="center", font=("Consolas", 14, "bold"))
            movePen(-25)
        elif (i == 25 or i == 35):
            movePen(20)
            tt.write(int(i / 5), align="center", font=("Consolas", 14, "bold"))
            movePen(-20)
        else:
            tt.write(int(i / 5), align="center", font=("Consolas", 14, "bold"))
        movePen(-R - 20)
    else:
        # 绘制指定半径和颜色的点
        tt.dot(5, "red")
        movePen(-R)
    tt.right(6)


# 表针的动态显示
def handsMove():
    t = datetime.today()
    second = t.second + t.microsecond * 0.000001
    minute = t.minute + second / 60.0
    hour = t.hour + minute / 60.0
    secHand.seth(6 * second)
    minHand.seth(6 * minute)
    hurHand.seth(30 * hour)

    tt.tracer(False)
```

```
    printer.fd(65)
    tt.pencolor("green")
    printer.write(Week(t), align="center", font = ("黑体", 14))
    printer.back(130)
    printer.write(Date(t), align="center", font = ("Consolas", 14))
    # 设置当前画笔位置为原点，方向朝东
    printer.home()
    tt.tracer(True)

    # 经过 100ms 后继续调用 handsMove 函数
    tt.ontimer(handsMove, 100)

# 调用定义的函数，打开和关闭动画，为更新图纸设置延迟；
tt.tracer(False)
initial()
drawClock(200)
tt.tracer(True)
handsMove()
tt.mainloop()
```

运行结果：



习题

一、简答题

1、Python 常用的时间、日期库有哪些函数？

2、turtle 库常用函数有哪些？

2、time.localtime 和 time.gmtime 两个函数，获取的时间有什么区别？

3、如何将一个 time.localtime 时间转化为数值型时间？反过来如何将数值型时间转化为时区时间？

4、计算当前时间 20 天前的日期时间。

5、打印 2023 年 3 月份的日历。

6、使用 turtle 库分别画出圆形、环形、菱形、梯形、椭圆形等基本形状并填充颜色。

第 13 章 Pyinstaller 与三方库简介

知识点

(1) Pyinstaller

(2) Jieba 库

(3) wordcloud 库

(4) 其他三方库简介

13.1 Pyinstaller

目前 python 打包工具有多个，目前比较好用的为 pyinstaller，该工具可以支持在 window 和 linux 下使用。

在 windows 下，大小为几十 K 的源文件打包成 exe 文件，可能为几十兆，原因是把程序所引用的库文件也一起打包在一起。

1. 安装 Pyinstaller

Python 默认并不包含 PyInstaller 模块，因此需要自行安装 PyInstaller 模块。安装 PyInstaller 模块与安装其他 Python 模块一样，使用 pip 命令安装即可。在命令行输入如下命令：

```
pip install pyinstaller
```

在 PyInstaller 模块安装成功之后，在 Python 的安装目录下的 Scripts (D:\Python\Python36\Scripts) 目录下会增加一个 pyinstaller.exe 程序，接下来就可以使用该工具将 Python 程序生成 EXE 程序了。

2. 使用 Pyinstaller

PyInstaller 工具的命令语法如下：

```
pyinstaller 选项 Python 源文件
```

不管这个 Python 应用是单文件的应用，还是多文件的应用，只要在使用 pyinstaller 命令时编译作为程序入口的 Python 程序即可。

PyInstaller 工具是跨平台的，它既可以在 Windows 平台上使用，也可以在 Linux 平台上运行。在不同的平台上使用 PyInstaller 工具的方法是一样的，它们支持的选项也是一样的。

下面先创建一个 app_one 文件夹，在该文件夹下创建 main.py 文件，文件中分别包含如下代码：

```
def helloworld(str):
    print('你好，'+str)

def main():
    print('开始执行程序')
    print(helloworld('Python 学习网'))
#  增加调用 main()函数
if __name__ == '__main__':
    main()
```

接下来使用命令行工具进入到此 app_one 目录下，执行如下命令：

```
D:\app_one>pyinstaller -F main.py
```

上面的 D:\app_one>是 Windows 的 cmd 命令行窗口中的提示符。执行上面命令，将看到详细的生成过程。当生成完成后，将会在此 app 目录下看到多了一个 dist 目录，并在该目录下看到有一个 main.exe 文件，这就是使用 PyInstaller 工具生成的 EXE 程序。

在 Windows 的 cmd 命令行窗口中进入 dist 目录下，在该目录执行 main.exe ，将会看到该程序生成如下输出结果：

```
D:\app_one\dist>main.exe
```

上面的 D:\app_one\dist >是 Windows 的 cmd 命令行窗口中的提示符。运行结果如下：

```
开始执行程序
你好，Python 学习网
```

由于该程序没有图形用户界面，因此如果大家要是通过双击来运行该程序，就只能看到程序窗口一闪就消失了，这是没法看到这个程序的输出结果的。

在上面命令中使用了-F 选项，该选项指定生成单独的 exe 文件，因此，在 dist 目录下生成了一个单独的大约为 6MB 的 main.exe 文件（在 Linux 平台上生成的文件就叫 main，没有后缀）；与 -F 选项对应的是 -D 选项（默认选项），该选项指定生成一个目录（包含多个文件）来作为程序。

现实中，一个项目往往有多个文件，可以把上面的简单例子改成两个文件的项目，即另外创建一个 app_tow 文件夹，在该文件夹下创建一个 helloworld .py 文件和 main.py 文件，把前面的 main.py 文件拆分成 helloworld .py 文件和 main.py 文件，拆分结果如下：

(1) helloworld.py

```
def helloworld(str):
    print(str)
```

(2) main.py

```
from helloworld import *

def main():
    print('程序开始执行')
    print(helloworld('Python 学习网'))
#  增加调用 main()函数
if __name__ == '__main__':
    main()
```

对于项目有多个文件，打包时可以把多个文件列出来，中间用空格隔开即可。即执行如下命令：

