

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Python 入门到高级进阶

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：



第 1 章：初识 Python

1、Python 概述

1.1、Python 简介

Python 英文本义是指“蟒蛇”。1989 年，由荷兰人 Guido van Rossum 发明的一种面向对象的解释型高级编程语言，命名为 Python。Python 的设计哲学为优雅、明确、简单。实际上，Python 也始终贯彻这个理念，以至于现在网络上流传着“人生苦短，我用 Python”的说法，可见 Python 有着简单、开发速度快、节省时间和容易学习等特点。

Python 是一种跨平台的、开源的、免费的解释型的高级编程语言。具有非常丰富和非常强大的库。Python 的应用领域也非常广泛，在 Web 编程、图形处理，黑客编程，大数据处理、网络爬虫和科学计算等领域都能找到 Python 的身影。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Python 蟒蛇：



Python Logo：



1.2、Python 的版本

Python 自从发布以来，主要经历了三个版本的变化。分别是 1994 年发布的 Python 1.0 版本（已过时了）、2000 年发布的 python 2.0 版本（几乎没人使用了）和 2008 年发布的 python 3.0 版本（2020 年 11 月已经更新到了 Python 3.10.0）。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

目前企业使用的 Python 版本为 3.x 的版本，一般建议大家学习直接学习 python 3.x 即可。现在再学 2.x 版本已经没什么意义了。

1.3、Python 的应用领域

Python 功能十分强大，概括起来主要有以下几个应用领域：

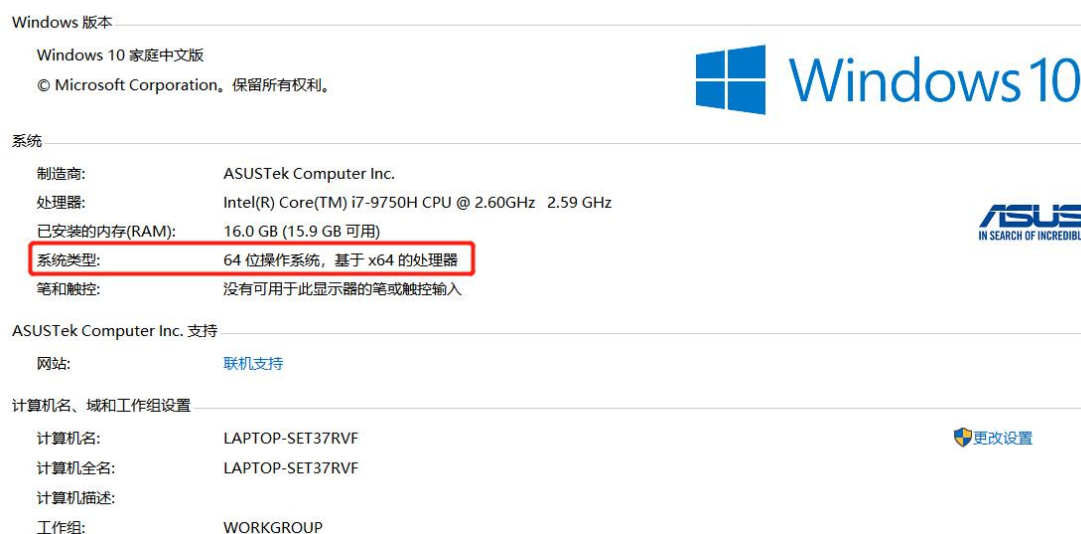
- Web 开发
- 大数据处理
- 人工智能
- 自动化运维开发
- 云计算
- 爬虫
- 游戏开发

2、搭建 Python 3.10 开发环境

2.1、Windows 系统搭建 Python 环境

1、首先查看自己电脑操作系统的位数

[查看有关计算机的基本信息](#)



Windows 版本

Windows 10 家庭中文版

© Microsoft Corporation. 保留所有权利。

系统

制造商: ASUSTek Computer Inc.

处理器: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

已安装的内存(RAM): 16.0 GB (15.9 GB 可用)

系统类型: 64 位操作系统, 基于 x64 的处理器

笔和触控: 没有可用于此显示器的笔或触控输入

ASUSTek Computer Inc. 支持

网站: [联机支持](#)

计算机名、域和工作组设置

计算机名: LAPTOP-SET37RVF

计算机全名: LAPTOP-SET37RVF

计算机描述:

工作组: WORKGROUP

现在电脑几乎都是 64 位操作系统，不过为了避免安装出错，这里大家确认一下自己操作系统的位数。

2、下载 Python 软件包

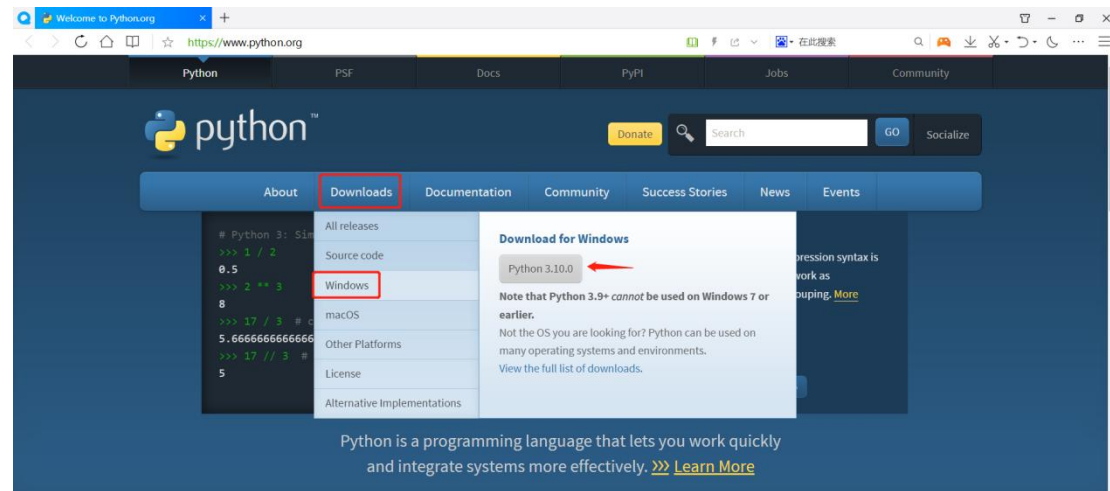
在 Python 的官方网站中，可以很方便地下载 Python 的开发环境。具体下载步骤：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1、进入 Python 的官方网站，地址为：“<https://www.python.org>”

直接选择对应的操作系统，也可以选择下载右侧的版本。默认右侧的版本为最新版本，这里选择对应的操作系统版本即可。



2、下载列表中选择对应版本进行下载。

Python 软件包分为 embeddable package 和 installer。embeddable package 下载之后可以直接使用。Install 则需要安装才能使用，就和我们平时安装软件是一样的道理。Help file 则是帮助文件。这里我们下载 Download Windows installer (64-bit)。

- Download Windows embeddable package (32-bit)
- Download Windows embeddable package (64-bit)
- Download Windows help file
- Download Windows installer (32-bit)
- Download Windows installer (64-bit)

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Python Releases for Windows

- [Latest Python 3 Release - Python 3.10.0](#)
- [Latest Python 2 Release - Python 2.7.18](#)

Stable Releases

- [Python 3.10.0 - Oct. 4, 2021](#)

Note that Python 3.10.0 *cannot* be used on Windows 7 or earlier.

- Download [Windows embeddable package \(32-bit\)](#)
- Download [Windows embeddable package \(64-bit\)](#)
- Download [Windows help file](#)
- Download [Windows installer \(32-bit\)](#)
- Download [Windows installer \(64-bit\)](#)
- [Python 3.7.12 - Sept. 4, 2021](#)

Note that Python 3.7.12 *cannot* be used on Windows XP or earlier.



下载完，双击图标运行：

3、双击安装包之后，得到如下对话框

Install Now: 默认安装，路径不能修改，建议不要安装在该目录下。

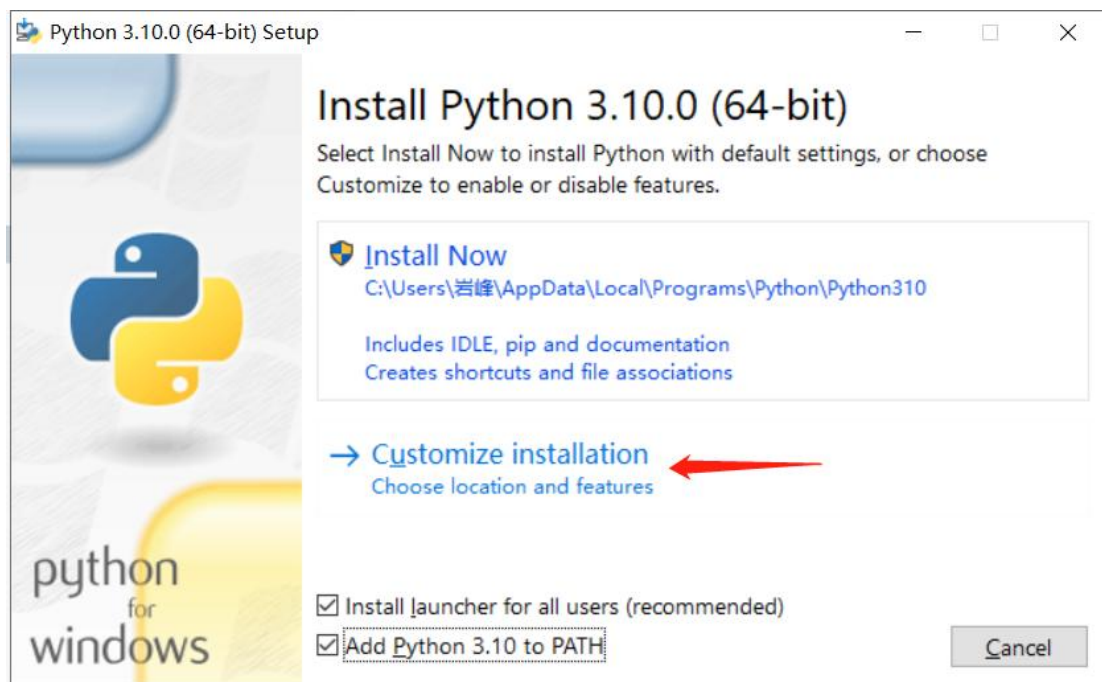
Customize installation: 选择自定义安装

勾选下面的“Add Python 3.10 to PATH”，这表示让安装程序自动配置环境变量。这个必须勾选，如果不勾选，那么在 CMD 窗口无法直接调用 python 相关指令。

当然如果你足够牛，可以自己添加环境变量。我是为了避免麻烦，嘿嘿。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



点击 Customize installation 自定义安装。

4、选择“Next” 下一步

Documentation: 安装 Python 帮助文档。

pip: 安装下载 Python 包的工具 pip。

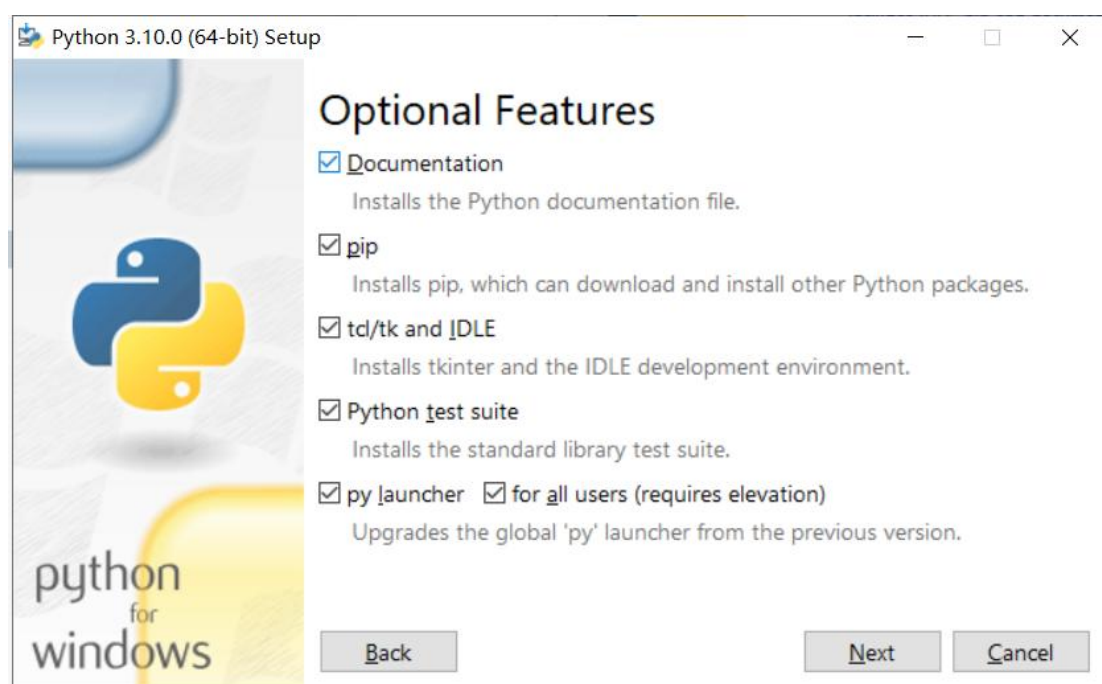
td/tk and IDLE: 安装 Tkinter 和 IDLE 开发环境。

Python test suite: 安装标准库测试套件。

“py launcher” “for all users (requires elevation)”：安装所有用户都可以启动 Python 的发射器。

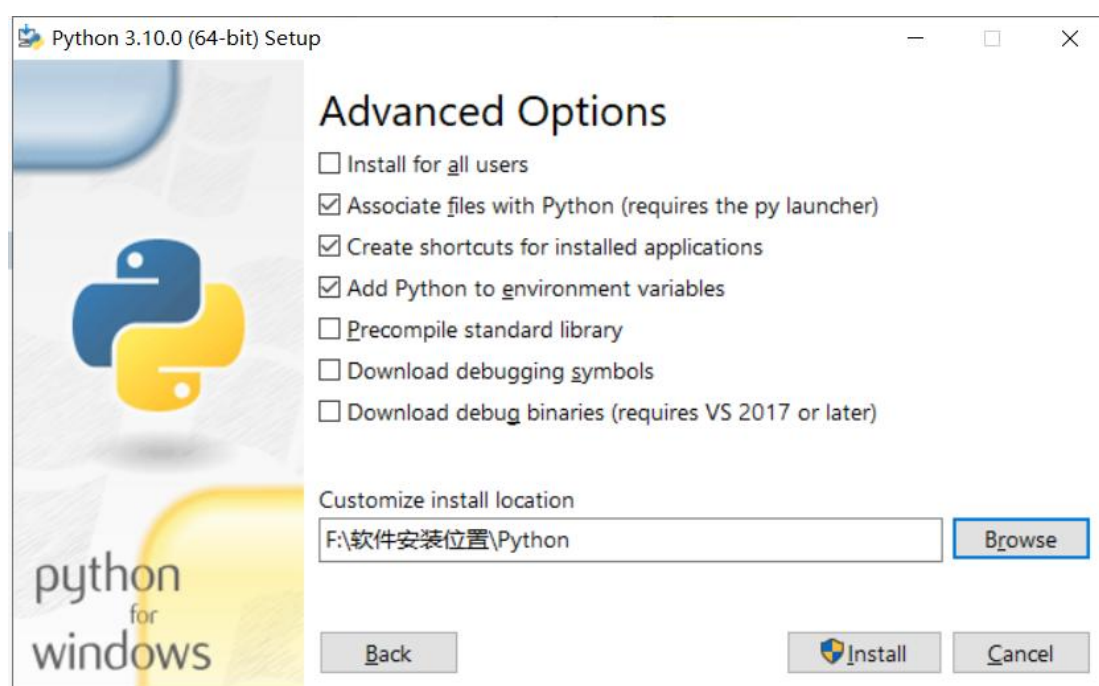
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



5、选择“Install”进行安装

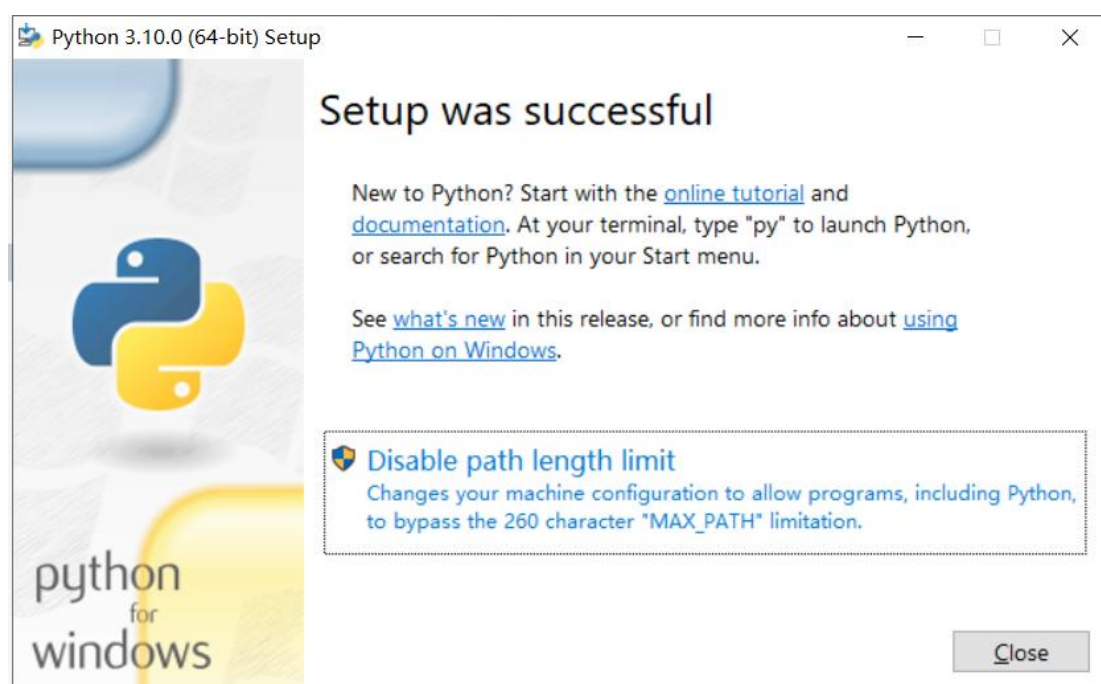
这里默认即可，唯一要修改的就是我们的安装路径。



6、安装完成，选择“Close”退出安装程序

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



7、测试 Python 是否安装成功

Python 安装成功后，需要检测 Python 是否安装成功。在 cmd 命令提示符后面输入“python”，得到如下结果，则说明 Python 安装成功，同时也进入到了交互式 Python 解释器中。

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [版本 10.0.19042.1288]
(c) Microsoft Corporation。保留所有权利。

C:\Users\岩峰>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

2.2、Linux 系统搭建 Python 环境

Linux 系统是为编程而设计的，因此在大多数 Linux 计算机中，都默认安装了 Python。编写和维护 Linux 的人认为，你可能会使用这种系统进行编程，他们也鼓励你这样做。鉴于此，要在这种系统中编程，你几乎不用安装什么软件，也几乎不用修改设置。不过有个弊端，自带的 python 版本都是比较低的，有的 Linux 版本还是 Python 2.x 的版本。所以建议自行部署。

1、下载 Python 软件包

下载地址：<https://www.python.org/downloads/source/>

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

我们这里使用最新版本，这里的 Download 下载的软件包都是压缩包的格式，这两种格式分别是 tgz 和 tar.xz。

- Download Gzipped source tarball: *.tgz 格式
- Download XZ compressed source tarball: *.tar.xz 格式



2、安装依赖软件包

```
[root@localhost ~]# yum -y install gcc-c++ zlib-devel bzip2-devel  
openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel  
gdbm-devel xz-devel make
```

3、上传、解压 Python 软件包

```
[root@localhost ~]# ll Python-3.10.0.tgz  
-rw-r--r--. 1 root root 25007016 Nov  1 10:27 Python-3.10.0.tgz  
[root@localhost ~]# tar -zxvf Python-3.10.0.tgz
```

4、编译安装 Python

```
[root@localhost ~]# cd Python-3.10.0  
[root@localhost Python-3.10.0]# ./configure  
--prefix=/usr/local/python3  
[root@localhost Python-3.10.0]# make
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@localhost Python-3.10.0]# make install
```

5、替换旧版本

```
[root@localhost ~]# mv /usr/bin/python /usr/bin/python_old
[root@localhost ~]# ln -s /usr/local/python/bin/python3
/usr/bin/python
[root@localhost ~]# python -V
Python 3.10.0
```

3、搭建 PyCharm 开发环境

3.1、PyCharm 简介

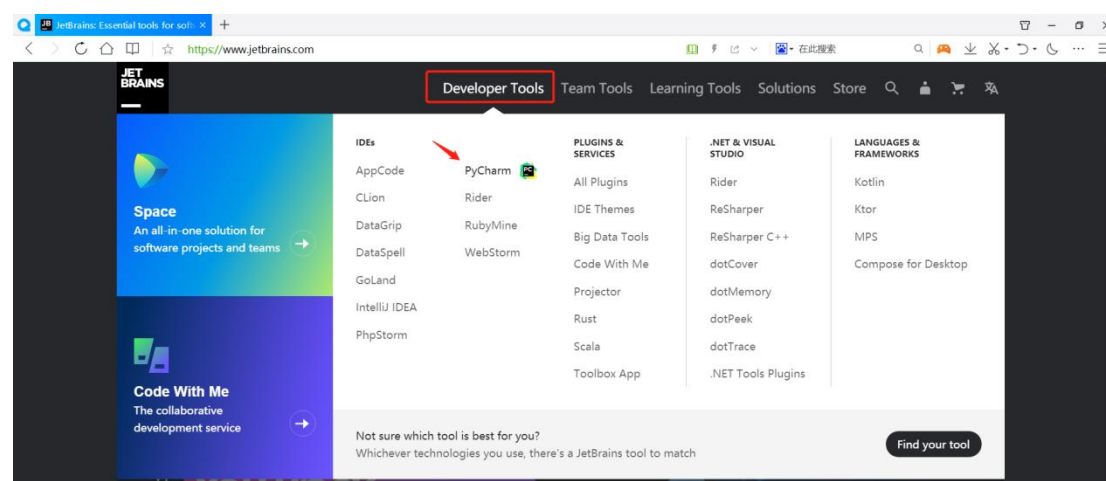
PyCharm 是由 JetBrains 公司开发的 Python 集成开发环境，由于其具有智能代码编辑器，可实现自动代码格式化、代码完成、智能提示、重构、单元测试、自动导入和一键代码导航等功能，目前已成为 Python 专业开发人员和初学者使用的有力工具。

3.2、PyCharm 下载与安装

PyCharm 官网地址：“<http://www.jetbrains.com>”

1、下载 PyCharm 软件包

打开“<http://www.jetbrains.com>”官网，选择“Developer Tools”，再选择“PyCharm”，进入下载界面。在选择“download”进行下载。

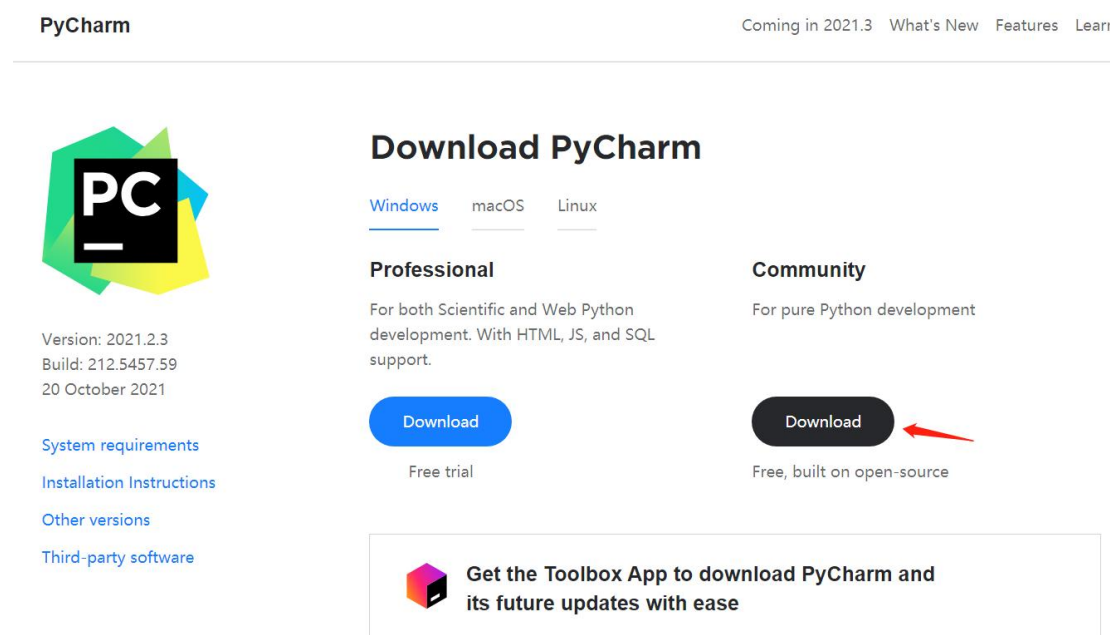


版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



PyCharm 分为“Professional”专业版（收费）和“Community”社区版（免费），这里选择“Community”进行下载，下载社区版。



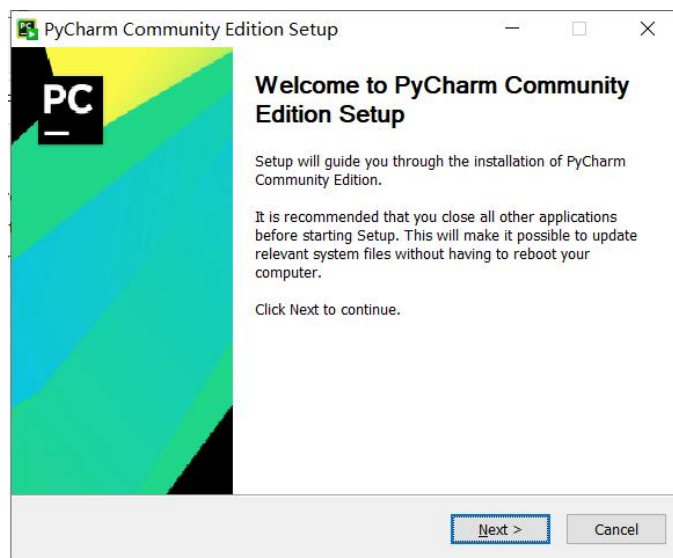
下载完成查看：

名称	修改日期	类型	大小
 pycharm-community-2021.2.3.exe	2021/11/3 16:23	应用程序	381,272 KB

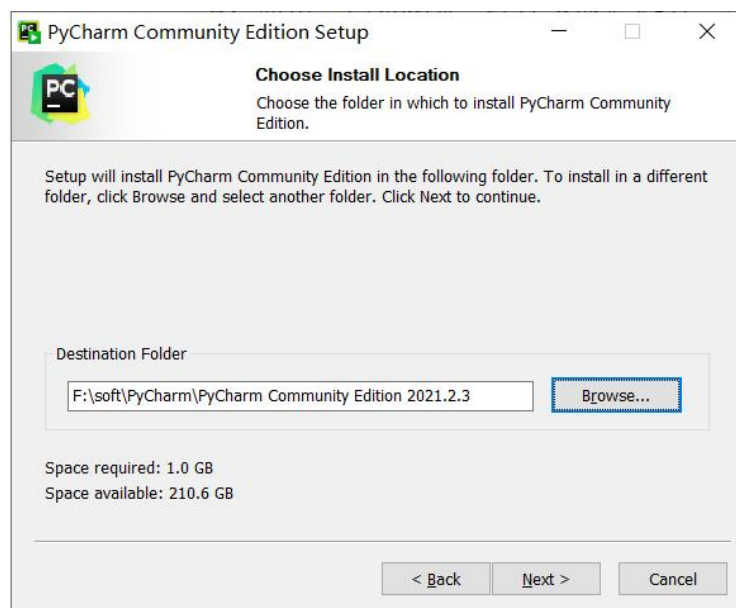
2、点击“PyCharm”进行安装。选择“Next”进行下一步。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



3、选择安装位置



4、勾选需要的功能

create desktop shortcut: 创建桌面快捷方式

Update Context Menu: 更新关联菜单

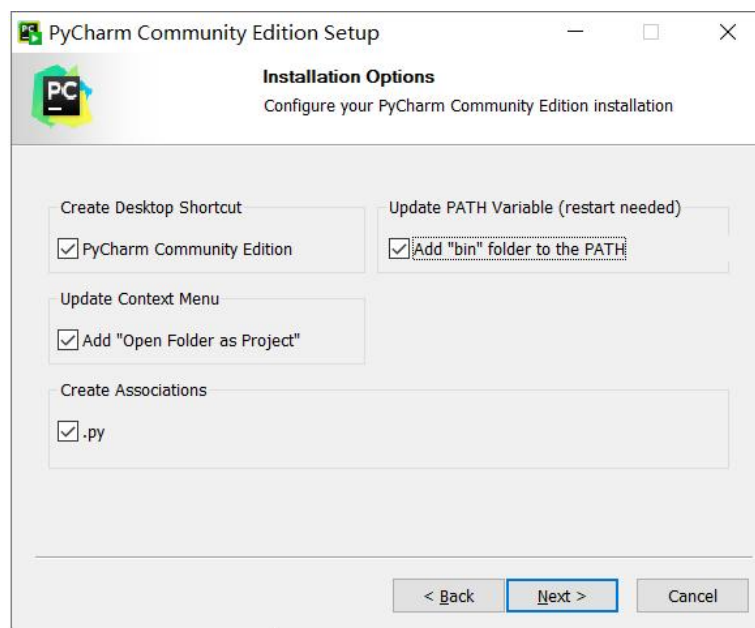
Create Associations: 创建关联

Update PATH Variable (restart needed): 更新环境变量（需要重新启动）

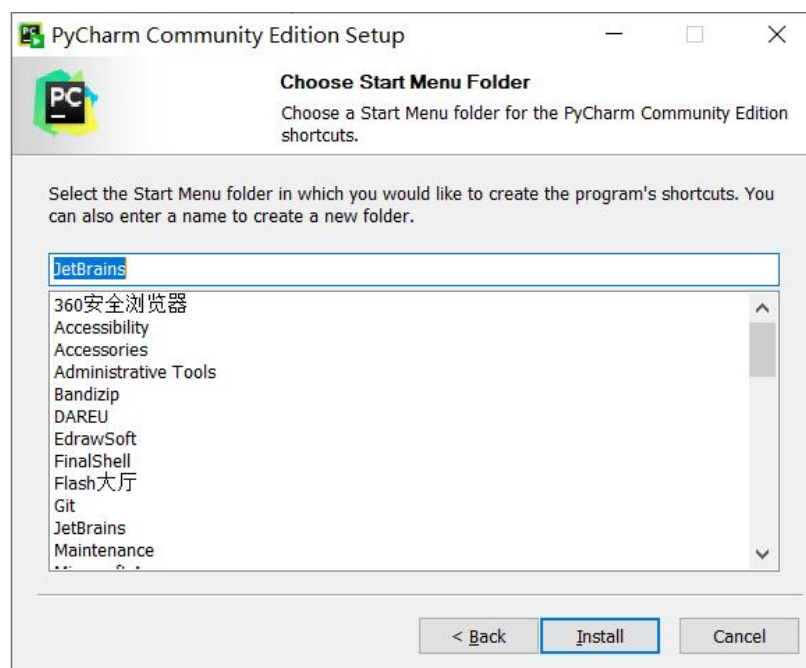
这么都选择即可，选择完毕之后点击“Next”下一步。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



5、确认没问题，选择“Install”进行安装。



6、完成安装

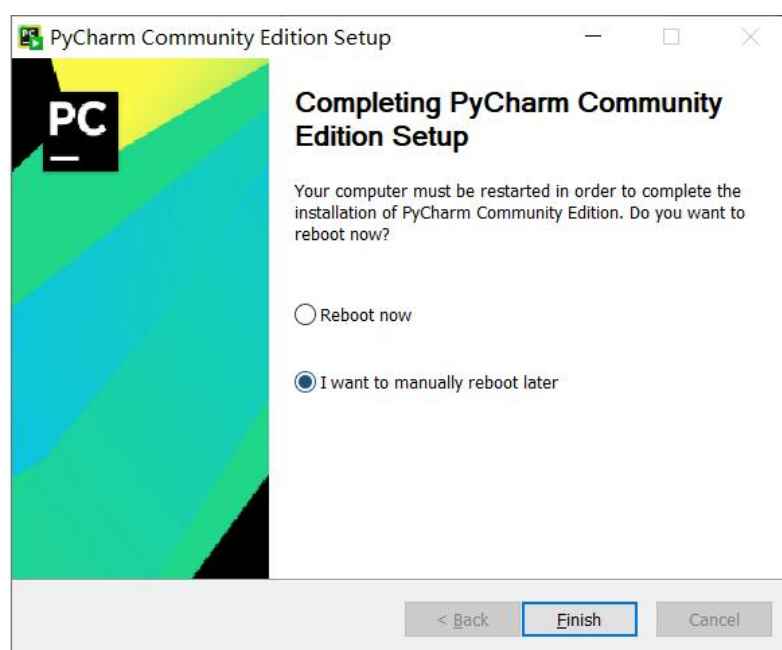
Reboot now: 重启

I want to manually reboot later: 稍后重新启动

可以选择“Reboot now”直接重启，重启即生效。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

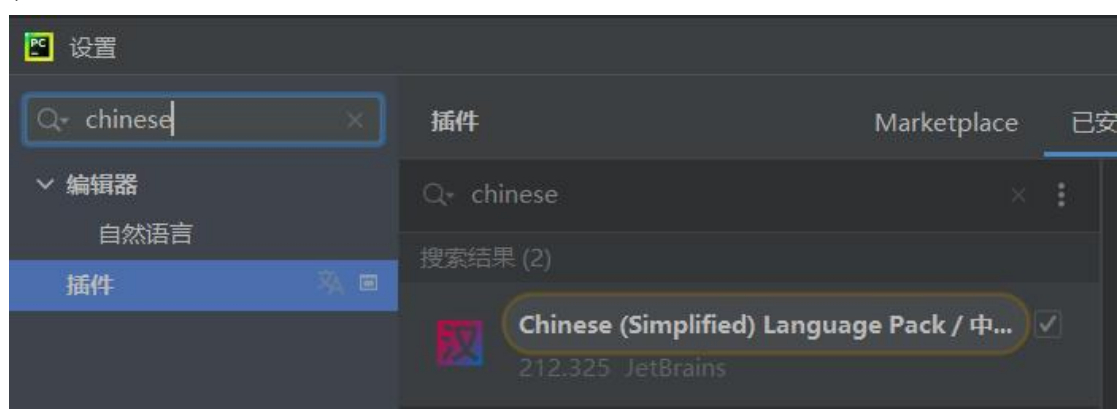
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



安装完毕。

7、安装 Python 中文汉化

搜索 Chinese，插件的 Logo 如下图所示，安装完重新打开软件即可完成汉化。



第 2 章：Python 输入与输出

1、print() 函数输出

print() 函数的基本语法格式如下：

```
print(输出内容)
```

输出内容可以是数字和字符串（字符串需要使用引号括起来），此类内容将直接输出，也可以是运算符的表达式，此类内容将计算结果输出。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 1:

代码如下:

```
a = 100    # 定义变量 a, 值为 100
b = 5      # 定义变量 b, 值为 5
print(10)  # 输出数字 10
print(a)   # 输出变量 a 的值
print(a*b) # 输出变量 a 的值乘以变量 b 的值的结果
print('hello world') # 输出 hello world 字符串
```

运行结果:

```
10
100
500
hello world
```

在 Python 中, 默认情况下, 一条 `print()` 语句输出后会自动换行, 如果想要一次输出多个内容, 而且不换行, 可以将要输出的内容使用英文半角的逗号分隔。如下实例 2

实例 2:

代码如下:

```
a = 100
b = 5
print(a,b,a*b,'hello world')
```

运行结果:

```
100 5 500 hello world
```

实例 3:

使用 `print()` 函数, 不但可以将内容输出到屏幕, 还可以输出到指定文件中。例如将一个字符串输出到文件中。

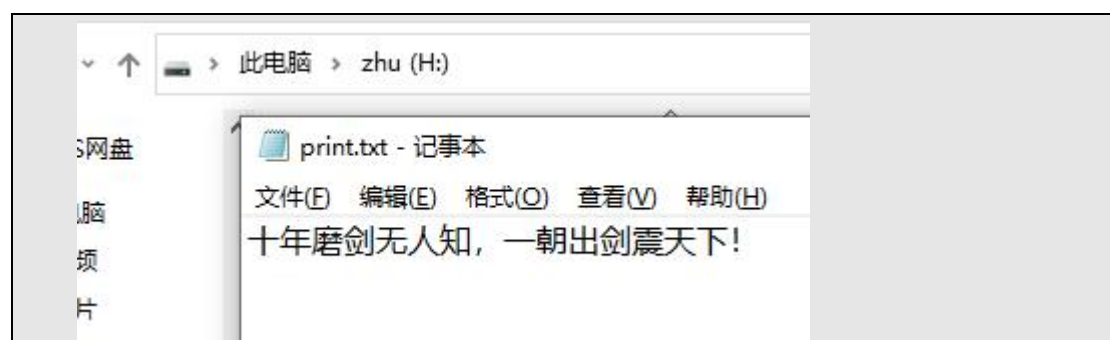
代码如下:

```
data = open(r'H:\print.txt','a+') # 打开文件
print("十年磨剑无人知, 一朝出剑震天下!",file=data) # 输出到文件中
data.close() # 关闭文件
```

运行结果:

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



提示：执行上面的代码，就是在指定的目录下创建的一个 print.txt 的文件，双击打开，写一些内容，关闭文件。无非就是这些动作。

实例 4:

在 linux 系统中，我们经常以时间为单位去备份文件，那么 Python 是否可以将年份、月份和日期也输出出来呢？

of course，当然可以了，但需要先调用 datetime 模块，并且按指定格式才可以输出相应日期。

输出当前年份和当前日期时间

代码如下：

```
import datetime    # 调用日期模块 datetime
print('当前年份:' + str(datetime.datetime.now().year))    #输出当前年份
print('当前日期时间:'
      + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'))    #输出当前日期和时间，%Y：年、%m：月、%d：日、%H：时、%M：分、%S：秒。
```

运行结果：

当前年份：2021

当前日期时间：2021-12-14 12:56:42

2、input () 函数输入

input () 函数的基本语法格式如下：

```
variable = input("提示文字")
```

其中，variable 为保存输入结果的变量，双引号内的文字用于提示要输入的内容。

例如，想要接收用户输入的内容，并保存到变量 var 中，可以使用下面的代码：

```
var = input("请输入内容：")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 3.x 中，无论输入的是数字还是字符都将被作为字符串读取。如果想要接收数值，需要把接收到的字符串进行类型转换。

例如：想要接收整型的数字并保存到变量 num 中，可以使用下面的代码：

```
num = int(input("请输入数字："))
```

实例 1：

实现根据输入出生的年份，计算出年龄大小

代码如下：

```
import datetime
nowyear = datetime.datetime.now().year
myyear = int(input("请输入您的出生年份："))
print(nowyear-myyear)
```

运行结果：

```
请输入您的出生年份：2001
20
```

3、Python 注释

注释，是指代码中对代码功能进行解释说明的标注性文字，可以提高代码的可读性。注释的内容将被 Python 解释器忽略，并不会在结果中体现出来。

● 单行注释

在 Python 中，使用“#”作为单行注释的符号。从符号“#”开始直到换行为止，后面跟的所有内容都作为注释的内容，从而被 Python 忽略。

语法如下：

```
# 注释内容
```

单行注释可以放在注释代码的前一行，也可以放在要注释代码的右侧。如下，下面两种注释形式都是正确的。

第一种注释形式：

```
# 定义变量 nowyear 等于当前年份
nowyear = int(datetime.datetime.now().year)
```

第二种注释形式：

```
nowyear = int(datetime.datetime.now().year) # 定义变量 nowyear 等于当前年份
```

● 多行注释

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 中，并没有一个单独的多行注释标记，而是将包含在一对三引号（'''.....'''）或者（"""....."""）之间的代码都称为注释。这样的代码解释器将忽略。由于这样的代码可以分为多行注释，所以也作为多行注释。

语法格式如下：

```
'''  
注释内容 1  
注释内容 2  
'''  
  
或  
"""  
注释内容 1  
注释内容 2  
"""
```

多行注释通常用来为 Python 文件、模块、类或者函数等添加版权、功能等信息，例如，下面代码将使用多行注释为程序添加功能、开发者、版权、开发日期等信息。

```
"""  
信息加密模块  
开发者：feng  
版权所有：feng  
开发日期：2021-12-14  
版本：v1  
"""
```

多行注释也经常用来解释代码中重要的函数、参数等信息，以便于后续开发者维护代码，例如：

```
"""  
库存类主要的函数用法  
del 删除  
find 查找  
"""
```

4、中文编码声明注释

在 Python 中编写代码的时候，如果用到指定字符编码类型的中文编码，需要在文件开头加上中文声明注释，这样可以在程序中指定字符编码类型的中文编码，不至于出现代码错误。所以说，中文注释很重要。

Python3.x 提供的中文注释声明语法格式如下：

```
# *_ coding:utf-8 *_  
或者
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# coding:utf-8
```

__没有特殊的作用，只是为了美观才加上的。

Python 开头规范书写格式