```
D:\app_tow\dist>main.exe
```

上面的 D:\ app_tow\dist>是 Windows 的 cmd 命令行窗口中的提示符。

此外，如果需要修改默认图标为指定图标，则使用“-i 图标文件名”来实现。

3. Pyinstaller 的选项参数

表 13-1Pyinstaler 的参数

参数	含义

-F, –onefile	打包单个文件，如果代码都写在一个.py 文件的话，可以用这个，如果是多个.py 文件就别用
-D, –onedir	打包多个文件，在 dist 中生成很多依赖文件，适合以框架形式编写工具代码
-K, –tk	在部署时包含 TCL/TK
-a, –ascii	不包含编码。在支持 Unicode 的 python 版本上默认包含所有的编码
-d, –debug	产生 debug 版本的可执行文件
-w,–windowed,–noconsole	使用 Windows 子系统执行。当程序启动的时候不会打开命令行(只对 Windows 有效)
-c,–nowindowed,–console	使用控制台子系统执行(默认)(只对 Windows 有效) pyinstaller -c xxxx.py pyinstaller xxxx.py --console
-s,–strip	可执行文件和共享库将 run through strip。注意 Cygwin 的 strip 往往使普通的 win32 Dll 无法使用
-X, –upx	如果有 UPX 安装(执行 Configure.py 时检测)，会压缩执行文件(Windows 系统中的 DLL 也会)
-o DIR, –out=DIR	指定 spec 文件的生成目录,如果没有指定,而且当前目录是 PyInstaller 的根目录,会自动创建一个用于输出(spec 和生成的可执行文件)的目录.如果没有指定,而当前目录不是 PyInstaller 的根目录,则会输出到当前的目录下
-p DIR, –path=DIR	设置导入路径(和使用 PYTHONPATH 效果相似).可以用路径分割符(Windows 使用分号，Linux 使用冒号)分割，指定多个目录.也可以使用多个-p 参数来设置多个导入路径，让 Pyinstaller 自己去找程序需要的资源
–icon=<FILE.ICO>	将 file.ico 添加为可执行文件的资源(只对 Windows 系统有效)，改变程序的图标 pyinstaller -i ico 路径 xxxxx.py
–icon=<FILE.EXE,N>	将 file.exe 的第 n 个图标添加为可执行文件的资源(只对 Windows 系统有效)
-v FILE, –version=FILE	将 verfile 作为可执行文件的版本资源(只对 Windows 系统有效)
-n NAME, –name=NAME	可选的项目(产生的 spec 的)名字.如果省略,第一个脚本的主文件名将作为 spec 的名字

13.2 Jieba

Jieba 是优秀的中文分词第三方库，中文文本需要通过分词获得单个词语。

1. 安装 Jieba

Python 默认并不包含 Jieba 模块，因此需要自行安装 Jieba 模块。安装 jieba 模块与安装其他 Python 模块一样，使用 pip 命令安装即可。在命令行输入如下命令：

```
pip install jieba
```

Jieba 库提供三种分词模式，最简单只需要掌握一个函数。

2. Jieba 库

Jieba 分词依靠中文词库，利用一个中文词库，确定汉字之间的关联概率。汉字间概率大的组成词组，形成分词结果。除了分词，用户还可以添加自定义的词组。

Jieba 库，分为精确模式、全模式、搜索引擎模式。

(1) 精确模式：把文本精确的且分开，不存在冗余单词

(2) 全模式 ：把文本中所有可能的词语都扫描出来，有冗余

(3) 搜索引擎模式：在精确模式基础上，对长词再次切分

Jieba 库常用的分词函数为：

(1) jieba.cut(s)　　精准模式，返回一个可迭代的数据类型

(2) jieba.cut(s,cut_all=True)　　全模式，输出文本 s 中所有可能的单词

(3) jieba.cut_for_search(s)　搜索引擎模式，适合搜索引擎建立索引的分词结果

(4) jieba.lcut(s)　　精准模式，返回一个列表类型，建议使用

(5) jieba.lcut(s,cut_all=True)　　全模式，返回一个列表类型，建议使用

(6) jieba.cut_for_search(s)　搜索引擎模式，返回一个列表类型，建议使用

(7) jieba.add_word(w)　向分词词典中增加新词 w

从上面的介绍可以看出，cut 函数和 lcut 函数的唯一区别，就是 cut 函数生成可迭代的数据类型。因此如果分词的字符串太长，建议使用 cut 函数，生成可迭代的数据类型，可以节约空间；如果分词的字符串不长，建议使用 lcut 函数。

3. Jieba 库的使用

由于 cut 函数和 lcut 函数的唯一区别，就是 cut 函数生成可迭代的数据类型，因此下面只是以 lcut 函数举例：

```
>>> import jieba
>>> jieba.lcut('教育部考试中心全国计算机等级考试二级')
Building prefix dict from the default dictionary ...
Dumping model to file cache C:\Users\ds_cf\AppData\Local\Temp\jieba.cache
Loading model cost 0.703 seconds.
Prefix dict has been built succesfully.
['教育部考试中心', '全国', '计算机', '等级', '考试', '二级']
>>> jieba.lcut('教育部考试中心全国计算机等级考试二级',cut_all=True)
['教育', '教育部', '教育部考试中心', '考试', '中心', '全国', '国计', '计算', '计算机', '算机', '等级', '考试', '二级']
>>> jieba.lcut_for_search('教育部考试中心全国计算机等级考试二级')
['教育', '考试', '中心', '教育部', '教育部考试中心', '全国', '计算', '算机', '计算机', '等级', '考试', '二级']
>>>
```

13.3 Wordcloud

Wordcloud 是优秀的词云展示第三方库。词云以词语为基本单位，更加直观和艺术的展示文本。

1. 安装 Wordcloud

Python 默认并不包含 Wordcloud 模块，因此需要自行安装 Wordcloud 模块。安装 Wordcloud 模块与安装其他 Python 模块一样，使用 pip 命令安装即可。在命令行输入如下命令：

```
pip install wordcloud
```

要使用 Wordcloud 库，还得安装 Matplotlib：

```
pip install matplotlib
```

2. WordCloud 对象的参数

Wordcloud 库，把词云当作一个 WordCloud 对象，通过调用 wordcloud.WordCloud()就可以创建 WordCloud 对象。一个 WordCloud 对象代表一个文本对应的词云，可以根据文本

中词语出现的频率等参数绘制词云，绘制词云的形状、尺寸和颜色都可以通过WordCloud()的参数设定。

Wordcloud 的参数为：

表 13-1 Wordcloud 的参数

参数	描述
width	指定词云对象生成图片的宽度，默认 400 像素
height	指定词云对象生成图片的高度，默认 200 像素
min_font_size	指定词云中字体的最小字号，默认 4 号
max_font_size	指定词云中字体的最大字号，根据高度自动调节
font_step	指定词云中字体字号的步进间隔，默认为 1
font_path	指定字体文件的路径，默认 None，中文必需指定
max_words	指定词云显示的最大单词数量，默认 200
stop_words	指定词云的排除词列表，即不显示的单词列表
mask	指定词云形状，默认为长方形，需要引用 imread()函数
background_color	指定词云图片的背景颜色，默认为黑色

3. Wordcloud 库的使用

使用可以概述为三步：

步骤 1：创建和配置对象参数

形如 w = wordcloud.WordCloud(**kwargs)。**kwargs 为参数，具体见表 13-1。

步骤 2：加载词云文本

形如 w.generate(txt)，txt 为要加载的文本，是空格分开的单词文本。对于中文，这一步可以使用 Jieba 分词，把文件中读取的中文文本分割成词组列表，然后用' '.join()连接为用空格分开的单词文本：

```
txt=''
with open('sample.txt','r') as f:   #可把 sample.txt 换成当前文件夹下的某一文本文件
    txt=f.read()
    f.close()
print(txt[:50])
```

步骤 3：输出词云文件，或者屏幕显示

(1) 输出词云文件形如 w.to_file(filename)，filename 为词云图像文件，格式为.png 或.jpg。

(2) 屏幕显示需要 Matplotlib 库。代码大致如下：

```
import matplotlib.pyplot as plt
plt.figure(dpi=100) #通过这里可以放大或缩小
plt.imshow(w, interpolation='catrom',vmax=1000)
plt.axis("off") #隐藏坐标
```

完整的例子：

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import jieba
#1. 准备
#读取文本文件
txt=''
with open('sample.txt','r') as f:   #可把 sample.txt 换成当前文件夹下的某一文本文件
    txt=f.read()
f.close()
#分词
words = jieba.lcut(txt)
cuted=' '.join(words)
#2. 创建 WordCloud 对象
fontpath='simkai.ttf'   #ttf 文件也必需复制到当前文件夹下
w = WordCloud(font_path=fontpath,   # 设置字体
              background_color="white",   # 背景颜色
              max_words=1000,   # 词云显示的最大词数
```

```
                    max_font_size=500,  # 字体最大值
                    min_font_size=20, #字体最小值
                    random_state=42, #随机数
                    collocations=False, #避免重复单词
                    width=1600,height=1200,margin=10, #图像宽高，字间距，需要配合下面的 plt.figure(dpi=xx)放缩才有效
                   )
#3. 加载文本
w.generate(cuted)
#4. 屏幕显示
plt.figure(dpi=100) #通过这里可以放大或缩小
plt.imshow(w, interpolation='catrom',vmax=1000)
plt.axis("off") #隐藏坐标
```

13.4 其他三方库简介

在 Python 中，有很多第三方模块（三方库），涉及到各个领域。

1. 知识导图

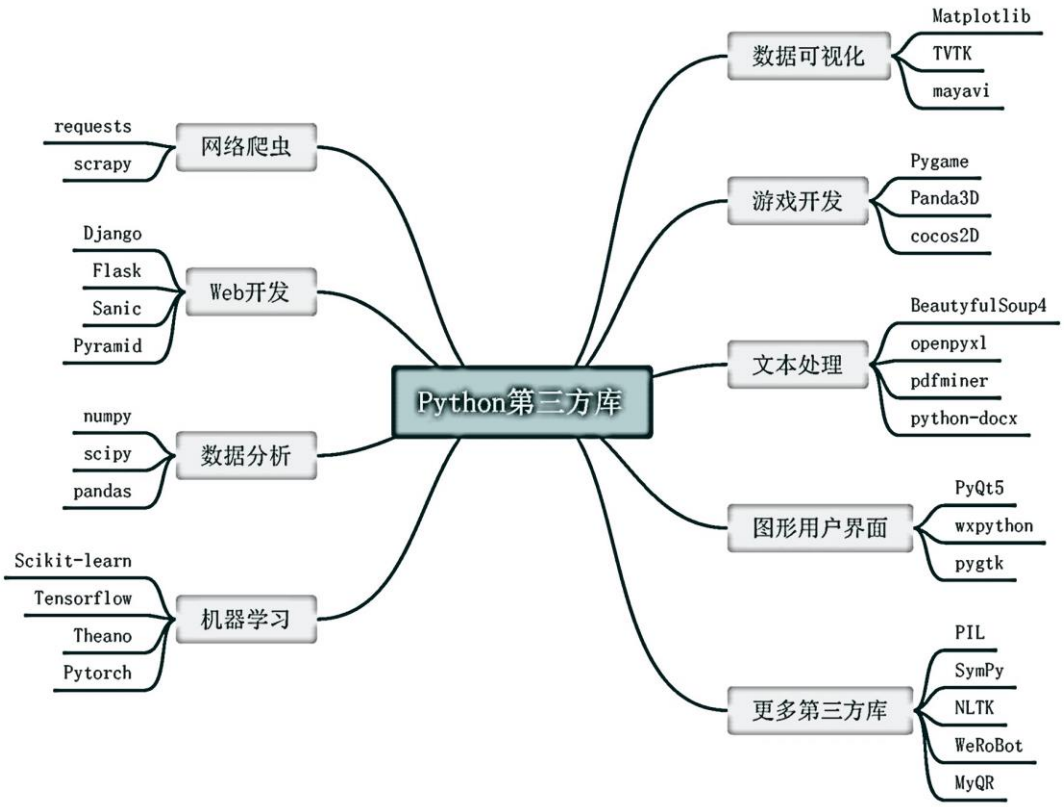


图 13-1 Python 常见第三方库知识导图

2. 爬虫方向

爬虫是自动进行 HTTP 访问并捕获 HTML 页面的程序。Python 语言提供了多个具备网络爬虫功能的第三方库。这里，仅介绍 2 个常用的 Python 网络爬虫库:requests 和 scrapy。

(1) requests

requests 库是一个简洁且简单的处理 HTTP 请求的第三方库，它的最大优点是程序编写过程更接近正常 URL 访问过程。这个库建立在 Python 语言的 urllib3 库基础上。 request 库支持非常丰富的链接访问功能。

(2) scrapy

scrapy 是 Python 开发的一个快速的、高层次的 Web 获取框架。不同于简单的网络爬虫功能，scrapy 框架本身包含了成熟网络爬虫系统所应该具有的部分共用功能

scrapy 用途广泛，可以应用于专业爬虫系统的构建、数据挖掘、网络监控和自动化测试等领域。

3. Web 开发方向

Web 开发是 Python 语言流行的一个重要方向， 主要用于服务器后端开发。根据 Python Web 开发框架很多，这里依次介绍 4 个 Python 第三方库:Django、Pyramid、Flask 和 Sanic