```
#!/usr/bin/env python
# __ coding:utf-8 __
# 作者: zyf
# 功能:
# 时间:
# 版本:
```

5、代码缩进规范

Python 不像其他程序设计语言（如 Java 或者 C 语言）采用大括号“{}”分隔代码块，Python 而是采用代码缩进和冒号“:”区分代码之间的层次。

缩进可以使用空格键或者<Tab>键实现。使用空格键时，通常情况下采用 4 个空格作为一个缩进量，而使用<Tab>键时，则采用一个<Tab>键作为一个缩进量。通常情况下建议采用空格进行缩进。

Python 对代码的缩进要求非常严格，同一级别的代码块缩进量必须相同。如果不采用合理的代码缩进，将抛出 SyntaxError 异常。

6、编码规范

Python 中采用 PEP8 作为编码规范，其中 PEP 是 Python Enhancement Proposal 的缩写，翻译过来是 Python 增强建议书，而“PEP8”中的“8”标识版本号，PEP8 是 Python 代码的样式指南。

下面给出一些应该严格遵守的条目：

1、每个 import 语句只导入一个模块，尽量避免一次导入多个模块。

推荐写法：

```
import os
import sys
```

不推荐写法：

```
import os, sys
```

2、不要在行尾添加分号“;”，也不用分号将两条命令放在同一行。

推荐写法：

```
height = float(input("请输入您的身高："))
weight = float(input("请输入您的体重："))
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
不推荐写法：
height = float(input("请输入您的身高："));
weight = float(input("请输入您的体重："));
或
height = float(input("请输入您的身高："));weight = float(input("
请输入您的体重："));
```

3、建议每行不超过 80 个字符，如果超过，建议用小括号“()”将多行内容隐式的连接起来，而不推荐使用反斜杠“\”进行连接。

例如一个字符串文本在一行上显示不下，那么可以使用小括号“()”将其分行显示。代码如下：

```
推荐写法：
print("白日依山尽，"
      "黄河入海流。")

不推荐写法：
print("白日依山尽，" \
      "黄河入海流。")
```

4、使用必要的空行可以增加代码的可读性。一般在顶级定义（如函数或者类的定义）之前空两行，而方法定义之间空一行。另外，在用于分隔某些功能的位置也可以空一行。

5、通常情况下，运算符两侧、函数参数之间、逗号“,”两侧建议使用空格进行分隔。

当然，大家在学习中，看老师是怎么写代码的，养成和老师一样的习惯就可以了。

PEP8 规范，官网链接：<https://legacy.python.org/dev/peps/pep-0008/>

7、命名规范

命名规范在编写代码中起到很重要的作用，虽然不遵循命名规范，程序也可以运行，但是使用命名规范可以更加直观地了解代码所代表的含义。

命名规范：

1、模块名尽量短小，并且全部使用小写字母，可以使用下划线分隔多个字母。例如：game_main、game_register 都是推荐使用的模块名称。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

- 2、包名进行短小，并且全部使用小写字母，不推荐使用下划线。例如 `com.my.book` 是推荐使用的包名称。而 `com_my_book` 是不推荐的。
- 3、类名采用单词首字母大写形式（即 Pascal 风格）。例如，`Books`。
- 4、模块内部的类采用下划线“`_`”+Pascal 风格的类名组成。例如 `BorrowBook` 类的内部类，可以使用 `_BorrowBook` 命名。
- 5、函数、类的属性和方法的命名规则同模块类似，也是全部使用小写字母，多个字母间用下划线“`_`”分隔。
- 6、常量命名时采用全部大写字母，可以使用下划线。
- 7、使用单下划线“`_`”开头的模块变量或者函数是受保护的，在使用 `from xxx import *` 语句从模块中导入时这些变量或者函数不能被导入。
- 8、使用双下划线“`__`”开头的实例变量或方法是私有的。

第 3 章：变量与基本的数据类型

1、保留字与标识符

1.1、保留字

保留字是 Python 语言中已经被赋予特殊意义的一些单词，开发程序时，不可以把这些保留字作为变量、函数、类、模块和其他对象的名称来使用。

Python 中的保留字有：

`False`、`None`、`True`、`and`、`as`、`assert`、`async`、`await`、`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`。

Python保留字列表

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

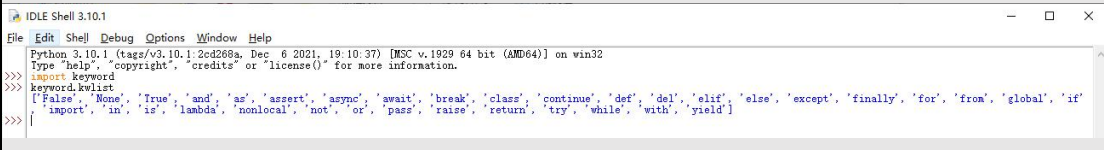
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 1:

两行代码查看 Python 中的保留字:

```
import keyword
keyword.kwlist
```



实例 2: 常见错误

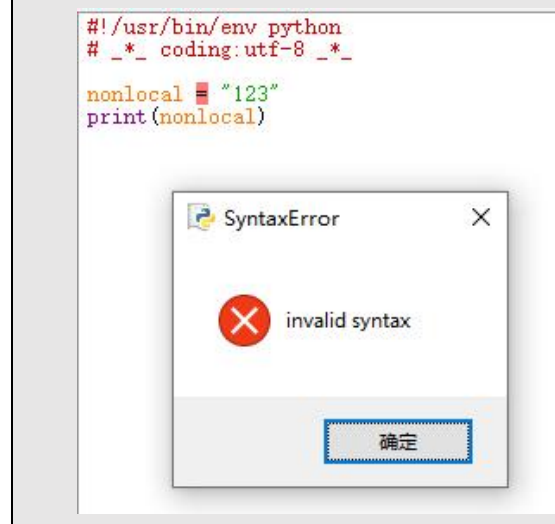
在开发过程中，使用 Python 中的保留字作为模块、类、函数或者变量等的名称，则会提示“invalid syntax”的错误信息。

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

nonlocal = "123"
print(nonlocal)
```

运行结果:



1.2、标识符

标识符可以简单地理解为一个名字，比如说每个人都有自己的名字，它主要是用来标识变量、函数类、模块和其他对象的名称。

Python 语言标识符命名规则如下:

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

- 1、由字母、下划线“_”和数字组成，并且第一个字符不能是数字。目前 Python 中只允许使用 ISO-Latin 字符集中的字符 A~Z 和 a~z。
- 2、不能使用 Python 中的保留字
- 3、区分字母大小写
- 4、Python 中以下划线开头的标识符有特殊意义，一般应避免使用相似的标识符。
 - 以单下划线开头的标识符（如 `_width`）表示不能直接访问的类属性。另外，也不能通过“`from xxx import*`”导入。
 - 以双下划线开头的标识符（如 `__add`）表示类的私有成员。
 - 以算下划线开头和结尾的是 Python 里专用的标识，例如，“`__init__()`”表示构造函数。

实例 1:

下面是合法的标识符:

```
USERID
book
user_id
book01
```

下面是非法的标识符:

```
4word # 以数字开头
class # class 是 python 中的保留字
@book # 不能使用特殊字符
book name # 不能使用空格
```

提示: Python 的标识符中不能包含空格、@、%和\$等特殊字符。

实例 2:

在 Python 中，标识符中的字母是严格区分大小写的，两个同样的单词，如果大小写格式不一样，所代表的意义也是完全不同的。

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

book = 1
Book = 2
BOOK = 3
print(book)
print(Book)
print(BOOK)
```

运行结果:

```
1
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2
3

实例 3:

在 Python 语言中允许使用汉字作为标识符，在程序运行时并不会出现错误，但是建议不要使用汉字作为标识符。

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

我的名字 = "张岩峰"
print(我的名字)
```

运行结果:

张岩峰

2、变量

2.1、什么是变量?

什么是变量?

在 Python 中，变量严格意义上来讲应该被称为“名字”，也可以理解为标签。当把一个值赋给一个名字时，Python 就称为变量。

变量是存储在内存当中的值，这就意味着在创建变量的时候会在内存中开辟一个空间。

基于变量的数据类型，解释器会分配指定内存，并决定什么数据可以被存储在内存中。

因此，变量可以指定不同的数据类型，这些变量可以存储整数，小数或字符。

2.2、什么是常量

常量就是程序在运行过程中，值不能被改变的量，比如咱们每个人的居民身份证号码、数学运算中的圆周率等，这些都是不会发生改变的，它们都可以定义未常量。

在 Python 中，并没有提供定义常量的保留字。不过在 PEP8 规范中规定了常量由大写字母和下划线组成，但是在实际工作中，常量首次被赋值后，还是可以被其他代码所修改。

所以，在 Python 中没有常量。

2.3、变量的定义和使用

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 中，不需要先声明变量名及其类型，直接赋值即可创建各种类型的变量。

变量名定义规则：

- 1、变量名必须是一个有效的标识符。
- 2、变量名不能使用 Python 中的保留字。
- 3、选择有意义的单词作为变量名。

给变量赋值可以通过等号(=)来实现，语法格式为：

单个变量赋值：

变量名 = 值

多个变量赋值：

变量名 1 = 变量名 2 = 变量名 3 = 值

变量名 1, 变量名 2, 变量名 3 = 值 1, 值 2, 值 3

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

number = 123 #整型的变量
print(number) #输出 number 变量的值
print(type(number)) #返回 number 变量类型
myname = "张岩峰"
print(myname)
print(type(myname))
```

运行结果：

```
123
<class 'int'>
张岩峰
<class 'str'>
```

扩展：内置函数 type() 可以返回变量类型。

实例 2：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

num = number = 123 #定义变量 num, number 的值为 123
print(num, number)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
num, number = 123, 456  #定义变量 num 的值为 123, 变量 number 的值为 456
print(num, number)
```

运行结果：

```
123 123
123 456
```

扩展：使用内置函数 `id()` 可以返回变量所指的内存地址

实例 3：

代码如下：

```
num = 123
print (id(num))
```

运行结果：

```
8791284219984
```

3、基本的数据类型

在内存存储的数据可以有很多类型。比如，一个人的姓名可以用字符型存储、年龄可以使用数值型存储、是否结婚可以使用布尔类型存储。这些都是 Python 中提供的基本数据类型。

3.1、数字类型

在开发时，经常使用数字记录游戏的得分、网站的总访问量等信息。在 Python 中，提供了数字类型用于保存这些数据，并且它们是不可改变的数据类型。如果修改数字类型变量的值，那么会先把该值存放到内容中，然后修改变量让其指向新的内存地址。

在 Python 中，数字类型主要包括 `int` 整型、`float` 浮点型、`complex` 复数类型。

● `int` 整型

整数类型，表示没有小数部分的数值。在 Python 中，整数包括正整数、负整数和 0。

实例 1：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
var1 = 1.5 + 1.6
var2 = 1.5 - 1.6
var3 = 1.5 - 2.6
print(int(var1))
print(int(var2))
print(int(var3))
```

运行结果：

```
3
0
-1
```

● float 浮点型

浮点数由整数部分和小数部分组成，主要用于处理包括小数的数。例如：1.451、0.6、-0.154、3.1415926 等。

注意：在使用浮点数进行计算时，可能会出现小数位数不确定情况。如下案例。

实例 2：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var1 = 1.5 + 1.6
var2 = 1.5 - 1.6
var3 = 1.5 - 2.6
print(float(var1))
print(float(var2))
print(float(var3))
```

运行结果：

```
3.1
-0.10000000000000009
-1.1
```

对于这种情况，所有语言都存在这个问题，暂时忽略多余的小数位即可。

● complex 复数类型

在 Python 中复数类型和数学中的复数的形式是一致的。都是由实部和虚部组成，复数由实部（real part）和虚部（imaginary part）构成，并且都是使用 j 或 J 表示虚部。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 3:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = 5 + 6j          # 定义一个变量 var
print(type(var))      # 输出变量的类型
print(id(var))        # 输出变量所指的内存地址
print(var)            # 输出变量 var 的值
```

运行结果:

```
<class 'complex'>
45391376
(5+6j)
```

实例 4: 从复数中提取实部和虚部

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = 5 + 6j
print("复数 var 中的实部为: ", var.real)
print("复数 var 中的虚部为: ", var.imag)
```

运行结果:

```
复数 var 中的实部为:  5.0
复数 var 中的虚部为:  6.0
```

3.2、字符串类型

字符串就是连续的字符序列，可以让计算机所能表示的一切字符的集合。在 Python 中，字符串属于不可变序列，通常使用单引号、双引号、三引号括起来。需要注意的是单引号和双引号中的字符序列必须在一行上，而三引号内的字符序列可以分布在连续的多行上。

实例 1: 定义三个类型的变量，并用 print() 函数输出。

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

title = '我喜欢名言警句!'
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
mot_cn = "命运给予我们的不是失望之酒，而是机会之杯。"  
mot_en = '''Our destiny offers not the cup of despair,  
but the chance of opportunity.'''  
print(title)  
print(mot_cn)  
print(mot_en)
```

运行结果：

```
我喜欢名言警句！  
命运给予我们的不是失望之酒，而是机会之杯。  
Our destiny offers not the cup of despair,  
but the chance of opportunity.
```

提示：字符串开始和结尾使用的引号形式必须一致。另外当需要表示复杂的字符串时，还可以进行引号的嵌套。

● 扩展知识：

Python 中的字符串还支持转义字符。所谓的转义字符是指使用反斜杠“\”对一些特殊字符进行转义。

常见的转移字符及其作用：

```
\: 续行符  
\n: 换行符  
\0: 空  
\t: 水平制表符，用于横向跳到下一制表位  
\': 单引号  
\": 双引号  
\\: 一个反斜杠  
\f: 换页  
\odd: 八进制数，dd 代表的字符，如\012 代表换行  
\xhh: 十六进制数，hh 代表的字符，如\x0a 代表换行
```

实例 2：

代码如下：

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_  
  
title_1 = '万里不惜死，\n朝得成功。'  
title_2 = r'万里不惜死，\n朝得成功。'  
print(title_1)  
print(title_2)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
万里不惜死，  
朝得成功。  
万里不惜死，\n朝得成功。
```

提示：在字符串前面加上字母 r（或 R），那么该字符串将原样输出，其中的转义字符将不进行转义。

3.3、布尔类型

布尔类型主要用来表示真或假的值。在 Python 中，标识符 True 和 False 被解释为布尔值。另外，Python 中的布尔值可以转化为数值，其中 True 表示 1，而 False 表示 0。

实例 1：

代码如下：

```
print(True+1)  
print(False+1)
```

运行结果：

```
2  
1
```

说明：Python 中的布尔值类型可以进行数值运算，例如“False+1”的结果为 1。但是不建议对布尔类型的值进行数值运算。

实例 2：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
x = 1  
y = 2  
if x > y:  
    print('x>y')  
if x < y:  
    print('x<y')
```

运行结果：

```
x<y
```

y 大于 x 为真，y 小于 x 为假。

3.4、数据类型转换

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Python 是动态类型的语言（也称为弱类型语言），虽然不需要先声明变量的类型，但有时仍然需要用到类型转换。

常用类型转换函数：

```
int(x): 将 x 转换成整数类型
float(x): 将 x 转换成浮点数类型
complex(real [, imag]): 创建一个复数
str(x): 将 x 转换为字符串
repr(x): 将 x 转换为表达式字符串
eval(x): 计算在字符串中的有效 Python 表达式，并返回一个对象
chr(x): 将整数 x 转换为一个字符
ord(x): 将一个字符 x 转换为它对应的整数值
hex(x): 将一个整数 x 转换为一个十六进制字符串
oct(x): 将一个整数 x 转换为一个八进制的字符串
bin(x): 将一个整数 x 转换为一个二进制的字符串
```

实例 1：模拟超时抹零结账行为

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

money_all = 56.75 + 63.24 + 24.15 + 17.5
money_all_str = str(money_all)
print("您购买的商品总价格为：", money_all_str)
money_real = int(money_all)
money_real_str = str(money_real)
print("实收金额为：", money_real_str)
```

运行结果：

```
您购买的商品总价格为： 161.64000000000001
实收金额为： 161
```

实例 2：十进制转换为二进制、八进制、十六进制

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

number = int(input("请输入一个十进制的数："))
two = bin(number)[2:]
eight = oct(number)[2:]
sixteen = hex(number)[2:]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(number, "的二进制数为", two, ", 八进制数为", eight, ", 十六进制数为", sixteen)
```

运行结果：

请输入一个十进制的数：10

10 的二进制数为 1010 ， 八进制数为 12 ， 十六进制数为 a

提示：[2:]表示从第 2 位开始取值。

第 4 章：运算符

运算符是一些特殊的符号，主要是用来科学计算、比较大小和逻辑运算等。Python 的运算符主要包括算术运算符、赋值运算符、比较运算符、逻辑运算符、位运算符。

1、算术运算符

算术运算符是处理四则运算的符号，在数字的处理中应用的最多。

算术运算符和对应关系如下：

```
+: 加
-: 减
*: 乘
/: 除
%: 求余，即返回除法的余数
//: 取整除，即返回商的整数部分
**: 幂，即返回 x 的 y 次方
```

在 Python 中进行数学计算时，与我们学过的数学中运算符优先级是一致的。

- 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

算术运算符优先级：

```
第一级：**
第二级：* / % //
第三级：+ -
```

实例 1：算术运算符直接对数字进行运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(3+5)      # 数字 3 与 5 相加
print(3-5)      # 数字 3 与 5 相减
print(3*5)      # 数字 3 与 5 相乘
print(3/5)      # 数字 3 与 5 相除
print(3%5)      # 数字 3 与 5 求余
print(3//5)     # 数字 3 与 5 取整除
print(8//3)     # 数字 8 与 3 取整除
print(3**5)     # 数字 3 的 5 次方
```

运行结果：

```
8
-2
15
0.6
3
0
2
243
```

实例 2：算术运算符对变量进行运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

a=19
b=13
c=4
print(a+b)
print(b-c)
print(a*c)
print(a/c)
print(a%b)
print(b//c)
print(c**b)
```

运行结果：

```
32
9
76
4.75
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
6
3
67108864
```

实例 3:

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

代码如下:

```
print("M"*10)
print(" " *10, "M"*10)
```

运行结果:

```
MMMMMMMMMMMM
                MMMMMMMMMMMM
```

实例 4: 计算学生成绩的分差及平均分

某学员 3 门课程成绩如下:

课程: 分数

Python: 95

Linux: 86

Kubernetes: 74

求 Python 课程和 Kubernetes 课程的分数之差, 3 门课程的平均分。

代码如下:

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

python = 95
linux = 86
kubernetes = 74
sub = python - kubernetes
avg = (python + linux + kubernetes) / 3
print("Python 课程和 Kubernetes 课程的分数之差: ", str(sub), "分")
print("3 门课程的平均分: ", str(avg), "分")
```

运行结果:

```
Python 课程和 Kubernetes 课程的分数之差:  21 分
3 门课程的平均分:  85.0 分
```

2、赋值运算符

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

赋值运算符主要用来为变量等赋值。使用时，可以直接把基本赋值运算符“=”右边的值赋给左边的变量，也可以进行某些运算后再赋值给左边的变量。

常见的赋值运算符说明如下：

=：简单的赋值运算
+=：加赋值
-=：减赋值
*=：乘赋值
/=：除赋值
%=：取余数赋值
**=：幂赋值
//=：取整除赋值

注意：不要混淆“=”和“==”，这两个符号是编程中最常见的错误之一。“=”是赋值运算符，“==”是比较运算符。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

a=19
b=13
c=4
a=a+b    # a+b 的值赋值给 a，此时 a 的值为 32
print(a)
a+=b     # a=a+b，此时 a 的值为 45
print(a)
a-=b     # a=a-b，此时 a 的值为 32
print(a)
a*=b     # a=a*b，此时 a 的值为 416
print(a)
a/=b     # a=a/b，此时 a 的值为 32.0
print(a)
a%=b     # a=a%b，此时 a 的值为 6.0
print(a)
a**=c    # a=a**c，此时 a 的值为 1296.0
print(a)
a//=c    # a=a/c，此时 a 的值为 324.0
print(a)
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
32
45
32
416
32.0
6.0
1296.0
324.0
```

3、比较运算符

比较运算符，也称为关系运算符。用于对变量或表达式的结果进行大小、真假等比较，如果比较的结果为真，则返回 True，如果为假，则返回 False。

Python 的比较运算符

```
>: 大于
<: 小于
==: 等于
!=: 不等于
>=: 大于或等于
<=: 小于或等于
```

在 Python 中，当需要判断一个变量是否介于两个值之间时，可以采用“值 1<变量<值 2”的形式。

实例 1：使用比较运算符比较大小关系

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

python = 97
linux = 75
print("python = " + str(python) + " linux = " + str(linux))
print("python < linux 的结果: " + str(python < linux))
print("python > linux 的结果: " + str(python > linux))
print("python == linux 的结果: " + str(python == linux))
print("python != linux 的结果: " + str(python != linux))
print("python >= linux 的结果: " + str(python >= linux))
print("python <= linux 的结果: " + str(python <= linux))
```

运行结果：

```
python = 97 linux = 75
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
python < linux 的结果: False
python > linux 的结果: True
python == linux 的结果: False
python != linux 的结果: True
python >= linux 的结果: True
python <= linux 的结果: False
```

4、逻辑运算符

假设一家火锅店每周五晚上 10 点至 11 点和每周日晚上 10 点至 11 点,对所有菜品进行打折活动,那么想参加打折活动的顾客,就要在时间上满足这样的条件:周五 22:00~23:00、周日 22:00~23:00。这种情况就需要用到逻辑关系了,在 Python 中也提供了这样的逻辑运算符来进行逻辑运算。

逻辑运算符是对真和假两种布尔值进行运算,运算后的结果仍是一个布尔值,Python 中的逻辑运算符主要包括 and (逻辑与)、or (逻辑或)、not (逻辑非)。

逻辑运算符:

```
and: 逻辑与
or: 逻辑或
not: 逻辑非
```

实例 1: 参加火锅店折扣活动

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("火锅店正在打折, 活动进行中.....")
strWeek = input("请输入您来的星期(如星期一): ")
intTime = int(input("请输入您来的小时(范围: 0~23): "))
if (strWeek == "星期五" and (intTime >= 22 and intTime <= 23)) \
    or (strWeek == "星期日" and (intTime >= 22 and intTime <= 23)):
    print("恭喜您, 获得了折扣活动参与资格, 快快消费吧。")
else:
    print("对不起, 您来晚了, 期待您下次参与。")
```

运行结果:

```
第一次运行:
火锅店正在打折, 活动进行中.....
请输入您来的星期(如星期一): 星期五
请输入您来的小时(范围: 0~23): 23
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

恭喜您，获得了折扣活动参与资格，快快消费吧。

第二次运行：
火锅店正在打折，活动进行中.....
请输入您来的星期（如星期一）：星期二
请输入您来的小时（范围：0~23）：15
对不起，您来晚了，期待您下次参与。

5、位运算符

位运算符是把数字看作二进制数来进行计算的，因此，需要先将要执行运算的数据转换为二进制，然后才能执行运算。Python 中的位运算符有位与（&）、位或（|）、位异或（^）、取反（~）、左移位（<<）和右移位（>>）运算符。

1、位与（&）运算符

“位与”运算的运算符为“&”。

“位与”运算的运算法则是：两个操作数据的二进制表示，只有对应位都是 1 时，结果位才是 1，否则为 0。

如：

12 的二进制数：0000 0000 0000 1100

8 的二进制数： 0000 0000 0000 1000

运算： 0000 0000 0000 1000

实例 1：位与运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("12&8 = "+str(12&8))
```

运行结果：

12&8 = 8

运算过程：将 12 的二进制数和 8 的二进制数进行位与运算，得出的二进制数结果是 1000，换算为十进制就是 8。

2、“位或运算”

“位或”运算的运算符为“|”。

“位或”运算的运算法则是：两个操作数据的二进制表示，只有对应位都是 0，结果位才是 0，否则为 1。

如：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

12 的二进制数：0000 0000 0000 1100
8 的二进制数： 0000 0000 0000 1000
运算： 0000 0000 0000 1100

实例 2：位或运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("12|8 = "+str(12|8))
```

运行结果：

12|8 = 12

3、“位异或”运算

“位异或”运算的运算符是“^”。

“位异或”运算的运算法则是：当两个操作数的二进制表示相同（同时为 0 或同时为 1）时，结果位才是 0，否则为 1。

如：

12 的二进制数：0000 0000 0000 1100
8 的二进制数： 0000 0000 0000 1000
运算： 0000 0000 0000 0100

实例 3：位异或运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("12^8 = "+str(12^8))
```

运行结果：

12^8 = 4

4、“位取反”运算

“位取反”运算也称为“位非”运算，运算符为“~”。

如： $\sim x = -(x+1)$ ；

实例 4：位取反运算

代码如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("~12 = "+str(~12))
```

运行结果：

```
~12 = -13
```

5、左移位运算符<<

左移位运算符<<是将一个二进制操作数向左移动指定的位数，左边溢出的位被丢弃，右边的空位用 0 补充。

如：

12 的二进制数：0000 0000 0000 1100

左移位 2 位： 0000 0000 0011 0000，即 48

实例 5：左移位运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("12 << = "+str(12<<2))
```

运行结果：

```
12 << = 48
```

6、右移位运算符>>

右移位运算符>>是将一个二进制操作数向右移动指定位数，右边溢出的位被丢弃，左边的空位用 0 补充。

如：

12 的二进制数：0000 0000 0000 1100

右移位 2 位： 0000 0000 0000 0011，即 3

实例 6：右移位运算

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("12 >> = "+str(12>>2))
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
12 >> = 3
```

6、运算符的优先级

所谓运算符的优先级，是指在应用中哪一个运算符先计算，哪一个后计算，与数学的四则运算应遵循的“先乘除，后加减”是一个道理。

Python 运算符的运算规则是：优先级高的运算先执行，优先级低的运算后执行，同一优先级的操作按照从左到右的顺序进行。也可以像四则运算那样使用小括号，括号内的运算最先执行。

运算符的优先级：

```
**： 幂
~、+、-： 取反、正号和负号
*、/、%、//： 算术运算符
+、-： 算术运算符
<<、>>： 位运算符中的左移和右移
&： 位运算符中的位与
^： 位运算符中的位异或
|： 位运算符中的位或
<、<=、>、>=、!=、==、： 比较运算符
```

提示：在编写程序时尽量使用括号“()”来限定运算次序，以免运算次序发生错误。

第 5 章：列表和元组

1、序列

序列是一块用于存放多个值的连续内存空间，并且按一定顺序排列，每一个值（称为元素）都分配一个数字，称为索引或位置。通过该索引可以取出相应的值。

在 Python 中，序列结构主要有列表、元组、集合、字典和字符串。

1.1、索引

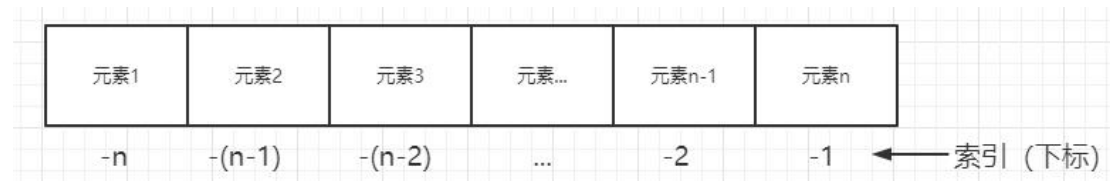
序列中的每一个元素都有一个编号，也称为索引。这个索引是从 0 开始递增的，即下标为 0 标识第 1 个元素，下标为 1 标识第 2 个元素，以此类推。如下图所示“序列的正数索引”：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



Python 比较神奇，它的索引可以是负数。这个索引从右向左计数，也就是从最后一个元素开始计数，即最后一个元素的索引值是-1，倒数第二个元素的索引值是-2，以此类推。如下图表示“序列的负数索引”：



注意：在采用负数作为索引值时，是从-1 开始的，而不是从 0 开始的，即最后一个元素的下标为-1，这是为了防止与第一个元素重合。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

ver = ["玉不啄", "不成器", "人不学", "不知义", "《礼记》"]
print(ver[2])    # 输出第 3 个元素
print(ver[-1])   # 输出最后一个元素
```

运行结果：

```
人不学
《礼记》
```

1.2、切片

切片操作是访问序列中元素的另一种方法，它可以访问一定范围内的元素。通过切片操作可以生成一个新的序列。

切片操作的语法格式如下：

```
sname[start : end : step]
```

参数说明：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

- sname: 表示序列的名称
- start: 表示切片的开始位置（包括该位置），如果不指定，则默认为 0。
- end: 表示切片的截止位置（不包括该位置），如果不指定则默认为序列的长度。
- step: 表示切片的步长，如果省略，则默认为 1，当省略该步长时，最后一个冒号也可以省略。

实例 1:

通过切片获取列表中的第 2 个到第 5 个元素，以及获取第 1 个、第 3 个、第 5 个元素。输出整个序列

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

ver = ["人之初", "性本善", "性相近", "习相远",
       "苟不教", "性乃迁", "教之道", "贵以专",
       "昔孟母", "择邻处", "子不学", "断机杼"]
print(ver[1:5])
print(ver[0:5:2])
print(ver[:])
```

运行结果:

```
['性本善', '性相近', '习相远', '苟不教']
['人之初', '性相近', '苟不教']
['人之初', '性本善', '性相近', '习相远', '苟不教', '性乃迁', '教之道', '贵以专', '昔孟母', '择邻处', '子不学', '断机杼']
```

提示:

如果想要复制整个序列，可以将 start 和 end 参数都省略，但是中间的冒号需要保留。

1.3、序列相加

在 Python 中，支持两种相同类型的序列相加操作。即将两个序列进行连接，使用加 (+) 运算符实现。

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
ver1 = ["人之初", "性本善", "性相近", "习相远",  
        "苟不教", "性乃迁", "教之道", "贵以专"]  
ver2 = ["昔孟母", "择邻处", "子不学", "断机杼"]  
print(ver1 + ver2)
```

运行结果：

```
['人之初', '性本善', '性相近', '习相远', '苟不教', '性乃迁', '教之道',  
'贵以专', '昔孟母', '择邻处', '子不学', '断机杼']
```

实例 2:

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
ver1 = ["人之初", "性本善", "性相近", "习相远",  
        "苟不教", "性乃迁", "教之道", "贵以专"]  
ver2 = ["1", "2", "3", "4"]  
print(ver1 + ver2[1:3])
```

运行结果：

```
['人之初', '性本善', '性相近', '习相远', '苟不教', '性乃迁', '教之道',  
'贵以专', '2', '3']
```

注意：在进行序列相加时，相同类型的序列是指，同为列表、元组或集合等，序列中的元素类型可以不同。

1.4、乘法

在 Python 中，使用数字 n 乘以一个序列会生成新的序列。新序列的内容为原来序列被重复 n 次的结果。

实例 1:

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
ver = ["centos", "ubuntu", "redhat"]  
print(ver * 2)
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
['centos', 'ubuntu', 'redhat', 'centos', 'ubuntu', 'redhat']
```

在进行序列的乘法运算时，还可以实现初始化指定长度列表的功能。如下面的“实例 2”的代码。

实例 2:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

ver = ["centos", "ubuntu", "redhat"]*2
print(ver)
```

运行结果:

```
['centos', 'ubuntu', 'redhat', 'centos', 'ubuntu', 'redhat']
```

1.5、检查某个元素是否是序列的成员

在 Python 中，可以使用 in 关键字检查某个元素是否是序列的成员，即检查某个元素是否包含在该序列中。

语法格式如下:

```
value in sequence
```

value: 表示要检查的元素。

sequence: 标识指定的序列。

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

ver = ["centos", "ubuntu", "redhat"]
print("redhat" in ver)
print("red" in ver)
```

运行结果:

```
True
False
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

扩展：在 Python 中，也可以使用 not in 关键字实现检查某个元素是否不包含在指定

1.6、计算序列的长度、最大值和最小值

在 Python 中，提供了内置函数计算序列的长度、最大值和最小值。

内置函数分别是：

len() 函数计算序列的长度。

max() 函数返回序列中的最大元素。

min() 函数返回序列中的最小元素。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

ver = ["56", "52", "74", "81", "25", "43", "52", "68"]
print("序列的长度为：", len(ver))
print("序列的最大元素为：", max(ver))
print("序列的最小元素为：", min(ver))
```

运行结果：

```
序列的长度为： 8
序列的最大元素为： 81
序列的最小元素为： 25
```

扩展：除了上面的这 3 个内置函数，Python 还提供了如下所示的内置函数

```
list(): 将序列转换为列表
str(): 将序列转换为字符串
sum(): 计算元素和
sorted(): 对元素进行排序
reversed(): 反向序列中的元素
enumerate(): 将序列组合为一个索引序列，多用在 for 循环中
```

2、列表

大家应该都听音乐吧，那么想必歌曲列表大家一定很熟悉。如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



Python 中的列表和歌曲列表类似，也是由一系列按特定顺序排列的元素组成的。它是 Python 中内置的可变序列。

如上图，则是 QQ 音乐的歌曲列表，这里就是用的列表。

2.2.1、列表的创建和删除

在 Python 中提供了多种创建列表的方法。下面分别进行介绍：

1、使用赋值运算符直接创建列表

同其他类型的 Python 变量一样，创建列表时，也可以使用赋值运算符“=”直接将一个列表赋值给变量。

语法格式为：

```
listname = [element 1,element 2,element 3,...,element n]
```

listname：表示列表的名称

element：表示列表中的元素，个数没有限制

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

number = [7, 15, 65, 25, 48, 36, 71, 82, 95, 12]
linux = ['readhat', 'centos', 28, [1, 2, 3, 4, 5]]
print(number)
print(linux)
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[7, 15, 65, 25, 48, 36, 71, 82, 95, 12]  
['readhat', 'centos', 28, [1, 2, 3, 4, 5]]
```

提示：在使用列表时，可以将不同类型的数据放入到同一个列表中，但是通常情况下，不建议这么做，而是在一个列表中只放入一种类型的数据，这样可以提高程序的可读性。

2、创建空列表

创建空列表可以直接使用“列表名 = []空列表”来进行定义，如下实例 2。

实例 2：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
number = []  
print(number)
```

运行结果：

```
[]
```

3、创建数值列表

在 Python 中，数值列表很常用。比如说，在考试系统中记录学生的成绩，或者在游戏中记录每个角色的位置，各个玩家的得分情况等都可以使用数值列表。在 Python 中，可以使用 `list()` 函数直接将 `range()` 函数循环出来的结果转换为列表。

`list()` 函数的基本语法如下：

```
list(data)
```

`data`：表示可以转换为列表的数据，类型可以是 `range` 对象、字符串、元组或者其他可迭代类型的数据。

实例 3：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
print(list(range(10, 20, 2)))
```

运行结果：

```
[10, 12, 14, 16, 18]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

提示：使用 `list()` 函数时，不仅能通过 `range` 对象创建列表，还可以通过其他对象创建列表。

4、删除列表

对于已经创建的列表，可以使用 `del` 语句删除。

语法如下：

```
del listname
```

`listname`：表示列表名称

提示：`del` 语句在开发中，并不常用。因为 Python 自带的垃圾回收机制会自动销毁不用的列表，所以即使我们不手动将其删除，Python 也会自动将其回收。

实例 4：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = [1, 2, 3, 4, 5]
print(var)
del var
print(var)
```

运行结果：

```
[1, 2, 3, 4, 5]
Traceback (most recent call last):
  File "K:\soft\1\2.py", line 7, in <module>
    print(var)
NameError: name 'var' is not defined. Did you mean: 'vars'?
```

提示：这里已经把 `var` 列表给删除了，所以 `print` 是无法输出的，所以就报错了，没有找到名为 `var` 的列表。

2.2.2、访问列表元素

访问列表元素可以直接使用 `print()` 函数。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
var = [1, 2, 3, 4, 5, ["ubuntu", "linux", "redhat"]]  
print(var)  
print(var[5])  #输出下标为 5 的元素
```

运行结果：

```
[1, 2, 3, 4, 5, ['ubuntu', 'linux', 'redhat']]  
['ubuntu', 'linux', 'redhat']
```

2.2.3、遍历列表

遍历列表中的所有元素是常用的一种操作，在遍历的过程中可以完成查询、处理等功能。

在生活中，如果想要去商场买一件衣服，就需要在商城中逛一遍，看是否有想要的衣服，逛商城的过程就相当于列表的遍历操作。

在 Python 中遍历列表的方法有多种，下面介绍两种常用的方法。

1、直接使用 for 循环实现

直接使用 for 循环遍历列表，只能输出元素的值。语法如下：

```
for item in listname:  
    print(item)
```

参数说明：

item：表示用于保存获取到的元素值，要输出元素内容时，直接输出该变量即可。

listname：表示为列表名称。

实例 1：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
print("常用的 Linux 系统有：")  
var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]  
for i in var:  
    print(i)
```

运行结果：

```
常用的 Linux 系统有：  
ubuntu  
linux
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
redhat
Debian
红旗 Linux
```

2、使用 for 循环和 enumerate() 函数实现

使用 for 循环和 enumerate() 函数可以实现同时输出索引值和元素内容的功能。语法格式如下：

```
for index,item in enumerate(listname):
    print(index + 1,item)
```

参数说明：

index：用于保存元素的索引。

item：用于保存获取到的元素值，要输出元素内容时，直接输出该变量即可。

listname：表示为列表名称。

实例 2：模仿如下歌单输出



代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

print("听过的 3:")
var = ["Infiniy", "Youngio(抖音版)", "Let Me Down Slowly", "时光背面的我", "自娱自乐(完整正式版)"]
for index,item in enumerate(var):
    print(index + 1,item)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
听过的 3:
1 Infiniy
2 Youngio(抖音版)
3 Let Me Down Slowly
4 时光背面的我
5 自娱自乐(完整正式版)
```

2.2.4、添加、修改和删除列表元素

添加、修改和删除列表元素也称为更新列表。在开发中，一个列表很少是写死的，要经常对列表进行更新。

1、添加元素

在讲序列的时候，咱们讲过序列的相加“+”，通过这种方法也可以实现列表元素的添加。但是这种方法的执行速度要比直接使用列表对象 `append()` 方法慢，所以建议在实现添加元素时，使用列表对象的 `append()` 方法实现。

列表对象 `append()` 方法用于在列表的末尾追加元素，语法格式如下：

```
listname.append(index)
```

参数说明：

`listname`：为要添加元素的列表名称

`index`：为要添加到列表末尾的元素

实例 1：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]
print(len(var))
var.append("ESXI")
print(len(var))
print(var)
```

运行结果：

```
5
6
['ubuntu', 'linux', 'redhat', 'Debian', '红旗 Linux', 'ESXI']
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

补充 1:

列表对象除了提供 `append()` 方法向列表中添加元素，还提供了 `insert` 向列表中添加元素。该方法用于向列表的指定位置插入元素。

注意：`insert()` 这种方法执行效率没有 `append()` 方法高。

语法格式:

```
listname.insert(index, obj)
```

参数说明:

listname: 列表名

index: 对象 obj 需要插入的索引位置。

obj: 要插入列表中的对象。

实例 2:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]
var.insert(1, "ESXI") #插入到索引为 1 的位置处
print(var)
```

运行结果:

```
['ubuntu', 'ESXI', 'linux', 'redhat', 'Debian', '红旗 Linux']
```

补充 2:

`insert()` 表示向列表中添加一个元素,如果想要将一个列表中的全部元素添加到另一个列表中,可以使用列表对象的 `extend()` 方法实现。

语法格式:

```
listname.extend(obj)
```

参数说明:

listname: 列表名

obj: 为要添加的列表

提示: 语句执行后, obj 的内容将追加到 listname 的后面。

实例 3:

代码如下:

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# -*- coding:utf-8 -*-  
  
var1 = ["ubuntu","linux"]  
var2 = ["redhat","Debian","红旗 Linux"]  
var1.extend(var2)  
print(var1)
```

运行结果：

```
['ubuntu', 'linux', 'redhat', 'Debian', '红旗 Linux']
```

2、修改元素

修改列表中的元素只需要通过索引获取该元素，然后再重新赋值即可。

实例 4：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
var = ["ubuntu","linux","redhat","Debian","红旗 Linux"]  
print(var)  
var[2] = "centos"  
print(var)
```

运行结果：

```
['ubuntu', 'linux', 'redhat', 'Debian', '红旗 Linux']  
['ubuntu', 'linux', 'centos', 'Debian', '红旗 Linux']
```

3、删除元素

删除元素主要有以下两种方法：

- 1、指定索引删除
- 2、指定元素值进行删除

● 指定索引删除

删除列表中的指定元素和删除列表类似，也可以使用 del 语句实现。所不同的是在指定列表名称时，换为列表元素。

实例 5：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]
print(var)
del var[2]
print(var)
```

运行结果：

```
['ubuntu', 'linux', 'redhat', 'Debian', '红旗 Linux']
['ubuntu', 'linux', 'Debian', '红旗 Linux']
```

● 指定元素值进行删除

如果想要删除一个不确定其位置的元素（即根据元素值删除），可以使用列表对象的 `remove()` 方法实现。

实例 6：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]
print(var)
var.remove("Debian")
print(var)
```

运行结果：

```
['ubuntu', 'linux', 'redhat', 'Debian', '红旗 Linux']
['ubuntu', 'linux', 'redhat', '红旗 Linux']
```

提示：如果指定删除的元素不存在，则会报如下错误：

```
Traceback (most recent call last):
  File "K:\soft\1\2.py", line 6, in <module>
    var.remove("Debian1")
ValueError: list.remove(x): x not in list
```

扩展：指定删除元素值，结合 `if` 判断使用

实例 7：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux"]
del_var = "Debian"
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
if var.count(del_var)>0:
    var.remove(del_var)
print(var)
```