(1) Django

Django 是 Python 生态中最流行的开源 Web 应用框架。 Django 采用模型、模板和视图的编写模式，称为 MTV 模式。Django 中提供了开发网站经常用到的模块，Django 的开发理念是 DRY(Don’t Repeat Yourself)，用于鼓励快速开发，进而减少程序员可以建立一个高性能 Web 应用所花费的时间和精力，形成一种一站式解决方案。

(2) Pyramid

Pyramid 是一个通用、开源的 Python Web 应用程序开发 框架。它主要的目的是让 Python 开发者更简单的创建 Web 应用，相比 Django，Pyramid 是一个相对小巧、快 速、灵活的开源 Python Web 框架。Pyramid 仍然面向较大规模的 Web 应用，但它更关注灵活性，开发者可以灵活选择所使用的数据库、模板风格、URL 结构等内容。

(3) Flask

Flask 是轻量级 Web 应用框架，相比 Django 和 Pyramid， 它也被称为微框架。使用 Flask 开发 Web 应用十分方便，甚至几行代码即可建立一个小型网站。Flask 核心十分简单，并不直接包含诸如数据库访问等的抽象访问层，而是通过扩展模块形式来支持。

(4) Sanic

Sanic 类似于 Flask，也是轻量级 Web 应用框架，也被称为微框架。使用 Sanic 开发 Web 应用十分方便，甚至几行代码即可建立一个小型网站。和 Flask 相比，Sanic 内核使用了 asyncio，因此在所有 Python Web 开发框架中速度是最快的。

3. 数据分析方向

数据分析是 Python 的一个优势方向，具有大批高质量的第三方库。这里仅介绍 3 个最常用的生态库:numpy、scipy 和 pandas。

(1) numpy

numpy 是 Python 的一种开源数值计算扩展第三方库，用于处理数据类型相同的多维数组(ndarray)，简称“数组”。numpy 库可用来存储和处理大型矩阵，比 Python 语言提供的列表结构要高效的多。numpy 提供了许多高级的数值编程工具，如:矩阵运算、矢量处理、N 维数据变换等。

numpy 内部是 C 语言编写，对外采用 Python 语言进行封装，因此，在进行数据运算时，基于 numpy 的 Python 程序可以达到接近 C 语言的处理速度。numpy 也成为 Python 数据分析方向各其他库的基础依赖库，已经成为了科学计算事实上的“标准库”。

(2) scipy

scipy 是一款方便、易于使用、专为科学和工程设计的 Python 工具包。在 numpy 库的基础上增加了众多的数学、 科学以及工程计算中常用的库函数。它包括统计、优化、 整合、线性代数、傅里叶变换、信号分析、图像处理、常微分方程求解等众多模块。

(3) pandas

pandas 是基于 numpy 扩展的一个重要第三方库，它是为了解决数据分析任务而创建的。Pandas 提供了一批标准的数据模型和大量快速便捷处理数据的函数和方法，提供了高效地操作大型数据集所需的工具。

pandas 提供两种最基本的数据类型:Series 和 DataFrame，分别代表一维数组和二维数组类型。

4. 数据可视化方向

数据可视化指根据数据特点将其展示为易于理解 图形的过程。Python 语言在数据可视化方面具有较强的优势。这里介绍 3 个最常用的生态库: matplotlib、TVTK、mayavi。

(1) matplotib

matplotlib 是提供数据绘图功能的第三方库，主要进行二维图表数据展示，广泛用于科学计算的 数据可视化。使用这个库可以利用 Python 程序绘制超过 100 种数据可视化效果。

(2) TVTK

TVTK 库在标准的 VTK 库之上用 Traits 库进行封装的 Python 第三方库。视觉工具函数库(VTK)是一个开源、 跨平台、支持平行处理的图形应用函数库，它是专业可编程的三维可视化工具。TVTK 在 Python 生态系统中被等同于 VTK。

(2) mayavi

mayavi 基于 VTK 开发，完全用 Python 编写，提供了一个更为方便实用的可视化软件，可以简洁地嵌入到用户编写的 Python 程序中，或者直接使用其面向脚本的 API 快速绘制三维可视化图形。值得注意的是，mayavi 也被称 为 mayavi2。

5. 机器学习方向

机器学习是人工智能领域的一个重要分支， Python 语言也是机器学习和人工智能的重要基础语言。这里介绍 3 个高质量的机器学习框架: Scikit−learn、TensorFlow、Theano。

(1) Scikit-learn

Scikit-learn 是一个简单且高效的数据挖掘和数据分析工具，它基于 NumPy、SciPy 和 matplotlib 构建。Scikit- learn 的基本功能主要包括 6 个部分:分类，回归，聚类，数据降维，模型选择和数据预处理。Scikit-learn 也被称为 sklearn。

(2) Tensorflow

TensorFlow 是谷歌公司基于 DistBelief 进行研发的第二代人工智能学习系统，也是用来支撑著名的 AlphaGo 系统的后台框架。Tensor(张量)指 N 维数组，Flow(流) 指基于数据流图的计算，TensorFlow 描述张量从流图的 一端流动到另一端的计算过程。

(3) Theano

Theano 为执行深度学习中大规模神经网络算法的运算而设计，擅长处理多维数组。Theano 开发始于 2007，可以 理解它是一个运算数学表达式的编译器，并可以高效运行在 GPU 或 CPU 上。Theano 是一个偏向底层开发的库，更像一个研究平台而非单纯的深度学习库。

6. 文本处理

Python 语言非常适合处理文本，因此，在这个方向也形成了大量有价值的第三方库。这里介绍 4 个最常用的生态库: BeautifulSoap、pdfminer、openpyxl、python−docx

(1) BeautifulSoap4

BeautifulSoup4 是一个可以从 HTML 或 XML 文件中提取数据的 Python 库。它能够通过你喜欢的转换器实现惯用的文档导航、查找、修改文档的方式，从而会帮你节省数小时甚至数天的工作时间。它的最大优点是 能根据 HTML 和 XML 语法建立解析树，进而高效解析其中的内容。