运行结果：

```
['ubuntu', 'linux', 'redhat', '红旗 Linux']
```

说明：列表对象的 count() 方法用于判断指定元素出现的次数，返回结果为 0 时，表示不存在该元素。

2.2.5、对列表进行统计计算

Python 的列表提供了内置的一些函数来实现统计、计算方面的功能。

1、获取指定元素出现的次数

使用列表对象的 count() 方法可以获取指定元素在列表中的出现次数。

语法格式如下：

```
listname.count(obj)
```

参数说明：

listname：列表的名称

count：获取指定元素出现的次数，方法函数。

obj：表示要判断是否存在的对象，这里只能进行精确匹配，即不能说元素值的一部分。

返回值：元素在列表中出现的次数。

实例 1：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux", "Debian"]
num = var.count("Debian")
print(num)
```

运行结果：

```
2
```

2、获取指定元素首次出现的下标

使用列表对象的 index() 方法可以获取指定元素在列表中首次出现的位置（索引）。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

语法格式如下：

```
listname.index(obj)
```

参数说明：

listname：列表的名称

index：获取指定元素第一次出现的下标，方法函数。

obj：表示要查找的对象，这里只能进行精确匹配。

返回值：首次出现的索引值。

实例 2：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = ["ubuntu", "linux", "redhat", "Debian", "红旗 Linux", "Debian"]
num = var.index("Debian")
print(num)
```

运行结果：

3

3、统计数值列表的元素和

在 Python 中，提供了 sum() 函数用于统计数值列表中各元素的和。

语法格式如下：

```
sum(listname)
```

参数说明：

sum：统计指定列表的元素和

listname：列表的名称

实例 3：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = [95, 98, 92, 94, 91, 99, 99, 100, 84, 95]
num = sum(var)
print(num)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

947

2.2.6、对列表进行排序

在开发工作中，我们经常需要用到对列表进行排序。

在 Python 中提供了两种常用的对列表进行排序的方法：使用列表对象的 `sort()` 方法和使用内置的 `sorted()` 函数。

1、使用列表对象的 `sort()` 方法实现

列表对象提供了 `sort()` 方法用于对原列表中的元素进行排序。排序后的原列表中的元素顺序将发生改变。

列表对象的 `sort()` 方法的语法格式如下：

```
listname.sort(key=None, reverse=False)
```

参数解释：

`listname`：表示要进行排序的列表。

`key`：表示指定一个从每个列表元素中提取一个用于比较的键（例如，设置“`key=str.lower`”表示在排序时不区分字母大小写”）。

`reverse`：可选参数，如果将其值指定为 `True`，则表示降序排列，如果为 `False`，则表示升序排序。默认为升序排序。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = [95, 98, 92, 94, 91, 99, 99, 100, 84, 95]
print("原列表：", var)
var.sort()
print("升序：", var)
var.sort(reverse=True)
print("降序：", var)
```

运行结果：

```
原列表： [95, 98, 92, 94, 91, 99, 99, 100, 84, 95]
升序：    [84, 91, 92, 94, 95, 95, 98, 99, 99, 100]
降序：    [100, 99, 99, 98, 95, 95, 94, 92, 91, 84]
```

实例 2：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = ['Linux', 'centos', 'Ubuntu', 'unix']
var.sort()
print("区分字母大小写：", var)
var.sort(key=str.lower)
print("不区分字母大小写：", var)
```

运行结果：

```
区分字母大小写： ['Linux', 'Ubuntu', 'centos', 'unix']
不区分字母大小写： ['centos', 'Linux', 'Ubuntu', 'unix']
```

提示：使用 `sort()` 方法对字符串列表进行排序时，采用的规则是先对大写字母进行排序，然后再对小写字母进行排序。

注意：`sort()` 方法对列表进行排序时，对于中文的支持不好。不建议使用 `sort()` 对中文进行排序。

2、使用内置的 `sorted()` 函数实现

在 Python 中，提供了一个内置的 `sorted()` 函数，用于对列表进行排序。使用该函数进行排序后，原列表的元素顺序不变。

`sorted()` 函数的语法格式如下：

```
sorted(iterable, key=None, reverse=False)
```

参数解释：

`iterable`：表示要进行排序的列表名称。

`key`：表示指定一个从每个列表元素中提取一个用于比较的键（例如，设置“`key=str.lower` 表示在排序时不区分字母大小写”）。

`reverse`：可选参数，如果将其值指定为 `True`，则表示降序排列，如果为 `False`，则表示升序排序。默认为升序排序。

实例 3：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

var = [95, 98, 92, 94, 91, 99, 99, 100, 84, 95]
var_as = sorted(var)
print("升序：", var_as)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
var_des = sorted(var,reverse = True)
print("降序：",var_des)
print("原列表：",var)
```

运行结果：

```
升序： [84, 91, 92, 94, 95, 95, 98, 99, 99, 100]
降序： [100, 99, 99, 98, 95, 95, 94, 92, 91, 84]
原列表： [95, 98, 92, 94, 91, 99, 99, 100, 84, 95]
```

说明：列表对象的 `sort()` 方法和内置 `sorted()` 函数的作用基本相同，所不同的就是使用 `sort()` 方法时，会改变原列表的元素排列顺序，而使用 `sorted()` 函数时，会建立一个原列表的副本，该副本为排序后的列表。

2.2.7、列表推导式

使用列表推导式可以快速生成一个列表，或者根据某个列生成满足指定需求的列表。列表推导式通常有以下几种常用的语法格式。

1、生成指定范围的数值列表

语法格式如下：

```
list = [Expression for var in range]
```

参数说明：

list：表示生成的列表名称

Expression：表达式，用于计算新列表的元素。

var：循环变量

range：采用 `range()` 函数生成的 range 对象

实例 1：生成一个包括 10 个随机数的列表，要求数的范围在 10~100（包括）之间。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
num = [random.randint(10,100) for i in range(10)]
print("生成的随机数为：",num)
```

运行结果：

```
生成的随机数为： [21, 69, 31, 27, 73, 73, 64, 95, 58, 81]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2、根据列表生成指定需求的列表
语法格式如下：

```
newlist = [Expression for var in list]
```

参数说明：

newlist：表示新生成的列表名称

Expression：表达式，用于计算新列表的元素。

var：变量，值后面列表的每个元素值。

list：用于生成新列表的原列表

实例 2：定义一个记录商品价格的列表，然后应用列表推导式生成一个将全部商品价格打五折的列表。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

price = [1200, 5330, 2988, 6345, 7777, 5264]
sale = [float(i*0.5) for i in price]
print("原价格：", price)
print("打五折的价格：", sale)
```

运行结果：

```
原价格： [1200, 5330, 2988, 6345, 7777, 5264]
打五折的价格： [600.0, 2665.0, 1494.0, 3172.5, 3888.5, 2632.0]
```

3、从列表中选择符合条件的元素组成新的列表
语法格式如下：

```
newlist = [Expression for var in list if condition]
```

参数说明：

newlist：表示新生成的列表名称

Expression：表达式，用于计算新列表的元素。

var：变量，值后面列表的每个元素值。

list：用于生成新列表的原列表。

condition：条件表达式，用于指定筛选条件。

实例 3：定义一个记录商品价格的列表，然后应用列表推导式生成一个商品价格高于 5000 的列表。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
price = [1200, 5330, 2988, 3345, 7777, 5264]
sale = [i for i in price if i>5000]
print("原列表：", price)
print("价格高于 500 的：", sale)
```

运行结果：

```
原列表： [1200, 5330, 2988, 3345, 7777, 5264]
价格高于 500 的： [5330, 7777, 5264]
```

3、元组

元组 (tuple) 是 Python 中另一个重要的序列结构，与列表类似，也是由一系列按特定顺序排列的元素组成，但是它是不可变序列。因此，元组也可以称为不可变的列表。

提示：

从元组和列表的定义上看，这两种结构比较相似，那么它们之间有哪些区别呢？

它们之间的主要区别就是元组是不可变序列，列表是可变序列。即元组中的元素不可以被程序所单独修改，而列表则可以被程序所任意修改。

3.1、元组的创建和删除

在 Python 中提供了多种创建元组的方法。下面分别进行介绍：

1、使用赋值运算符直接创建元组

同其他类型的 Python 变量一样，创建元组时，也可以使用赋值运算符“=”直接将一个元组赋值给变量。

语法格式如下：

```
tuplename = (element 1, element 2, element 3, ..., element n)
```

参数说明：

tuplename：表示元组的名称，可以是任何 Python 命名规则的标识符。

element 1, element 2, element 3, ..., element n：表示元组中的元素，个数没有限制。

注意：创建元组的语法与创建列表的语法类似，只是创建列表时使用的是“[]”，而创建元组时使用的是“()”。

实例 1：定义元组

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

temp1 = (7, 12, 52, 61, 15, 35)
temp2 = ("红帽", "乌班图", "红旗")
temp3 = ('Linux', 21, ("人生苦短", "我用 Pyhon"))
print(temp1)
print(temp2)
print(temp3)
```

运行结果：

```
(7, 12, 52, 61, 15, 35)
('红帽', '乌班图', '红旗')
('Linux', 21, ('人生苦短', '我用 Pyhon'))
```

2、创建空元组

在 Python 中，也可以创建空元组。实例如下

实例 2：定义空元组

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

temp1 = ()
```

3、创建数值元组

在 Python 中，可以使用 tuple() 函数直接将 range() 函数循环出来的结果转换为数值元组。

tuple() 函数的语法格式如下：

```
tuple(data)
```

参数说明：

data：表示可以转换为元组的数据，类型可以是 range 对象、字符串、元组或者其他可迭代类型的数据。

实例 3：创建一个 10~20 之间（不包括 20）所有偶数的元组。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
templ = tuple(range(10, 20, 2))
print(templ)
```

运行结果：

```
(10, 12, 14, 16, 18)
```

4、删除元组

对于已经创建的元组，不再使用时，可以使用 del 语句将其删除。

语法格式如下：

```
del tuplename
```

参数说明：

tuplename：表示为要删除元组的名称。

提示：在实际开发工作中，del 语句不常用。因为 Python 自带的垃圾回收机制会自动销毁不用的元组，所以即使我们不手动删除，Python 也会自动将其回收。

实例 4：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

templ = tuple(range(10, 20, 2))
del templ
print(templ)
```

运行结果：

```
Traceback (most recent call last):
  File "I:\soft\1\2.py", line 6, in <module>
    print(templ)
NameError: name 'templ' is not defined
```

3.2、访问元组元素

访问元组元素和访问列表元素使用的函数一样，都是 print() 函数。

实例 1：

代码如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

temp = ('Linux', 21, ("人生苦短", "我用 Pyhon"))
print(temp)
```

运行结果：

```
('Linux', 21, ('人生苦短', '我用 Pyhon'))
```

同样的，元组也可以使用 for 循环进行遍历。通 2.2.3 节。

3.3、修改元组元素

元组是不可变序列，所以不能对它的单个元素值进行修改。但是元组也不是完全不能修改。我们可以对元组进行重新赋值。

实例 1：对元组重新赋值

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

temp = ('Centos', 'Ubuntu', 'Redhat', '红旗 Linux', "UNIX")
temp = ('Centos', 'Ubuntu', 'Debian', '红旗 Linux', "UNIX")
print("新元组", temp)
```

运行结果：

```
新元组 ('Centos', 'Ubuntu', 'Debian', '红旗 Linux', 'UNIX')
```

实例 2：对元组进行连接组合

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

temp1 = ('Centos', 'Ubuntu', 'Redhat')
print("原元组", temp1)
temp2 = temp1 + ('Debian', '红旗 Linux', "UNIX")
print("新元组", temp2)
```

运行结果：

```
原元组 ('Centos', 'Ubuntu', 'Redhat')
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
新元组 ('Centos', 'Ubuntu', 'Redhat', 'Debian', '红旗 Linux', 'UNIX')
```

注意：在进行元组连接时，连接的内容必须都是元组。不能将元组和字符串或者列表进行连接。

3.4、元组推导式

使用元组推导式可以快速生成一个元组，它的表现形式和列表推导式类似，只是将列表推导式中的中括号“[]”修改为小括号“()”。

实例 1：生成一个包含 10 个随机数的元组

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
num = (random.randint(10,100) for i in range(10))
print("生成的元组为：", num)
```

运行结果：

生成的元组为： <generator object <genexpr> at 0x0000024CCB7E3530>

从上面的执行结果中，可以看出使用元组推导式生成的结果并不是一个元组或者列表，而是一个生成器对象，这一点和列表推导式是不同的，需要使用该生成器对象可以将其转换为元组或者列表。其中，转换为元组需要使用 tuple() 函数，而转换为列表则需要使用 list() 函数。

实例 2：生成一个包含 10 个随机数的元组，然后将其转换为元组并输出。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
num = (random.randint(10,100) for i in range(10))
num = tuple(num)
print("转换后的元组为：", num)
```

运行结果：

转换后的元组为： (73, 91, 77, 36, 48, 91, 82, 31, 16, 76)

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

要使用通过元组推到器生成的生成器对象，还可以直接使用 for 循环遍历或者直接使用 `__next__()` 方法进行遍历。

实例 3：通过生成器推导式生成一个包含 3 个元素的生成器对象 num，然后调用 3 次 `__next__()` 方法输出每个元素，再将生成器对象 num 转换为元组输出。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

num = (i for i in range(3))
print(num.__next__())
print(num.__next__())
print(num.__next__())
num = tuple(num)
print("转换后：", num)
```

运行结果：

```
0
1
2
转换后： ()
```

实例 4：通过生成器推导式生成一个包括 4 个元素的生成器对象 num，然后应用 for 循环遍历该生成器对象，并输出每一个元素的值，最后再将其转换未元组输出。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

num = (i for i in range(4))
for i in num:
    print(i, end=" ")
print("转换后：", tuple(num))
```

运行结果：

```
0 1 2 3 转换后： ()
```

从实例 3、4 两个案例中可以看出，无论通过哪种方式遍历后，如果再想使用生成器对象，都必须重新创建一个生成器对象，因为遍历后原生成器对象已经不存在了。

3.5、元组与列表的区别

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

元组和列表都属于序列，而且它们又都可以按照特定顺序存放一组元素，类型又不受限制，只要是 Python 支持的类型都可以。那么它们之间有什么区别呢？

列表类似于我们用铅笔在纸上写下自己喜欢的歌词，写错了还可以擦掉。而元组则类似于用钢笔写下的歌词，写上了就擦不掉了，除非换一张纸重新写。

列表和元组的主要区别主要体现在以下几个方面：

- 列表属于可变序列，它的元素可以随时修改或者删除。而元组属于不可变序列，其中的元素不可以修改，除非整体替换。

- 列表可以使用 `append()`、`extend()`、`insert()`、`remove()` 和 `pop()` 等方法实现添加和修改列表元素。而元组则没有这几个方法，因为不能向元组中添加和修改元素，同样也不能删除元素。

- 列表可以使用切片访问和修改列表中的元素。元组也支持切片，但是它只支持通过切片访问元组中的元素，不支持修改。

- 元组比列表的访问和处理速度快。所以如果只需要对其中的元素进行访问，而不进行任何修改，建议使用元组。

- 列表不能作为字典的键，而元组则可以。

第 6 章：字符串与正则表达式

字符串是几乎所有编程语言在项目开发过程中涉及最多的一块内容。其实，“在开发一个项目，基本上就是在不断地处理字符串”。

1、字符串常用操作

1.1、拼接字符串

使用“+”运算符可完成对多个字符串的拼接，“+”运算符可以连接多个字符串并产生一个字符串对象。

实例 1：拼接中文+英文字符串

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

en = "If you don't succeed, you swear!"
cn = "不达成功誓不休!"
print(en + '----' + cn)
```

运行结果：

```
If you don't succeed, you swear!----不达成功誓不休!
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

字符串不允许直接与其他类型的数据拼接，例如如下实例 2：“将字符串和数值拼接在一起”

实例 2:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "我的钱包余额"
num = 15369
str2 = "元"
print(str1 + num + str2)
```

运行结果:

```
Traceback (most recent call last):
  File "I:\soft\1\2.py", line 7, in <module>
    print(str1 + num + str2)
TypeError: can only concatenate str (not "int") to str
```

问题解决:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "我的钱包余额"
num = 15369
str2 = "元"
print(str1 + str(num) + str2)
```

运行结果:

我的钱包余额 15369 元

1.2、计算字符串的长度

由于不同的字符串所占字节数不同，所以要计算字符串的长度，需要先了解各字符所占的字节数。在 Python 中，数字、英文、小数点、下划线和空格占一个字节。一个汉字可能会占 2~4 个字节，占几个字节取决于采用的编码。汉字在 GBK/GB2312 编码中占 2 个字节，在 UTF-8/Unicode 中一般占用 3 个字节（或 4 个字节）。

在 Python 中，提供了 len() 函数计算字符串的长度。语法格式如下:

```
len(string)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

string: 用于知道要进行长度统计的字符串

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "Kubernetes 是最流行的容器编排集群!"
str2 = len(str1)
print(str2)
```

运行结果:

22

从运行结果中可以看出，在默认情况下，通过 len() 函数计算字符串的长度时，不区分英文、数字和汉字，所有字符串都认为是一个。

在实际开发中，有时需要获取字符串实际所占的字节数，即如果采用 UTF-8 编码，汉字占 3 个字节，采用 GBK 时，汉字占 2 个字节。这时，可以通过使用 encode() 方法进行编码后再进行获取。

实例 2: 获取采用 UTF-8 编码的字符串的长度

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "Kubernetes 是最流行的容器编排集群!"
str2 = len(str1.encode())
print(str2)
```

运行结果:

46

上面的运行结果是 46，英文占 10 个字节，11 个汉字和 1 个中文字符占 36 个字节。

实例 3: 获取采用 GBK 编码的字符串的长度

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "Kubernetes 是最流行的容器编排集群!"
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
str2 = len(str1.encode('gbk'))  
print(str2)
```

运行结果：

34

上面的运行结果是 34，英文占 10 个字节，11 个汉字和 1 个中文字符占 24 个字节。

提示：

UTF-8：中文和中文字符占 3 个字节。英文和英文字符占 1 个字节。

GBK：中文和中文字符占 2 个字节。英文和英文字符占 1 个字节。

1.3、截取字符串

由于字符串也属于序列，所以要截取字符串，可以采用切片方法实现。

通过切片方法截取字符串的语法格式如下：

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

参数说明：

- string：表示要截取的字符串。
- start：表示要截取的第一个字符的索引（包括该字符），如果不指定，则默认为 0。
- end：表示要截取的最后一个字符的索引（不包括该字符），如果不指定则默认认为字符串的长度。
- step：表示切片的步长，如果省略，则默认为 1，当省略该步长时，最后一个冒号也可以省略。

提示：

字符串的索引同序列的索引是一样的，也是从 0 开始，并且每个字符占一个位置。



实例 1：

代码如下：

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# *_ coding:utf-8 *_  
  
str1 = "Kubernetes 是最流行的容器编排集群！"  
substr1 = str1[1] #截取第 2 个字符  
substr2 = str1[5:] #从第 6 个字符截取  
substr3 = str1[:5] #从左边开始截取 5 个字符  
substr4 = str1[2:5] #截取第 3 个到第 5 个字符  
print("原字符串：",str1)  
print(substr1 + "\n" + substr2 + "\n" + substr3 + "\n" + substr4 +  
"\n")
```

运行结果：

```
原字符串： Kubernetes 是最流行的容器编排集群！  
u  
netes 是最流行的容器编排集群！  
Kuber  
ber
```

在进行字符串截取时，如果指定的索引不存在，则会抛出如下异常：

```
Traceback (most recent call last):  
  File "I:\soft\1\2.py", line 5, in <module>  
    substr1 = str1[30] #截取第 2 个字符  
IndexError: string index out of range
```

要解决这个问题，可以使用 try...except 语句捕获异常。

实例 2：

代码如下：

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_  
  
str1 = "Kubernetes 是最流行的容器编排集群！"  
try:  
    substr1 = str1[30]  
except IndexError:  
    print("您指定的索引不存在，请核对！")
```

运行结果：

```
您指定的索引不存在，请核对！
```

try...except 是异常处理语句，这里作为扩展知识，后面还会讲。

1.4、分割字符串

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 中，字符串对象提供了分割字符串的方法。分割字符串是把字符串分割为列表。

字符串对象的 `split()` 方法可以实现字符串分割，也就是把一个字符串按照指定的分隔符切分为字符串列表，该列表的元素中，不包括分隔符。

`split()` 方法的语法格式如下：

```
str.split(sep, maxsplit)
```

参数说明：

- `str`：表示要进行分割的字符串。
- `sep`：用于指定分隔符，可以包含多个字符，默认为 `None`，即所有空字符（包括空格、换行“`\n`”、制表符“`\t`”等）。
- `maxsplit`：可选参数，用于指定分割的次数，如果不指定或者为 `-1`，则分割次数没有限制，否则返回结果列表的元素个数最多为 `maxsplit+1`。
- 返回值：分割后的字符串列表。