BeautifulSoup4 库将数据从 HTML 和 XML 文件中解析出来，它能够提供一种符合习惯的方法去遍历搜索和修改解析树，将专业的 Web 页面格式解析部分封装成函数，提供了若干有用且便捷的处理函数。

BeautifulSoup4 因为只是提取 HTML 和 XML 数据而不能获取 Web 数据，因此往往和 requests 配合使用。

(2) openpyxl

openpyxl 是一个处理 Microsoft Excel 文档的 Python 第三方库，它支持读写 Excel 的 xls、xlsx、xlsm、xltx、xltm 等格式文件，并进一步能处理 Excel 文件中 excel 工作表、表单和数据单元。

(3) pdfminer

pdfminer 是一个可以从 PDF 文档中提取各类信息的第三方库。与其他 PDF 相关的工具不同，它能够完全获取并分析 PDF 的文本数据。Pdfminer 能够获取 PDF 中文本的准确位置、字 体、行数等信息，能够将 PDF 文件转换为 HTML 及文本格式。pdfminer 包含两个重要的工具:pdf2txt.py 和 dumppdf.py。pdf2txt.py 能够从 PDF 文件中提取所有文本内容。 dumppdf.py 能够把 PDF 文件内容变成 XML 格式，并进 一步提取其中的图片。

(4) python-docx

python-docx 是一个处理 Microsoft Word 文档的 Python 第三方库，它支持读取、查询以及修改 doc、docx 等格式文件，并能够对 Word 常见样式进行编程设置，包括: 字符样式、段落样式、表格样式等，进一步可以使用这 个库实现添加和修改文本、图像、样式和文档等功能。

7. 用户图形界面方向

Python 标准库内置了一个 GUI 库——tkinter，这个库基于 Tck/Tk 开发，然而，这个库十分陈旧，提供的开发控件也很有限，编写出来的 GUI 风格与现代程序 GUI 风格相差甚远，从用户体验角度说，tkinter 库并不成熟。这里介绍 3 个高质量的用户图形界面 Python 生态 库:PyQt5、wxpython、pygtk。

(1) PyQt5

PyQt5 是 Qt5 应用框架的 Python 第三方库，它有超过 620 个类和近 6000 个函数和方法。它是 Python 中最为成熟的商业级 GUI 第三方库。这个库是 Python 语言当前最好的 GUI 第三方库，它可以在 Windows 和 Linux 等操作系统上跨平台使用。

PyQt5 采用“信号-槽”机制将事件和对应的处理程序进行绑定。PyQt5 窗体有很多内置信号，也可以自定义信号。

(2) wxpython

wxPython 是 Python 语言的一套优秀的 GUI 图形库，它是跨平台 GUI 库 wxWidgets 的 Python 封装，可以使 Python 程序员能够轻松地创建健壮可靠、功能强大的图形用户界面的程序。

(3) pygtk

pygtk 是基于 GTK+的 Python 语言封装，它提供了各式的可视元素和功能，能够轻松创建具有图形用户界面的程序。pygtk 具有跨平台性,利用它编写的代码能够不加修改地稳定运行在各操作系统中，如 Windows 和 Linux。

8. 游戏开发方向

游戏开发是一个有趣的方向，在游戏逻辑和功能 实现层面，Python 已经成为重要的支撑性语言。 这里介绍 3 个 Python 第三方生态库:Pygame、Panda3D、cocos2d。

(1) Pygame

Pygame 是在 SDL 库基础上进行封装的、面向游戏开发入门的 Python 第三方库，除了制作游戏外，还用于制作多媒体应用程序。其中，SDL (Simple DirectMedia Layer)是开源、跨平台的多媒体开发库，通过 OpenGL 和 Direct3D 底层函数提供对音频、键盘、鼠标和图形硬件的简洁访问。

Pygame 是一个游戏开发框架，提供了大量与游戏相关的底层逻辑和功能支持，非常适合作为入门库理解并实践游戏开发。

(2) Panda3D

Panda3D 是一个开源、跨平台的 3D 渲染和游戏开发库，简答说，它是一个 3D 游戏引擎，由迪士尼和卡耐基梅隆大学娱乐技术中心共同进行开发。Panda3D 支持 Python 和 C++ 两种语言，但对 Python 支持更全面。

Panda3D 支持很多当代先进游戏引擎所支持的特性:法线贴图、光泽贴图、HDR、卡通渲染和线框渲染等。

(3) cocos2d

cocos2d 是一个构建 2D 游戏和图形界面交互式应用的框 架，它包括 C++、JavaScript、Swift、Python 等多个版本。cocos2d 基于 OpenGL 进行图形渲染，能够利用 GPU 进行加速。cocos2d 引擎采用树形结构来管理游戏对象，一个游戏划分为不同场景，一个场景又分为不同层，每个层处理并响应用户事件。

9. 更多第三方库

Python 语言有超过 15 万个第三方库，覆盖信息技术几乎所有领域。即使在每个方向，也会有大量的专业人员开发多个第三方库来给出具体设计。除了本章所提到的方向

外，这里再列出 5 个有趣且有用的 Python 第三方库，展示 Python 在工程 实践方面强大的魅力。

(1) PIL

PIL 库是 Python 语言在图像处理方面的重要第三方库， 支持图像存储、显示和处理，它能够处理几乎所有图片 格式，可以完成对图像的缩放、剪裁、叠加以及向图像 添加线条、图像和文字等操作。

PIL 库主要可以完成图像归档和图像处理两方面功能需求:

1) 图像归档:对图像进行批处理、生成图像预览等;

2) 图像处理:图像基本处理、像素处理、颜色处理等。

(2) SymPy

SymPy 是一个支持符号计算的 Python 第三方库，它是一个全功能的计算机代数系统。SymPy 代码简洁、易于理 解，支持符号计算、高精度计算、模式匹配、绘图、解方程、微积分、组合数学、离散数学、几何学、概率与 统计、物理学等领域计算和应用。

(3) NLTK

NLTK 是一个非常重要的自然语言处理 Python 第三方库， 它支持多种语言，尤其对中文支持良好。NLTK 可以进 行语料处理、文本统计、内容理解、情感分析等多种应用，具备非常可靠的应用价值。

(4) WeRoBot

WeRoBot 是一个微信公众号开发框架，也称为的微信机器人框架。WeRoBot 可以解析微信服务器发来的消息，并将消息转换成成 Message 或者 Event 类型。

(5) MyQR

MyQR 是一个能够产生基本二维码、艺术二维码和动态 效果二维码的 Python 第三方库。

第 14 章 综合应用——学生信息管理系统

知识点

综合应用已经学习过的知识点：

(1) **变量**

(2) **流程控制**

(3) **函数**

(4) **模块**

(5) **文件**

开发一个简单的**学生信息管理系统**

系统需求：

(1) 程序启动，显示学生信息管理系统欢迎界面，并显示功能菜单

(2) 用户用数字选择不同的功能

(3) 根据功能选择，执行不同的功能

(4) 用户学生信息需要记录用户的**姓名**、**电话**、**QQ**、**邮件**

(5) 如果查询到指定的学生信息，用户可以选择**修改**或者**删除**学生信息

```
**********************************************************
欢迎使用【学生信息管理系统】V1.0

1. 新建学生信息
2. 显示全部
3. 查询学生信息
4. 保存到文件

0. 退出系统
**********************************************************
```

步骤

(1) 框架搭建

(2) 新增学生信息

(3) 显示所有学生信息

(4) 查询学生信息

(5) 查询成功后修改、删除学生信息

(6) 保存学生信息到文件

(7) 让 Python 程序能够直接运行

14.1 框架搭建

目标

搭建学生信息管理系统**框架结构**

(1) **准备文件**，确定文件名，保证能够**在需要的位置**编写代码

(2) 编写**主运行循环**，实现基本的**用户输入和判断**

14.1.1 文件准备

(1) 新建 info_main.py 保存**主程序功能代码**

1) 程序的入口

2) 每一次启动学生信息管理系统都通过 info_main 这个文件启动

(2) 新建 info_tools.py 保存**所有学生信息功能函数**

- 将对学生信息的**新增**、**查询**、**修改**、**删除**等功能封装在不同的函数中

14.1.2 编写主运行循环

- 在 info_main 中添加一个**无限循环**

```
while True:
    # TODO(陈福明) 显示系统菜单
    action = input("请选择操作功能：")
    print("您选择的操作是：{}".format(action))
    # 根据用户输入决定后续的操作
    if action in ["1", "2", "3", "4"]:
        pass
    elif action == "0":
        print("欢迎再次使用【学生信息管理系统】")
        break
    else:
```

```
        print("输入错误，请重新输入")
```

字符串判断

```
if action in ["1", "2", "3", "4"]:
    pass
if action == "1" or action == "2" or action == "3" or action == "4":
    pass
```

(1) 使用 in 针对**列表**判断，避免使用 or 拼接复杂的逻辑条件

(2) 没有使用 int 转换用户输入，可以避免**一旦用户输入的不是数字**，导致程序运行出错

pass

(1) pass 就是一个空语句，不做任何事情，一般用做占位语句

(2) 是为了保持程序结构的完整性

无限循环

(1) 在开发软件时，如果**不希望程序执行后**立即退出

(2) 可以在程序中增加一个**无限循环**

(3) **由用户来决定**退出程序的时机

TODO 注释

在#后跟上 TODO，用于标记需要去做的工作

```
# TODO(作者/邮件) 显示系统菜单
```

14.1.3 在 info_tools 中增加六个新函数

```
def show_menu():
    """显示菜单
    """
    pass
def new_info():
```

```
    """新建学生信息
    """
    print("-" * 50)
    print("功能：新建学生信息")


def show_all():
    """显示全部
    """
    print("-" * 50)
    print("功能：显示全部")


def search_info():
    """搜索学生信息
    """
    print("-" * 50)
    print("功能：搜索学生信息")


def store_info():
    """保存学生信息到文件
    """
    print("-" * 50)
    print("功能：保存学生信息到文件")
def restore_info():
    """从文件中读取已经保存的学生信息
    """
    print("-" * 50)
    print("功能：保存学生信息到文件")
```

14.1.4 导入模块

在 info_main.py 中使用 import 导入 info_tools 模块

```
import info_tools
```

修改 while 循环的代码如下：

```
import info_tools
while True:
    info_tools.show_menu()
    action = input("请选择操作功能：")
    print("您选择的操作是：{}".format( action))
    # 根据用户输入决定后续的操作
    if action in ["1", "2", "3", "4"]:
        if action == "1":
            info_tools.new_info()
        elif action == "2":
            info_tools.show_all()
        elif action == "3":
            info_tools.search_info()
        elif action == "4":
            info_tools.store_info()

    elif action == "0":
        print("欢迎再次使用【学生信息管理系统】")
        break
    else:
        print("输入错误，请重新输入：")
```

至此，info_main 中的所有代码全部开发完毕！

14.1.5 完成 show_menu 函数

```
def show_menu():
    """显示菜单
    """
    print("*" * 50)
    print("欢迎使用【菜单管理系统】V1.0")
    print("")
    print("1. 新建学生信息")
    print("2. 显示全部")
    print("3. 查询学生信息")
    print("4. 保存到文件")

    print("")
    print("0. 退出系统")
    print("*" * 50)
```

14.2 保存学生信息数据的结构

程序就是用来处理数据的，而变量就是用来存储数据的

(1) 使用**字典**记录**每一张学生信息**的详细信息

(2) 使用**列表**统一记录所有的**学生信息字典**

定义学生信息列表变量

- 在 info_tools 文件的顶部增加一个**列表变量**

```
# 所有学生信息记录的列表
```

```
info_list = []
```

★注意

(1) **所有学生信息相关操作**，都需要使用这个列表，所以应该**定义在程序的顶部**

(2) **程序第一次运行时，没有数据**，所以是**空列表**

- 一般来说，对于各种简单的管理系统，大部分都是这种列表加字典的数据结构。

14.3 新增学生信息

14.3.1 功能分析

(1) 提示用户依次输入学生信息

(2) 将学生信息保存到一个字典

(3) 将字典添加到学生信息列表

(4) 提示学生信息添加完成

14.3.2 实现 new_info 方法

- 根据步骤实现代码

```
def new_info():
    """
    新建学生信息
    """
    print("-" * 50)
    print("功能：新建学生信息")

    # 1. 提示用户输入学生信息
    name = input("请输入姓名：")
    num = input("请输入学号：")
    phone = input("请输入家庭联系电话：")
    address = input("请输入家庭地址：")
```

```
    # 2. 将学生信息保存到一个字典
    info_dict = {"name": name,
                 "num": num,
                 "phone": phone,
                 "address": address }

    # 3. 将学生信息字典添加到学生信息列表
    info_list.append(info_dict)

    print(info_list)

    # 4. 提示添加成功信息
    print("成功添加{}的学生信息".format( info_dict["name"]))
```