注意：在 `split()` 方法中，如果不指定 `sep` 参数，那么也不能指定 `maxsplit` 参数。

实例 1：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

str1 = "K8S 官方地址 >>> https://kubernetes.io"
print("原字符串：", str1)
str1_1 = str1.split() #采用默认分隔符进行分割
str1_2 = str1.split('>>>') #采用多个字符串进行分割
str1_3 = str1.split('.') #采用“.”进行分割
str1_4 = str1.split(' ', 4) #采用空格进行分割，并且只分割前 4 个
print(str(str1_1) + '\n' + str(str1_2) + '\n' + str(str1_3) + '\n'
+ str(str1_4))
str1_5 = str1.split('>') #采用“>”进行分割
print(str1_5)
```

运行结果：

```
原字符串： K8S 官方地址 >>> https://kubernetes.io
['K8S', '官', '方', '地', '址', '>>>', 'https://kubernetes.io']
['K8S 官方地址', 'https://kubernetes.io']
['K8S 官方地址 >>> https://kubernetes', 'io']
['K8S', '官', '方', '地', '址 >>> https://kubernetes.io']
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
['K8S 官方地址', '', '', ' https://kubernetes.io']
```

说明：

在使用 `split()` 方法时，如果不指定参数，默认采用空白符进行分割，这时无论有几个空格或者空白符都将作为一个分隔符进行分割。

`str1_2` 和 `str1_5` 会比较难以理解，如果指定一个分隔符，那么当这个分隔符出现多个时，就会每个分割一次，没有得到内容的，将参数一个空元素。例如 `str1_5` 就得到了两个空元素。

1.5、检索字符串

在 Python 中，字符串对象提供了很多应用于字符串查找的方法，这里主要介绍以下几种方法：

1、`count()` 方法

`count()` 方法用于检索指定字符串在另一个字符串中出现的次数。如果检索的字符串不存在，则返回 0，否则返回出现的次数。

`count()` 语法格式如下：

```
str.count(sub[, start[, end]])
```

参数说明：

- `str`：表示原字符串。
- `sub`：表示要检索的子字符串。
- `start`：可选参数，表示检索范围的起始位置的索引，如果不指定，则从头开始检索。
- `end`：可选参数，表示检索范围的结束位置的索引，如果不指定，则一直检索到结尾。

实例 1：统计字符串中“@”符号出现的次数

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

str1 = '@Linux @Centos @Ubuntu'
print('字符串 “', str1, '” 中包括', str1.count('@'), '个@符号')
```

运行结果：

字符串 “ @Linux @Centos @Ubuntu ” 中包括 3 个@符号

2、`find()` 方法

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

find() 方法用于检索是否包含指定的子字符串。如果检索的字符串不存在，则返回-1，否则返回首次出现该子字符串时的索引。

find() 语法格式如下：

```
str.find(sub[, start[, end]])
```

参数说明：

- str：表示原字符串。
- sub：表示要检索的子字符串。
- start：可选参数，表示检索范围的起始位置的索引，如果不指定，则从头开始检索。
- end：可选参数，表示检索范围的结束位置的索引，如果不指定，则一直检索到结尾。

实例 2：检索字符串中首次出现“@”符号的位置索引。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = '@Linux @Centos @Ubuntu'
print('字符串 “’,str1,’ ” 中@符号首次出现的位置索引为：’,str1.find('@'),)
```

运行结果：

字符串 “ @Linux @Centos @Ubuntu ” 中@符号首次出现的位置索引为： 0

如果只想要判断指定的字符串是否存在，可以使用 in 关键字实现。如下实例 3：

实例 3：判断字符串中是否存在@符号，如果存在就返回 True，否则返回 False。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = '@Linux @Centos @Ubuntu'
print('@符号是否存在此字符串中：’,('@')in str1)
```

运行结果：

@符号是否存在此字符串中： True

3、rfind() 方法

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

rfind()方法，作用与find()方法类似，只是从右边开始查找。

4、index()方法

index()方法同find()方法类似，也是用于检索是否包含指定的子字符串。只不过如果使用index()方法，当指定的字符串不存在时会抛出异常。

index()语法格式如下：

```
str.index(sub[, start[, end]])
```

参数说明：

- str：表示原字符串。
- sub：表示要检索的子字符串。
- start：可选参数，表示检索范围的起始位置的索引，如果不指定，则从头开始检索。
- end：可选参数，表示检索范围的结束位置的索引，如果不指定，则一直检索到结尾。

实例 4：使用 index()方法检索首次出现“@”符号的位置索引。检索首次出现“A”字母的位置索引。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = '@Linux @Centos @Ubuntu'
print('字符串 “’,str1,’ ” 中@符号首次出现的位置索引为：',str1.index('@'),)
print('字符串 “’,str1,’ ” 中 A 符号首次出现的位置索引为：',str1.index('A'),)
```

运行结果：

```
字符串 “ @Linux @Centos @Ubuntu ” 中@符号首次出现的位置索引为： 0
Traceback (most recent call last):
  File "I:\soft\1\2.py", line 6, in <module>
    print('字符串 “’,str1,’ ” 中 A 符号首次出现的位置索引为：',str1.index('A'),)
ValueError: substring not found
```

index 检索不存在元素时会出现如上的异常报错。

5、rindex()方法

rindex()方法，作用与index()方法类似，只是从右边开始查找。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

6、startswith() 方法

startswith() 方法用于检测字符串是否以指定子字符串开头。如果是则返回 True，否则返回 False。

startswith() 语法格式如下：

```
str.startswith(prefix[, start[, end]])
```

参数说明：

- str：表示原字符串。
- prefix：表示要检索的子字符串。
- start：可选参数，表示检索范围的起始位置的索引，如果不指定，则从头开始检索。
- end：可选参数，表示检索范围的结束位置的索引，如果不指定，则一直检索到结尾。

实例 5：检索字符串是否以“@”符号开头。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = '@Linux @Centos @Ubuntu'
print('判断字符串 “', str1, '” 是否以@符号开头，结果为：', str1.startswith('@'))
```

运行结果：

判断字符串“ @Linux @Centos @Ubuntu ”是否以@符号开头，结果为： True

7、endswith() 方法

endswith() 方法用于检测字符串是否以指定子字符串结尾。如果是则返回 True，否则返回 False。

endswith() 语法格式如下：

```
str.endswith(suffix[, start[, end]])
```

参数说明：

- str：表示原字符串。
- prefix：表示要检索的子字符串。
- start：可选参数，表示检索范围的起始位置的索引，如果不指定，则从头开始检索。
- end：可选参数，表示检索范围的结束位置的索引，如果不指定，则一直检索到结尾。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 6：检索字符串是否以“@”符号开头。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "K8S 官方地址: https://kubernetes.io"
print('判断字符串 "', str1, '" 是否以.io 结尾, 结果为: ', str1.endswith('.io'))
```

运行结果：

判断字符串 “ K8S 官方地址: https://kubernetes.io ” 是否以.io 结尾, 结果为: True

1.6、字母的大小写转换

在 Python 中，字符串对象提供了 lower() 方法和 upper() 方法进行字母的大小写转换，即可用于将大写字母转换为小写字母或者将小写字母修改为大写字母。

1、lower() 方法

lower() 方法用于将大写字母转换为小写字母。

如果字符串中没有需要被转换的字符，则将原字符串返回。否则将返回一个新的字符串，将原字符串中每个需要进行小写转换的字符都转换成等价的小写字母。

字符长度与原字符长度相同。

lower() 语法格式如下：

```
str.lower()
```

参数说明：

● str：表示要进行转换的字符串。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "K8S 官方地址: https://Kubernetes.io"
print("原字符串: ", str1)
print("新字符串: ", str1.lower())
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

原字符串： K8S 官方地址： <https://Kubernetes.io>

原字符串： k8s 官方地址： <https://kubernetes.io>

2、upper() 方法

upper() 方法用于将小写字母修改为大写字母。

如果字符串中没有需要被转换的字符，则将原字符串返回。否则将返回一个新的字符串，将原字符串中每个需要进行大写转换的字符都转换成等价的大写字符。

字符长度与原字符串长度相同。

upper() 语法格式如下：

```
str.upper()
```

参数说明：

● str：表示要进行转换的字符串。

实例 2：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

str1 = "K8S 官方地址： https://Kubernetes.io"
print("原字符串：", str1)
print("新字符串：", str1.upper())
```

运行结果：

原字符串： K8S 官方地址： <https://Kubernetes.io>

新字符串： K8S 官方地址： [HTTPS://KUBERNETES.IO](https://KUBERNETES.IO)

1.7、去除字符串中的空格和特殊字符

用户在输入数据时，可能会无意中输入多余的空格，或在一些情况下，字符串前后不允许出现空格和特殊字符，此时就需要去除字符串中的空格和特殊字符。

strip() 方法可以去除字符串左右两边的空格和特殊字符。

lstrip() 方法可以去除字符串左边的空格和特殊字符。

rstrip() 方法可以去除字符串右边的空格和特殊字符。

这里的特殊字符是指制表符“\t”、回车符“\r”、换行符“\n”等。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1、strip()方法

strip()方法用于去除字符串左、右两侧的空格和特殊字符。

strip()语法格式如下：

```
str.strip()[chars]
```

参数说明：

- str：表示要去除空格的字符串。
- chars：可选参数，用于指定要去除的字符，可以指定多个。

如果设置 chars 为“@.”，则去除左、右两侧包括“@”或“.”。如果不指定 chars 参数，默认将去除空格、制表符“\t”、回车符“\r”、换行符“\n”等。

实例 1：

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

str1 = " K8S 官方地址: https://Kubernetes.io \n"
print('原字符串 str1: ' + str1 + '!')
print('修改后的字符串 str1: ' + str1.strip() + '!') #去除字符串首尾的空格和特殊字符

str2 = "@Kubernetes@."
print('原字符串 str2: ' + str2 + '!')
print('修改后的字符串 str2: ' + str2.strip("@.") + '!') #去除字符串首尾的 "@" "."
```

运行结果：

```
原字符串 str1:  K8S 官方地址: https://Kubernetes.io
!
修改后的字符串 str1: K8S 官方地址: https://Kubernetes.io!
原字符串 str2: @Kubernetes@.!
修改后的字符串 str2: Kubernetes!
```

2、lstrip()方法

lstrip()方法用于去除字符串左侧的空格和特殊字符。

lstrip()语法格式如下：

```
str.lstrip()[chars]
```

参数说明：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

- str: 表示要去除空格的字符串。
- chars: 可选参数，用于指定要去除的字符，可以指定多个。
如果设置 chars 为“@.”，则去除左侧包括“@”或“.”。如果不指定 chars 参数，默认将去除空格、制表符“\t”、回车符“\r”、换行符“\n”等。

实例 2:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

str1 = "\n K8S 官方地址: https://Kubernetes.io"
print('原字符串 str1: ' + str1 + '!')
print('修改后的字符串 str1: ' + str1.lstrip() + '!') #去除字符串
左侧的空格和特殊字符
str2 = "@Kubernetes"
print('原字符串 str2: ' + str2 + '!')
print('修改后的字符串 str2: ' + str2.lstrip("@") + '!') #去除字符
串左侧的@
```

运行结果:

```
原字符串 str1:
K8S 官方地址: https://Kubernetes.io!
修改后的字符串 str1: K8S 官方地址: https://Kubernetes.io!
原字符串 str2: @Kubernetes!
修改后的字符串 str2: Kubernetes!
```

3、rstrip() 方法

rstrip() 方法用于去除字符串右侧的空格和特殊字符。

rstrip() 语法格式如下:

```
str.rstrip()[chars]
```

参数说明:

- str: 表示要去除空格的字符串。
- chars: 可选参数，用于指定要去除的字符，可以指定多个。
如果设置 chars 为“@.”，则去除右侧包括“@”或“.”。如果不指定 chars 参数，默认将去除空格、制表符“\t”、回车符“\r”、换行符“\n”等。

实例 3:

代码如下:

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# *_ coding:utf-8 *_  
  
str1 = " K8S 官方地址: https://Kubernetes.io \n"  
print(' 原字符串 str1: ' + str1 + '!')  
print(' 修改后的字符串 str1: ' + str1.rstrip() + '!') #去除字符串  
右侧的空格和特殊字符  
str2 = "Kubernetes@,"  
print(' 原字符串 str2: ' + str2 + '!')  
print(' 修改后的字符串 str2: ' + str2.rstrip("@,") + '!') #去除字  
符串右侧的@和,
```

运行结果:

```
原字符串 str1:  K8S 官方地址: https://Kubernetes.io  
!  
修改后的字符串 str1:  K8S 官方地址: https://Kubernetes.io!  
原字符串 str2: Kubernetes@,!  
修改后的字符串 str2: Kubernetes!
```

1.8、格式化字符串

格式化字符串是指先制定一个模板，在这个模板中预留几个空位，然后再根据需求填上相应的内容。这些空位需要通过指定的符号标记（也称为占位符），而这些符号还不会显示出来。在 Python 中，格式化字符串有以下两种方法：

1、使用 “%” 操作符

“%”是 Python 风格的字符串格式化操作符，非常类似 C 语言里的 printf() 函数的字符串格式化（C 语言中也是使用%）。

语法格式如下：

```
'%[-][+][0][m][.n]格式化字符'%exp
```

参数说明：

- -：可选参数，用于指定左对齐，正数前方无符号，负数前面加负号。
- +：可选参数，用于指定右对齐，正数前方加正号，负数前方加负号。
- 0：可选参数，表示右对齐，正数前方无符号，负数前方加负号，用 0 填充空白处（一般与 m 参数一起使用）。
- m：可选参数，表示占有宽度。
- .n：可选参数，表示小数点后保留的位数。
- 格式化字符：用于指定类型，常用的格式化字符如下所示：
 %s：格式化字符串（采用 str() 显示）
 %r：格式化字符串（采用 repr() 显示）

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

%c: 格式化字符及其 ASCII 码
%o: 八进制整数
%d 或者%i: 十进制整数
%e: 用科学计数法格式化浮点数
%x: 十六进制整数
%E: 同%e
%f 或者%F: 格式化浮点数字，可指定小数点后的精度
%: 字符%

● exp: 要转换的项。如果要指定的项有多个，需要通过元组的形式进行指定，但不能使用列表。

提示:

str() 得到的字符串是面向用户的，具有较好的可读性

repr() 得到的字符串是面向机器的

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

template = '编号: %09d\t 公司名称: %s \t 官网: %s' #定义模板
context1 = (1, '百度', 'www.baidu.com') #定义要转换的内容 1
context2 = (2, 'Kubernetes', 'kubernetes.io')
print(template%context1) #格式化输出
print(template%context2)
```

运行结果:

```
编号: 000000001  公司名称: 百度    官网: www.baidu.com
编号: 000000002  公司名称: Kubernetes    官网: kubernetes.io
```

2、使用字符串对象的 format() 方法

字符串对象提供了 format() 方法用于进行字符串格式化。

format() 语法格式如下:

str.format(args)

参数说明:

- str: 用于指定字符串的显示样式（即模板）
- args: 用于指定要转换的项。如果有多项，则用逗号分隔。

创建模板时，需要使用“{}”和“:”指定占位符，基本语法格式如下:

{[index][:[[fill]align][sign][#][width][.precision][type]]}

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

参数说明：

- index: 可选参数，用于指定要设置格式的对象在参数列表中的索引位置，索引值从 0 开始。如果省略，则根据值的先后顺序自动分配。
- file: 可选参数，用于指定空白处填充的字符。
- align: 可选参数，用于指定对齐方式（值为“<”表示内容左对齐；值为“>”表示内容右对齐；值为“=”表示内容右对齐，将符号放在填充内容的最左侧，且支队数字类型有效；值为“^”表示内容居中），需要配合 width 一起使用。
- sign: 可选参数，用于指定有无符号数（值为“+”表示正数加号，负数加负号；值为“-”表示正数不变，负数加负号；值为空格表示正数加空格，负数加负号）。
- #: 可选参数，对于二进制、八进制和十六进制数，如果加上#，表示会显示“0b/0o/0x”前缀，否则不显示前缀。
- width: 可选参数，用于指定所占宽度。
- .precision: 可选参数，用于指定保留的小数位数。
- type: 可选参数，用于指定类型，如下所示：
 - s: 对字符串类型格式化
 - d: 十进制整数
 - c: 将十进制整数自动转换成对应的 Unicode 字符
 - e 或者 E: 转换为科学计数法表示再格式化
 - g 或者 G: 自动在 e 和 f 或者 E 和 F 中切换
 - b: 将十进制整数自动转换成二进制表示再格式化
 - o: 将十进制整数自动转换成八进制表示再格式化
 - x 或者 X: 将十进制整数自动转换成十六进制表示再格式化
 - f 或者 F: 转换为浮点数（默认小数点后保留 6 位）在格式化
 - %: 显示百分百（默认显示小数点后 6 位）

实例 2:

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

template = '编号: {:0>9s}\t 公司名称: {:s} \t 官网: {:s}' #定义模板
context1 = template.format('1','百度','www.baidu.com') #定义要转换的内容 1
context2 = template.format('2','Kubernetes','kubernetes.io')
print(context1)
print(context2)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
编号：000000001  公司名称：百度    官网：www.baidu.com
编号：000000002  公司名称：Kubernetes  官网：kubernetes.io
```

2、正则表达式基础

我们在学习 Linux 操作系统的时候，大家应该都执行过这个命令吧“ls *.txt”，按下<Enter>键后，所有的“.txt”文件都会被列出来。这里的“*.txt”就可以理解为一个简单的正则表达式。

2.1、行定位符

行定位符就是用来描述子串的边界。“^”表示行的开始；“\$”表示行的结尾。

例如：

```
^zyf: 匹配 zyf 行的开始
zyf$: 匹配 zyf 行的结尾
zyf:  匹配出现在字符串的任意部分
```

2.2、元字符

除了前面介绍的元字符“^”和“\$”外，正则表达式里还有很多的元字符，例如下面的正则表达式就应用了元字符“\b”和“\w”。

```
\bmr\w*\b
```

表示用匹配以字母 mr 开头的单词，先是从某个单词开始处（\b），然后匹配字母 mr，接着是任意数量的字母或数字（\w*），最后是单词结束处（\b）。该表达式可以匹配“mrsoft”“mrbook”和“mr123456”等，但不能与“amr”匹配。

常用的元字符如下：

```
.: 匹配除换行符以外的任意字符
\w: 匹配字母、数字、下划线或汉字
\W: 匹配除字母、数字、下划线或汉字以外的字母
\s: 匹配单个的空白符（包括<Tab>键和换行符）
\S: 除单个空白字符（包括<Tab>键和换行符）以外的所有字符
\d: 匹配数字
\b: 匹配单词的开始或结束，单词的分界符通常是空格、标点符号或者换行
^: 匹配字符串的开始
$: 匹配字符串的结束
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

提示：括号在正则表达式中也算是一个元字符。

2.3、限定符

使用 `(\w*)` 匹配任意数量的字母或数字。如果想匹配特定数量的数字，该如何表示呢？

正则表达式为我们提供了限定符（指定数量的字符）来实现该功能。

如匹配 8 位数 QQ 号可用如下表达式：

```
^\d{8}$
```

常用的限定符如下：

?: 匹配前面的字符零次或一次

例如：`colou?r`，该表达式可以匹配 `colour` 和 `color`。

+: 匹配前面的字符一次或多次

例如：`go+gle`，该表达式可以匹配的范围从 `gogle` 到 `goo...gle`

*: 匹配前面的字符零次或多次

例如：`go*gle`，该表达式可以匹配的范围从 `ggle` 到 `goo...gle`

{n}: 匹配前面的字符 n 次

例如：`go{2}gle`，该表达式值匹配 `google`

{n,}: 匹配前面的字符最少 n 次

例如：`go{2,}gle`，该表达式可以匹配的范围从 `google` 到 `goo...gle`

{n,m}: 匹配前面的字符最少 n 次，最多 m 次

例如：`employe{0,2}`，该表达式可以匹配 `employ`、`employe` 和 `employee` 3 种情况。

2.4、字符类

正则表达式查找数字和字母是很简单的，因为已经有了对应这些字符集合的元字符（如 `“\d”`，`“\w”`），但是如果匹配没有预定义元字符的字符集合，应该怎么办？

很简单，只需要在方括号里列出它们就行了。如：`[.?!]` 匹配标点符号（`“.”` 或 `“?”` 或 `“!”`）

也可以指定一个字符的范围，像 `“[0-9]”`、`“[a-z0-9A-Z]”`。

2.5、排除字符

匹配不符合指定字符集合的字符串。正则表达式提供了 `“^”` 字符。

如下：

```
[^a-zA-Z]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

表示用于匹配一个不是字母的字符。

2.6、选择字符

试想一下，如何匹配身份证号码？

首先需要了解一下身份证号码的规则。身份证号码长度为 18 位。最后一位可能是数字或字符 X。

在上面的描述中，包含着条件选择的逻辑，这就需要使用选择字符（|）来实现。该字符可以理解为“或”，匹配身份证的表达式可以写成如下方式：