技巧：在 PyCharm 中，可以使用 SHIFT + F6 统一修改变量名

14.4 显示所有学生信息

14.4.1 功能分析

- 循环遍历学生信息列表，顺序显示每一个字典的信息

14.4.2 基础代码实现

```
def show_all():
    """
    显示全部
    """
    print("-" * 50)
    print("功能：显示全部")
```

```
    for info_dict in info_list:
        print(info_dict)
```

- 显示效果不好！

14.4.3 增加标题和使用\t 显示

```
def show_all():
    """
    显示全部
    """
    print("-" * 50)
    print("功能：显示全部")

    # 打印表头
    for name in ["姓名", "学号", "家庭电话", "家庭住址"]:
        print(name, end="\t\t")
    print("")
    # 打印分隔线
    print("=" * 50)
    for info_dict in info_list:
        print("{}\t\t{}\t\t{}\t\t{}".format(info_dict["name"],
                                           info_dict["num"],
                                           info_dict["phone"],
                                           info_dict["address"]))
```

14.4.4 增加没有学生信息记录判断

```
def show_all():
    """显示全部
```

```
    """
    print("-" * 50)
    print("功能：显示全部")

    # 1. 判断是否有学生信息记录
    if len(info_list) == 0:
        print("提示：没有任何学生信息记录")
        return
    …
```

★注意

(1) 在函数中使用 return 表示返回

(2) 如果在 return 后没有跟任何内容，只是表示该函数执行到此就不再执行后续的代码

14.5 查询学生信息

14.5.1 功能分析

(1) 提示用户要搜索的姓名

(2) 根据用户输入的姓名遍历列表

(3) 搜索到指定的学生信息后，再执行后续的操作

14.5.2 代码实现

- 查询功能实现

```
def search_info():
    """
    搜索学生信息
    """
    print("-" * 50)
    print("功能：搜索学生信息")
```

```
    # 1. 提示要搜索的姓名
    find_name = input("请输入要搜索的姓名：")

    # 2. 遍历字典
    for info_dict in info_list:
        if info_dict["name"] == find_name:
            print("姓名\t\t\t 学号\t\t\t 家庭电话\t\t\t 家庭住址")
            print("-" * 40)

            print("{}\t\t\t{}\t\t\t{}\t\t\t{}".format( (
                info_dict["name"],
                info_dict["num"],
                info_dict["phone"],
                info_dict["address"]))
            print("-" * 40)
            # TODO(小明) 针对找到的字典进行后续操作：修改/删除
            break
    else:
        print("没有找到 {}" .format( find_name))
```

- 增加学生信息操作函数：**修改/删除/返回主菜单**

```
def deal_info(find_dict):
    """
    操作搜索到的学生信息字典
    :param find_dict:找到的学生信息字典
```

```
    """
    print(find_dict)
    action = input("请选择要执行的操作 "
                   "[1] 修改 [2] 删除 [0] 返回上级菜单")
    if action == "1":
        print("修改")
    elif action == "2":
        print("删除")
```

14.6 修改和删除

14.6.1 查询成功后删除学生信息

(1) 由于找到的字典记录已经在列表中保存

(2) 要删除学生信息记录，只需要把列表中对应的字典删除即可

```
    elif action == "2":
        info_list.remove(find_dict)
        print("删除成功")
```

14.6.2 修改学生信息

(1) 由于找到的字典记录已经在列表中保存

(2) 要修改学生信息记录，只需要把列表中对应的字典中每一个键值对的数据修改即可

```
    if action == "1":
        find_dict["name"] = input("请输入姓名：")
        find_dict["phone"] = input("请输入学号：")
        find_dict["qq"] = input("请输入家庭联系电话：")
        find_dict["email"] = input("请输入家庭住址：")
```

```
            print("{} 的学生信息修改成功".format(find_dict["name"]))
```

修改学生信息细化

如果用户在使用时，某些学生信息内容并不想修改，应该如何做呢？—— 既然系统提供的 input 函数不能满足需求，那么就新定义一个函数 input_info_info 对系统的 input 函数进行扩展：

```
def input_info_info(dict_value, tip_message):
    """输入学生信息信息
    :param dict_value: 字典原有值
    :param tip_message: 输入提示信息
    :return: 如果输入，返回输入内容，否则返回字典原有值
    """
    # 1. 提示用户输入内容
    result_str = input(tip_message)
    # 2. 针对用户的输入进行判断，如果用户输入了内容，直接返回结果
    if len(result_str) > 0:
        return result_str
    # 3. 如果用户没有输入内容，返回 `字典中原有的值`
    else:
        return dict_value
```

14.7 保存学生信息列表到文件

保存数据到文件，也就是数据的持久化。我们的学生信息系统，所有学生信息构成列表，列表和字典都是高级数据。对于高级数据的持久化，可以参考文件那一章中的高级数据文件存取一节编写。这里使用 pickle 模块，pickle 模块使用 pickle.dump 保存数据到文件，使用 pickle.load 把保存在文件中的数据读取到变量中。

```
import pickle
filename="info.data"    #把保存学生信息的文件的文件名定义为全局变量
```

```
def store_info():
    """保存学生信息到文件
    """
    global filename, info_list
    with open(filename,'wb') as file:
        pickle.dump(info_list,file)
    print("已经保存学生信息到文件{}了".format(filename))
def restore_info():
    """从文件中读取已经保存的学生信息
    """
    global filename, info_list
    with open(filename,'rb') as file:
        info_list = pickle.load(file)
    print("从文件中读取到了已经保存的学生信息")
```

系统每次启动前，都要判断有没有已经保存的学生信息，如果有，那么就存在那个文件名 filename，否则就不存在。对存在已经保存了的学生信息，那么就得读取到学生信息列表变量 info_list 中。我们用 init_info 函数来完成：

```
def init_info():
    """初始化学生信息列表
    """
    import os
    global info_list
    if os.path.exists (filename):
        info_list = restore_info()
```

14.8 __name__属性的使用

在 Python 中，经常使用 if __name__ == '__main__':作为整个 py 文件(模块)的执行的入口。如下面的代码：

```
if __name__ == '__main__':
    #初始化学生信息列表
    init_info()
    #循环显示系统菜单
    while True:
        ……
```

__name__的含义是：

(1) __name__就是标识模块的名字的一个系统变量

(2) 如果模块是被导入，__name__的值为模块名字

(3) 如果模块是被直接执行，__name__的值为'__main__'

14.9 Linux 上的 Shebang 符号(#!)

(1) #!这个符号叫做 Shebang 或者 Sha-bang

(2) Shebang 通常在 Unix 系统脚本的中**第一行开头**使用

(3) 指明**执行这个脚本文件**的**解释程序**

使用 Shebang 的步骤：

使用 which 查询 python 解释器所在路径

```
$ which python
```

修改要运行的**主 python 文件**，在第一行增加以下内容

```
#! /usr/bin/python
```

修改**主 python 文件**的文件权限，增加执行权限

```
$ chmod +x info_main.py
```

在需要时执行程序即可

```
./info_main.py
```

14.10 完整的代码

14.10.1 info_tools.py

```
"""学生信息管理系统操作函数
"""
# 所有学生信息记录的列表
info_list = []
def show_menu():
    """显示菜单
    """
    print("*" * 50)
    print("欢迎使用【菜单管理系统】V1.0")
    print("")
    print("1. 新建学生信息")
    print("2. 显示全部")
    print("3. 查询学生信息")
    print("4. 保存到文件")

    print("")
    print("0. 退出系统")
    print("*" * 50)

def new_info():
    """
    新建学生信息
```

```
    """
    print("-" * 50)
    print("功能：新建学生信息")
    # 1. 提示用户输入学生信息
    name = input("请输入姓名：")
    num = input("请输入学号：")
    phone = input("请输入家庭联系电话：")
    address = input("请输入家庭地址：")

    # 2. 将学生信息保存到一个字典
    info_dict = {"name": name,
                 "num": num,
                 "phone": phone,
                 "address": address }
    # 3. 将用户字典添加到学生信息列表
    info_list.append(info_dict)
    print(info_list)
    # 4. 提示添加成功信息
    print("成功添加{}的学生信息".format( info_dict["name"]))

def show_all():
    """
    显示全部
    """
    print("-" * 50)
    print("功能：显示全部")
```

```
    # 打印表头
    for name in ["姓名", "学号", "家庭电话", "家庭住址"]:
        print(name, end="\t\t")
    print("")
    # 打印分隔线
    print("=" * 50)
    # 1. 判断是否有学生信息记录
    if len(info_list) == 0:
        print("提示：没有任何学生信息记录")
        return
    for info_dict in info_list:
        print("{}\t\t{}\t\t{}\t\t{}".format(info_dict["name"],
                                            info_dict["num"],
                                            info_dict["phone"],
                                            info_dict["address"]))
def input_info_info(dict_value, tip_message):
    """输入学生信息信息
    :param dict_value: 字典原有值
    :param tip_message: 输入提示信息
    :return: 如果输入，返回输入内容，否则返回字典原有值
    """
    # 1. 提示用户输入内容
    result_str = input(tip_message)
    # 2. 针对用户的输入进行判断，如果用户输入了内容，直接返回结果
    if len(result_str) > 0:
        return result_str
    # 3. 如果用户没有输入内容，返回 `字典中原有的值`
```

```
        else:
            return dict_value
def deal_info(find_dict):
        """
        操作搜索到的学生信息字典
        :param find_dict:找到的学生信息字典
        """
        print(find_dict)
        action = input("请选择要执行的操作 "
                                    "[1] 修改 [2] 删除 [0] 返回上级菜单")
        if action == "1":
            find_dict["name"] = input_info_info(find_dict["name"],"请输入姓名：")
            find_dict["num"] = input_info_info(find_dict["num"],"请输入学号：")
            find_dict["phone"] = input_info_info(find_dict["phone"],"请输入家庭联系电话：")
            find_dict["address "] = input_info_info(find_dict["address"],"请输入家庭住址：")
            print("{}的学生信息修改成功".format(find_dict["name"]))

        elif action == "2":
            info_list.remove(find_dict)
            print("删除成功")

def search_info():
        """
        搜索学生信息
        """
        print("-" * 50)
        print("功能：搜索学生信息")
```

```
        # 1. 提示要搜索的姓名
        find_name = input("请输入要搜索的姓名：")
        # 2. 遍历字典
        for info_dict in info_list:
            if info_dict["name"] == find_name:
                print("姓名\t\t\t 学号\t\t\t 家庭电话\t\t\t 家庭住址")
                print("-" * 40)

                print("{}\t\t{}\t\t{}\t\t{}".format(
                    info_dict["name"],
                    info_dict["num"],
                    info_dict["phone"],
                    info_dict["address"]))
                print("-" * 40)
                # TODO(明) 针对找到的字典进行后续操作：修改/删除
                deal_info(info_dict)
                break
        else:
            print("没有找到 {}" .format(find_name))


import pickle
filename="info.data"   #把保存学生信息的文件的文件名定义为全局变量
def store_info():
    """保存学生信息到文件
    """
    global filename,info_list
    with open(filename,'wb') as file:
```

```
            pickle.dump(info_list,file)
        print("已经保存学生信息到文件{}了".format(filename))
def restore_info():
        """从文件中读取已经保存的学生信息
        """
        global filename
        with open(filename,'rb') as file:
            info = pickle.load(file)
        print("从文件中读取到了已经保存的学生信息")
        return info
def init_info():
        """初始化学生信息列表
        """
        import os
        global info_list
        if os.path.exists (filename):
            info_list = restore_info()
```

14.10.2 info_main.py

```
#! /usr/bin/python
import info_tools
if __name__ == '__main__':
        info_tools.init_info()
        while True:
            info_tools.show_menu()
            action = input("请选择操作功能：")
            print("您选择的操作是：{}".format(action))
```

```
        # 根据用户输入决定后续的操作
        if action in ["1", "2", "3", "4"]:
            if action == "1":
                info_tools.new_info()
            elif action == "2":
                info_tools.show_all()
            elif action == "3":
                info_tools.search_info()
            elif action == "4":
                info_tools.store_info()
        elif action == "0":
            print("欢迎再次使用【学生信息管理系统】")
            break
        else:
            print("输入错误，请重新输入：")
```

学过 C 语言、Java 语言或者 C#语言的读者，可以看到，用 Python 做一个简单的管理系统，代码量要少很多，因此：人生苦短，我用 Python！

习 题

一、程序题

1、仿照本章实例，开发一个简单的管理系统。