```
(^\d{18}$)|(^\\d{17})(\\d|X|x)$
```

表达式的意思是匹配 18 位数字，或者 17 位数字和最后一位。最后一位可以是数字或者是 X 或者是 x。

2.7、转义字符

正则表达式中的转移字符（\）和 Python 中的大同小异，都是将特殊字符（如“.” “?” “\”等）变为普通的字符。

2.8、分组

分组就是用一对圆括号“()”括起来的正则表达式，匹配出的内容就表示一个分组。从正则表达式的左边开始看，看到的第一个左括号“(”表示第一个分组，第二个表示第二个分组，依次类推。

例如：

```
(six|four)th
```

这个表达式的意思是匹配单词 sixth 或 fourth

3、使用 re 模块实现正则表达式操作

Python 提供了 re 模块，用于实现正则表达式的操作。在实现时，可以使用 re 模块提供的方法（例如：search()、match()、findall()等）进行字符串处理，也可以先使用 re 模块的 compile() 方法将模块字符串转换为正则表达式对象，然后再使用该正则表达式对象的相关方法来操作字符串。

使用 re 模块之前，必须先引用 re 模块，语法如下：

```
import re
```

3.1、匹配字符串

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

匹配字符串可以使用 re 模块提供的 search()、match()、findall() 等方法实现。

1、使用 match() 方法进行匹配

match() 方法用于从字符串的开始处进行匹配，如果在起始位置匹配成功，则返回 Match 对象，否则返回 None。

match() 语法格式如下：

```
re.match(pattern, string, [flags])
```

参数说明：

pattern：表示模式字符串，由要匹配的正则表达式转换而来。

string：表示要匹配的字符串。

flags：可选参数，表示标志位，用于控制匹配方式，如是否区分字母大小写。常用标志如下所示：

A 或者 ASCII：对于 \w、\W、\b、\B、\d、\D、\s 和 \S 只进行 ASCII 匹配。

I 或者 IGNORECASE：执行不区分字母大小写的匹配

M 或 MULTILINE：将 ^ 和 \$ 用于包括整个字符串的开始和结尾的每一行（默认情况下，仅适用于整个字符串的开始和结尾处）

S 或 DOTALL：使用 (.) 字符匹配所有字符，包括换行符

X 或 VERBOSE：忽略模式字符串中未转义的空格和注释

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'mr_\w+' # 模式字符串
string = 'MR_SHOP mr_shop' # 要匹配的字符串
match = re.match(pattern, string, re.I) # 匹配字符串，不区分大小写
print(match) # 输出匹配结果
string = '项目名称 MR_SHOP mr_shop'
match = re.match(pattern, string, re.I) # 匹配字符串，不区分大小写
print(match) # 输出匹配结果
```

运行结果：

```
<re.Match object; span=(0, 7), match='MR_SHOP'>
None
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

提示：

```
pattern = r'mr_\w+'
```

由于模式字符串中可能包括大量的特殊字符和反斜杠，所以需要写为原生字符串，即在模式字符串前加 r 或 R。

\w 表示匹配字母、数字、下划线或汉字。

Match 对象中包含了匹配值的位置和匹配数据。其中：

- 要获取匹配值的起始位置可以使用 Match 对象的 start() 方法。
- 要获取匹配值的结束位置可以使用 Match 对象的 end() 方法。
- 通过 Match 对象的 span() 方法可以返回匹配位置的元组。
- 通过 Match 对象的 string 属性可以获取要匹配的字符串。

实例 2：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'mr_\w+' # 模式字符串
string = 'MR_SHOP mr_shop' # 要匹配的字符串
match = re.match(pattern, string, re.I) # 匹配字符串，不区分大小写
print('匹配值的起始位置：', match.start())
print('匹配值的结束位置：', match.end())
print('匹配位置的元组：', match.span())
print('要匹配的字符串：', match.string)
```

运行结果：

```
匹配值的起始位置： 0
匹配值的结束位置： 7
匹配位置的元组： (0, 7)
要匹配的字符串： MR_SHOP mr_shop
```

2、使用 search() 方法进行匹配

search() 方法用于在整个字符串中搜索第一个匹配的值，如果匹配成功，则返回 Match 对象，否则返回 None。

search() 语法格式如下：

```
re.search(pattern, string, [flags])
```

参数说明：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

pattern: 表示模式字符串，由要匹配的正则表达式转换而来。
string: 表示要匹配的字符串。
flags: 可选参数，表示标志位，用于控制匹配方式，如是否区分字母大小写。常用标志如下所示：
A 或者 ASCII: 对于 \w、\W、\b、\B、\d、\D、\s 和 \S 只进行 ASCII 匹配。
I 或者 IGNORECASE: 执行不区分字母大小写的匹配
M 或 MULTILINE: 将 ^ 和 \$ 用于包括整个字符串的开始和结尾的每一行（默认情况下，仅适用于整个字符串的开始和结尾处）
S 或 DOTALL: 使用 (.) 字符匹配所有字符，包括换行符
X 或 VERBOSE: 忽略模式字符串中未转义的空格和注释

实例 3:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'mr_\w+' # 模式字符串
string = 'MR_SHOP mr_shop' # 要匹配的字符串
match = re.search(pattern, string, re.I) # 搜索字符串，不区分大小写
print(match)
string = '项目名称 MR_SHOP mr_shop'
match = re.search(pattern, string, re.I) # 搜索字符串，不区分大小写
print(match)
```

运行结果:

```
<re.Match object; span=(0, 7), match='MR_SHOP'>
<re.Match object; span=(4, 11), match='MR_SHOP'>
```

从上面的输出结果可以看出，search() 方法不仅仅是在字符串的起始位置搜索，其他位置有符号的匹配也可以。

3、使用 findall() 方法进行匹配

findall() 方法用于在整个字符串中搜索所有符合正则表达式的字符串，并以列表的形式返回。如果匹配成功，则返回包含匹配结构的列表，否则返回空列表。

findall() 语法格式如下:

```
re.findall(pattern, string, [flags])
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

参数说明：
pattern: 表示模式字符串，由要匹配的正则表达式转换而来。
string: 表示要匹配的字符串。
flags: 可选参数，表示标志位，用于控制匹配方式，如是否区分字母大小写。常用标志如下所示：
A 或者 ASCII: 对于\w、\W、\b、\B、\d、\D、\s和\S只进行ASCII匹配。
I 或者 IGNORECASE: 执行不区分字母大小写的匹配
M 或 MULTILINE: 将^和\$用于包括整个字符串的开始和结尾的每一行(默认情况下，仅适用于整个字符串的开始和结尾处)
S 或 DOTALL: 使用(.)字符匹配所有字符，包括换行符
X 或 VERBOSE: 忽略模式字符串中未转义的空格和注释

实例 4:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'mr_\w+' # 模式字符串
string = 'MR_SHOP mr_shop' # 要匹配的字符串
match = re.findall(pattern, string, re.I) # 搜索字符串，不区分大小写
print(match)
string = '项目名称 MR_SHOP mr_shop'
match = re.findall(pattern, string) # 搜索字符串，区分大小写
print(match)
```

运行结果:

```
['MR_SHOP', 'mr_shop']
['mr_shop']
```

3.2、替换字符串

sub()方法用于实现字符串替换。

sub()方法语法格式如下:

```
re.sub(pattern, repl, string, count, flags)
```

参数说明:

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

pattern: 表示模式字符串，由要匹配的正则表达式转换而来。
repl: 表示替换的字符串。
string: 表示要匹配的字符串。
count: 可选参数，表示模式匹配后替换的最大次数，默认为 0，表示替换所有的匹配。
flags: 可选参数，表示标志位，用于控制匹配方式，如是否区分字母大小写。常用标志如下所示：
A 或者 ASCII: 对于 \w、\W、\b、\B、\d、\D、\s 和 \S 只进行 ASCII 匹配。
I 或者 IGNORECASE: 执行不区分字母大小写的匹配
M 或 MULTILINE: 将 ^ 和 \$ 用于包括整个字符串的开始和结尾的每一行（默认情况下，仅适用于整个字符串的开始和结尾处）
S 或 DOTALL: 使用 (.) 字符匹配所有字符，包括换行符
X 或 VERBOSE: 忽略模式字符串中未转义的空格和注释

实例 1：替换手机号码

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'1[34578]\d{9}'
monile = "您的手机号码为：13411111111"
result = re.sub(pattern, '13412345678', monile)
print(result)
```

运行结果：

您的手机号码为：13412345678

提示：

pattern = r'1[34578]\d{9}': 表示匹配 1 开头的字符串，第二位是 [34578] 其中一位，{9} 表示后面还有 9 位数字。

3.3、使用正则表达式分割字符串

split() 方法用于实现根据正则表达式分割字符串，并以列表的形式返回，其作用与字符串对象的 split() 方法类似，所不同的就是分割字符由模式字符串指定。

split() 语法格式如下：

```
re.split(pattern, repl, string, [maxsplit], [flags])
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

参数说明：
pattern: 表示模式字符串，由要匹配的正则表达式转换而来。
repl: 表示替换的字符串。
maxsplit: 可选参数，表示最大拆分次数。
count: 可选参数，表示模式匹配后替换的最大次数，默认为 0，表示替换所有的匹配。
flags: 可选参数，表示标志位，用于控制匹配方式，如是否区分字母大小写。常用标志如下所示：
A 或者 ASCII: 对于 \w、\W、\b、\B、\d、\D、\s 和 \S 只进行 ASCII 匹配。
I 或者 IGNORECASE: 执行不区分字母大小写的匹配
M 或 MULTILINE: 将 ^ 和 \$ 用于包括整个字符串的开始和结尾的每一行（默认情况下，仅适用于整个字符串的开始和结尾处）
S 或 DOTALL: 使用 (.) 字符匹配所有字符，包括换行符
X 或 VERBOSE: 忽略模式字符串中未转义的空格和注释

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re

pattern = r'[?|&]' # 定义分隔符
monile = "USER?admin&PASSWD?123456"
result = re.split(pattern, monile) # 分割字符串
print(result)
```

运行结果:

```
['USER', 'admin', 'PASSWD', '123456']
```

第 7 章: if 条件语句

流程控制对于任何一门编程语言来说都是十分重要的，例如：我们学习的 Shell，也有流程控制，因为它提供了控制程序如何执行的方法。

如果没有流程控制的话，整个程序都将按照从上至下的顺序来执行，而不能根据客户的需求决定程序执行的顺序。

1、程序结构

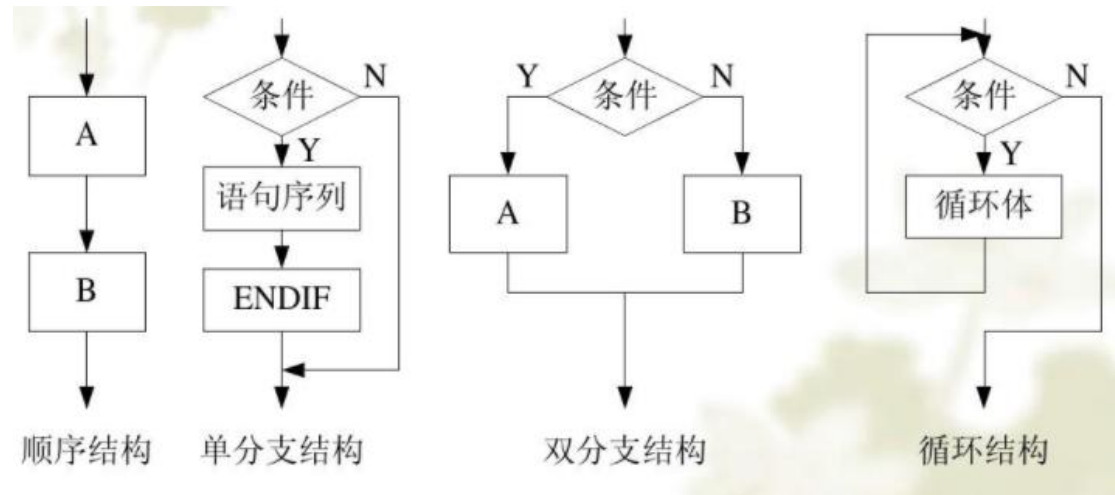
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

计算机在解决某个具体问题时，主要有 3 种情形，分别是：

- 1、顺序执行所有的语句
- 2、选择执行部分语句
- 3、循环执行部分语句

对应下图中的顺序结构、分支结构、循环结构



第一幅图是顺序结构的流程图，编写完毕的语句按照编写顺序依次被执行。

第二、三幅图是选择结构的流程图，它主要根据条件语句的结果选择执行不同的语句。

第四幅图是循环结构的流程图，它是在一定条件下反复执行某段程序的流程结果，其中，被反复执行的语句称为循环体，决定循环是否终止的判断条件称为循环条件。

2、if 条件语句

2.1、单 if 语句

Python 中使用 if 保留字来组成选择语句，单 if 语句语法如下：

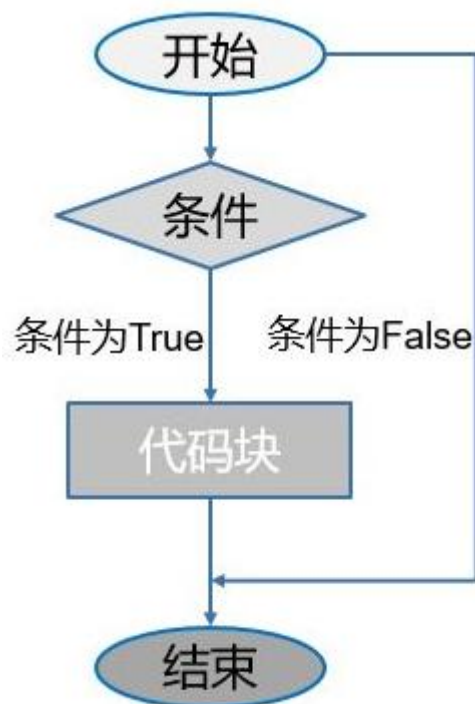
```
if 表达式:  
    语句块
```

表达式可以是一个单纯的布尔值或变量，也可以是比较表达式或逻辑表达式，在条件语句中常用的是比较算术符。

流程图如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



实例 1：判断奖票号码是否中奖

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
number = input("请输入您的 6 位奖票号码：")
if number == "010429":
    print("您的号码是：" + number + "，恭喜您中了本期的大奖，请速来领奖！")

if number != "010429":
    print("您的号码是：" + number + "，很遗憾您未中奖！")
```

实例 2：判断数值是否相等

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
number = 5
if number == 5:
    print("number 的值为：5")
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

number 的值为：5

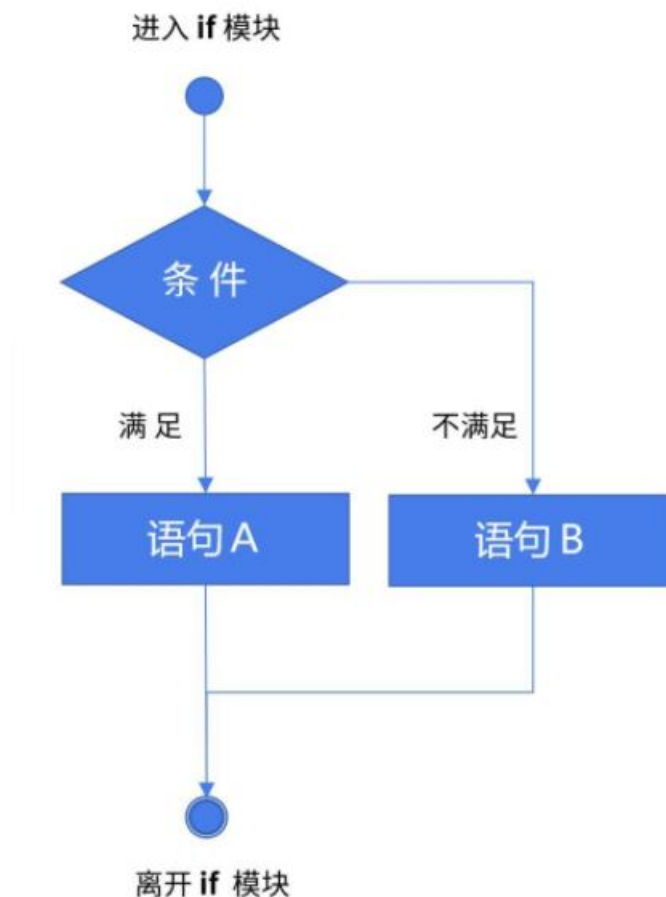
2.2、if...else 语句

如果遇到只能二选一的条件，那么就可以选择 Python 中提供的 if...else 语句解决类似的问题。

if...else 语法格式如下：

```
if 表达式：
    语句块 1
else：
    语句块 2
```

if...else 语句流程图如下图所示：



实例 1：判断奖票号码是否中奖

代码如下：

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# *_ coding:utf-8 *_  
number = input("请输入您的 6 位奖票号码：")  
if number == "010429":  
    print("您的号码是：" + number + "，恭喜您中了本期的大奖，请速来领奖！")  
else:  
    print("您的号码是：" + number + "，很遗憾您未中奖！")
```

2.3、if...elif...else 语句

如果遇到要判断的条件比较多的话，并且 if...else 语法解决不了的，那么我们就可以使用 if...elif...else 语法。

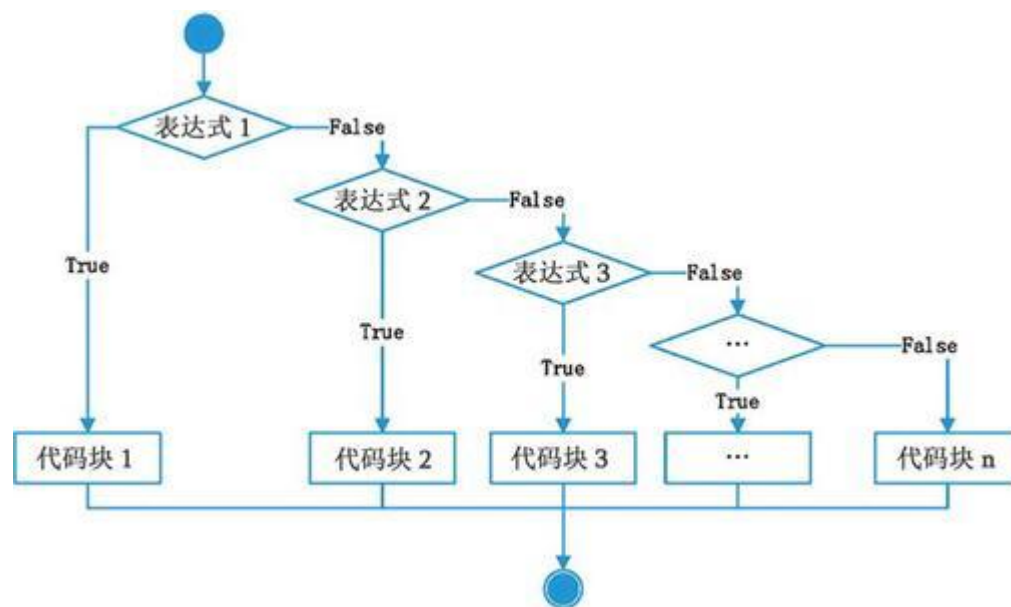
if...elif...else 语法格式如下：

```
if 表达式:  
    语句块 1  
elif:  
    语句块 2  
elif:  
    语句块 3  
...  
elif:  
    语句块 n  
else:  
    语句块 n+1
```

if...elif...else 语句流程图如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



实例 1：输入小王（语文，英语，数学）成绩（单科满分 100 分）判断成绩评定等级

学员评定标准如下：

成绩 \geq 90 分：A

90 分 $>$ 成绩 \geq 80 分：B

80 分 $>$ 成绩 \geq 70 分：C

70 分 $>$ 成绩 \geq 60 分：D

成绩 $<$ 60 分：E

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

chinese_result = int(input("请输入语文成绩："))
maths_result = int(input("请输入数学成绩："))
englist_result = int(input("请输入英语成绩："))
avg_result = (chinese_result + maths_result + englist_result) / 3

if avg_result >= 90:
    print("你的平均分为：%.2f, 成绩的总和评定为：A" % avg_result)
elif avg_result >= 80 and avg_result < 90:
    print("你的平均分为：%.2f, 成绩的总和评定为：B" % avg_result)
elif avg_result >= 70 and avg_result < 80:
    print("你的平均分为：%.2f, 成绩的总和评定为：C" % avg_result)
elif avg_result >= 60 and avg_result < 70:
    print("你的平均分为：%.2f, 成绩的总和评定为：D" % avg_result)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
else:  
    print("你的平均分为: %.2f, 成绩的总和评定为: E" % avg_result)
```

提示: elif 和 else 必须跟 if 一起使用, 不能够单独使用。

2.4、if 语句的嵌套

前面介绍了 3 种形式的 if 选择语句, 这 3 种形式的选择语句之间都可以进行互相嵌套。

在最简单的 if 语句中嵌套 if...else, 语法如下:

```
if 表达式 1:  
    if 表达式 2:  
        语句块 1  
else:  
    语句块 2
```

在 if...else 语句中嵌套 if...else 语句, 语法如下:

```
if 表达式 1:  
    if 表达式 2:  
        语句块 1  
    else:  
        语句块 2  
else:  
    if 表达式 3:  
        语句块 3  
    else:  
        语句块 4
```

实例 1:

输入小王 (语文, 英语, 数学) 成绩 (单科满分 100 分) 判断成绩评定等级

学员评定标准如下:

成绩 >= 90 分: A

90 分 > 成绩 >= 80 分: B

80 分 > 成绩 >= 70 分: C

70 分 > 成绩 >= 60 分: D

成绩 < 60 分: E

成绩 >= 80 分奖励一朵小红花, 80 > 成绩 >= 60 继续加油, < 60 叫家长。

代码如下:

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
chinese_result = int(input("请输入语文成绩: "))
maths_result = int(input("请输入数学成绩: "))
englist_result = int(input("请输入英语成绩: "))
avg_result = (chinese_result + maths_result + englist_result) / 3

if avg_result >= 80:
    print("成绩不错，继续加油，奖励你一朵小红花🌸!")
    if avg_result >= 90:
        print("你的平均分为: %.2f, 成绩的总和评定为: A" % avg_result)
    else:
        if avg_result >= 80 and avg_result < 90:
            print("你的平均分为: %.2f, 成绩的总和评定为: B" %
avg_result)
        elif avg_result >= 60 and avg_result < 80:
            print("分数有点低，再接再厉，好好努力!")
            if avg_result >= 70 and avg_result < 80:
                print("你的平均分为: %.2f, 成绩的总和评定为: C" % avg_result)
            else:
                if avg_result >= 60 and avg_result < 70:
                    print("你的平均分为: %.2f, 成绩的总和评定为: D" %
avg_result)
                else:
                    print("分数太低了，把你家长叫来，我和你家长交流一下!")
                    print("你的平均分为: %.2f, 成绩的总和评定为: E" % avg_result)
```

提示：上面用到了三 if 语句嵌套。If 选择语句可以有多种嵌套方式，开发程序时可以根据自身需要选择合适的嵌套方式，但一定要严格控制好不同级别代码块的缩进量。

3、使用 and、or、not 连接条件的选择语句

其实 and 在上面我们也讲到了，大家都是做运维的，写过 shell 程序的话，应该对 and 和 or 不陌生吧。好吧，我们再以一个案例讲一遍。

and（并且）表示列出的条件都满足则为真。
or（或者）表示列出的条件满足一条则为真。
not（非）取反

实例 1：白富美测试

代码如下：

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# *_ coding:utf-8 *_  
  
color = input("你白么? ")  
money = int(input("你的存款余额: "))  
beautiful = input("你美么? ")  
if color == "白" and money >= 1000000 and beautiful == "美":  
    print("白富美!")  
else:  
    print("矮穷矬!")
```

实例 2:

代码如下:

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_  
  
a = input("请输入 1 位数字密码: ")  
b = ['0','1','2','3','4','5','6','7','8','9']  
if a not in b: # 输入内容未在数字列表中  
    print("非法输入")
```

第 8 章：循环结构语句

什么是循环？

生活例子：

日常生活中有很多问题都无法一次解决，如盖楼，所有高楼都是一层一层地垒起来的，还有一些事务必须要周而复始地运转才能保证其存在的意义，如公交车、地铁等交通工具必须每天往返于始发站和终点站之间。类似这样反复做同一件事的情况，称为循环。

循环主要有两种类型：

- 1、重复一定次数的循环，称为计次循环。如 for 循环。
- 2、一直循环，直到条件满足时才结束循环，称为条件循环。只要条件不为真，这种循环会一直持续下去。如 while 循环。

1、for 循环

for 循环是一个计次循环，通常适用于枚举或遍历序列，以及迭代对象中的元素。

for 循环语法如下：

```
for 迭代变量 in 对象:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

循环体

参数说明：

迭代变量用于保存读取处的值。

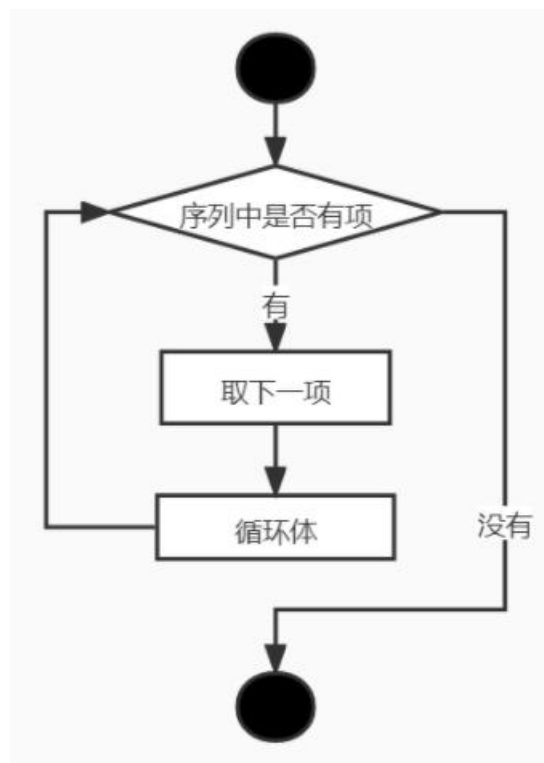
对象为要遍历或迭代的对象，该对象可以是任何有序的序列对象，如字符串、列表和元组等。

循环体为一组被重复执行的语句。

生活例子解释 for 循环：

在体育课上，体育老师要求同学们排队进行踢毽球测试，每个同学只有一次机会，毽球落地则换另一个同学，直到全部同学都测试完毕，则循环结束。

for 循环语句的执行流程图如下图所示：



1、进行数值循环

实例 1：实现计算从 1 到 100 的累加

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

print("计算 1+2+3+...+100 的结果为：")
result = 0 # 保存累加结果的变量
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
for i in range(101):  
    result += i # 实现累加功能  
print(result)
```

运行结果：

计算 1+2+3+...+100 的结果为：
5050

range(101)表示从 0 开始生成一系列连续的整数，生成到 101，不包含 101。也就是 1、2、3、...、100。

2、遍历字符串

使用 for 循环语句除了可以循环数值，还可以遍历字符串。

实例 2：将下列字符串转换为纵向显示

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
string = "天道酬勤"  
print(string)  
for ch in string:  
    print(ch)
```

运行结果：

天道酬勤
天
道
酬
勤

2、while 循环

while 循环是通过一个条件来控制是否要继续反复执行循环体中的语句。

while 循环语法如下：

```
while 条件表达式:  
    循环体
```

参数说明：

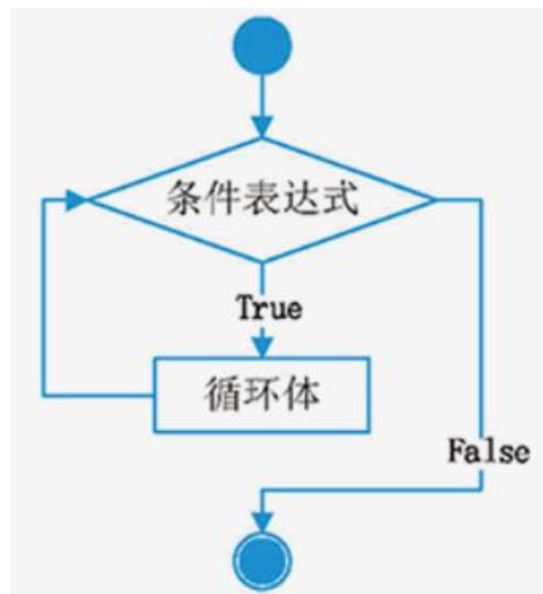
循环体是指一组被重复执行的语句。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

当条件表达式的返回值为 True 时，则执行循环体中的语句。执行完毕后，重新判断条件表达式的返回值，直到表达式返回的结果为 False 时，退出循环。

while 循环语句的执行流程如下图所示：



生活例子解释 while 循环：

在体育课上，体育老师要求同学们沿着环形操场跑圈。要求当听到老师吹的哨子声时就停下来。同学们每跑一圈，可能会请求一次老师吹哨子。如果老师吹哨子，则停下来，即循环结束。否则继续跑步，即执行循环。

实例 1：

在取款机上取款时需要 6 位数银行卡密码。下面模拟一个取款机，输入取款密码则可以进行取款，输入错误则提示“密码错误，已经输错 x 次”，输错 5 次后，则冻结账户。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

password = "010429"
i = 1
while i < 6:
    num = input("请输入六位数密码：")
    password = int(password)
    num = int(num)
    if num == password:
        print("密码正确，正在进行操作，请稍等！")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
i = 7
else:
    print("密码错误，已经输错", i ,"次")
    i += 1

if i == 6:
    print("密码错误 5 次，账户已冻结，请联系客户！")
```

除了以上这种方法，还可以写 `while true:`，表示一直为真，一直运行。就形成了死循环。

3、循环嵌套

在 Python 中，for 循环和 while 循环都可以进行循环嵌套。

1、while 循环中套用 while 循环格式如下：

```
while 条件表达式 1:
    while 条件表达式 2:
        循环体 2
    循环体 1
```

2、for 循环中套用 for 循环格式如下：

```
for 迭代变量 1 in 对象 1:
    for 迭代变量 2 in 对象 2:
        循环体 2
    循环体 1
```

3、while 循环中套用 for 循环格式如下：

```
while 条件表达式:
    for 迭代变量 in 对象:
        循环体 2
    循环体 1
```

4、for 循环中套用 while 循环格式如下：

```
for 迭代变量 in 对象:
    while 条件表达式:
        循环体 2
    循环体 1
```

除了上面介绍的 4 中嵌套格式外，还可以实现更多层的嵌套。和上面的方法类似。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

4、跳转语句

当循环条件一直满足时，程序将会一直执行下去，就像一辆迷路的车，在某个地方不停地转圈。如果希望在中间离开循环，也就是 for 循环结束计数之前，或者 while 循环找到结束条件之前。有两种方法来做到：

- 1、使用 break 语句完全中止循环。
- 2、使用 continue 语句直接跳到下一次循环。

4.1、break 语句

break 语句可以终止当前的循环，包括 while 和 for 在内的所有控制语句。

生活例子：

以独自一人沿着操场跑步为例，原计划跑 10 圈。可是在跑到第 2 圈的时候，遇到自己的女神或者男神，于是果断停下来，终止跑步，这就相当于使用了 break 语句提前终止了循环。

在 while 语句中使用 break 语句的语法如下：

```
while 条件表达式 1:
    执行代码
    if 条件表达式 2:
        break
```

其中，条件表达式 2 用于判断何时调用 break 语句跳出循环。

在 for 语句中使用 break 语句的语法如下：

```
for 迭代变量 in 对象:
    if 条件表达式:
        break
```

其中，条件表达式用于判断何时调用 break 语句跳出循环。

实例 1：输出 1 到 10，输出到 5 则结束循环。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

num = 5
for i in range(1, 11):
    if i == num:
        break
    else:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(i)
```

运行结果：

```
1
2
3
4
```

4.2、continue 语句

continue 语句的作用没有 break 语句强大，它只能终止本次循环而提前进入到下一次循环中。

生活例子：

仍然以独自一人沿着操场跑步为例，原计划跑步 10 圈。当跑到第 2 圈的时候，遇到自己的女神或者男神，于是果断停下来，跑回起点等待，制造一次完美邂逅，然后从第 3 圈开始继续。

在 while 语句中使用 continue 语句的语法如下：

```
while 条件表达式 1:
    执行代码
    if 条件表达式 2:
        continue
```

其中，条件表达式 2 用于判断何时调用 continue 语句跳出循环。

在 for 语句中使用 continue 语句的语法如下：

```
for 迭代变量 in 对象:
    if 条件表达式:
        continue
```

其中，条件表达式用于判断何时调用 continue 语句跳出循环。

实例 1：输出 1 到 10，输出到 5 则跳出本次循环。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

num = 5
for i in range(1,11):
    if i == num:
        continue
    else:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(i)
```

运行结果：

```
1
2
3
4
6
7
8
9
10
```

实例 2：逢七拍腿游戏

从 1 开始依次数数，当数到 7（包括尾数是 7 的情况）或 7 的倍数时，则不说出该数，而是拍一下腿。现在编写程序，计算从 1 数到 99，一共要拍多少次腿？

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

total = 99 #记录拍腿次数的变量
for number in range(1,100): #创建一个从 1 到 99 的循环
    if number % 7 == 0: #判断是否为 7 的倍数
        continue #继续下一次循环
    else:
        string = str(number) #将数值转换为字符串
        if string.endswith('7'): #判断是否以数字 7 结尾
            continue #继续下一次循环
        total -= 1 #拍腿次数 1
print("从 1 数到 99 共拍腿",total,"次") #显示拍腿次数
```

运行结果：

```
从 1 数到 99 共拍腿 22 次
```

第 9 章：字典与集合

1、字典

字典和列表类似，也是可变序列，不过与列表不同，它是无序的可变序列，保存的内容是以“键值对”的形式存放的。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

字典类似于《新华字典》，可以把拼音和汉字关联起来。通过音节表可以快速找到想要找到的汉字。其中《新华字典》里的音节表相当于键（key），而对应得汉字，相当于值（value）。键是唯一的，而值可以有多个。

学过 C++ 或者 Java 的同学应该知道 Map 对象吧？

Python 中的字典相当于 C++ 或者 Java 中的 Map 对象。

字典的主要特征如下：

1、通过键而不是通过索引来读取

字典有时也称为关联数组或者散列表（hash）。它是通过键将一系列的值联系起来的，这样就可以通过键从字典中获取指定项，但不能通过索引来获取。

2、字典是任意对象的无序集合

字典是无序的，各项是从左到右随机排序的，即保存在字典中的项没有特定的顺序。这样可以提高查找的效率。

3、字典是可变的，并且可以任意嵌套

字典可以在原处增长或者缩短（无须生成一份备份），并且它支持任意深度的嵌套（即它的值可以是列表或者其他的字典）。

4、字典中的键必须唯一

不允许同一个键出现两次，如果出现两次，则后一个键会被记住。

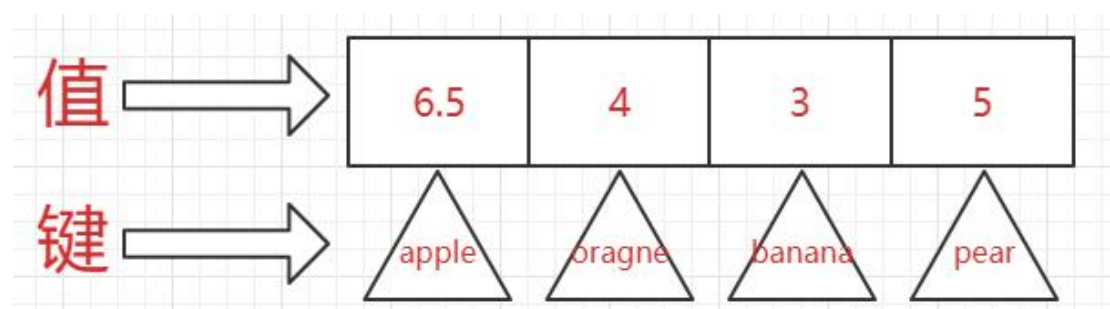
5、字典中的键必须不可变

字典中的键是不可变的，所以可以使用数值、字符串或者元组，但不能使用列表。

1.1、字典的创建和删除

定义字典时，每个元素都包含两个部分“键”和“值”。

以水果名称和价钱的字典为例，键为水果名称，值为水果的价格，如下图所示：



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

创建字典时，在“键”和“值”之间使用冒号分隔，相邻两个元素使用逗号分隔，所有元素放在一个大括号“{}”中。

字典语法格式如下：

```
dictionary = {'key1': 'value1', 'key2': 'value2', ..., 'keyn': 'valuen', }
```

参数说明：

- **dictionary**：表示字典名称
- **key**：表示元素的键，必须是唯一的，并且是不可变的，可以是字符串、数字或者元组。
- **value**：表示元素的值，可以是任意数据类型，不是必须唯一。

实例 1：创建一个个人联系方式的字典

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '1754111111', 'wx': 'ZYF0429', 'zyf': '134012345678'}
print(dictionary)
```

Python 的 `dict()` 方法可以创建一个空字典，也可以通过已有的数据快速创建字典。

1、创建空字典

```
dictionary = dict()
```

2、通过映射函数创建字典

语法如下：

```
dictionary = dict(zip(list1, list2))
```

参数说明：

- **dictionary**：表示字典名称
- **zip()** 函数：用于将多个列表或元组对应位置的元素组合为元组，并返回包含这些内容的 `zip` 对象。如果想得到元组，可以使用 `tuple()` 函数将 `zip` 对象转换为元组。如果想得到列表，则可以使用 `list()` 函数将其转换为列表。
- **list1**：一个列表，用于指定要生成字典的键。
- **list2**：一个列表，用于指定要生成字典的值。

实例 2：

代码如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

system1 = ['linux', 'windows']
system2 = ['centos', 'windows server 2019']
dictionary = dict(zip(system1, system2))
print(dictionary)
```

运行结果：

```
{'linux': 'centos', 'windows': 'windows server 2019'}
```

3、通过给定的“键值对”创建字典

语法如下：

```
dictionary = dict(key1='value1', key2='value2', ...,
keyn='valuen',)
```

参数说明：

- dictionary：表示字典名称
- key：表示元素的键，必须是唯一的，并且是不可变的，可以是字符串、数字或者元组。
- value：表示元素的值，可以是任意数据类型，不是必须唯一。

实例 3：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary = dict(qq = '1754111111', wx = 'ZYF0429', zyf =
'134012345678')
print(dictionary)
```

字典的删除也是使用 del 命令删除整个字典，如下删除字典：

```
del dictionary
```

如果只想删除字典的全部元素，可以使用字典对象的 clear() 方法实现，如下清除字典的全部元素：

```
dictionary.clear()
```

1.2、通过“键值对”访问字典

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

访问字典中的对象可以使用字典对象的 `get()` 方法获取指定键的值。语法如下：

```
dictionary.get(key, [default])
```

参数说明：

- `dictionary`：表示字典名称，即要从中获取值的字典。
- `key`：表示指定的键
- `default`：可选参数，用于指定当指定的“键”不存在时，返回一个默认值，如果省略，则返回 `None`。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary = {'qq': '175412345678', 'zyf': '13401234567'}
print("我的 qq 号是：", dictionary.get('qq', '您的 qq 账号不存在'))
print("我的 wx 号是：", dictionary.get('wx', '您的 wx 账号不存在'))
```

运行结果：

```
我的 qq 号是： 175412345
我的 wx 号是： 您的 wx 账号不存在
```

1.3、遍历字典

字典是以“键值对”的形式存储数据的，所以在使用字典时需要获取这些“键值对”。Python 提供了遍历字典的方法，通过遍历可以获取字典中的全部“键值对”。

使用字典对象的 `items()` 方法可以获取字典的“键值对”列表，语法格式如下：

```
dictionary.items()
```

参数说明：

`dictionary` 为字典对象。返回值为可遍历的（“键值对”）的元组列表。想要获取到具体的“键值对”，可以通过 `for` 循环遍历该元组列表。

实例 1：获取元组中的各个元素

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
for i in dictionary.items():
    print(i)
```

运行结果：

```
('qq', '175412345')
('wx', 'ZYF0429')
('zyf', '134012345678')
```

实例 2：获取到具体的每个键和值

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
for key,value in dictionary.items():
    print(key,"的联系方式是：",value)
```

运行结果：

```
qq 的联系方式是： 175412345
wx 的联系方式是： ZYF0429
zyf 的联系方式是： 134012345678
```

在 Python 中，字典对象还提供了 values() 和 keys() 方法，用于返回字典的值和列表。也需要通过 for 循环遍历该字典列表，获取对应的值和键。

案例如下：

实例 3：根据用户输入的指定 key，查找 key 的 value 值

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
print("当前元组定义的 keys 有如下：")
for key in dictionary.keys():
    print(key)
int_key = input("请输入要查询 key 的值：")
print(dictionary.get(int_key))
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

1.4、添加、修改和删除字典元素

由于字典是可变序列，所以可以随时在其中添加“键值对”，这和列表类似。向字典中添加元素的语法格式如下：

```
dictionary[key] = value
```

参数说明：

- dictionary：表示字典名称。
- key：表示要添加元素的键，必须是唯一的，并且是不可变的，可以是字符串、数字或者元组。
- value：表示元素的值，可以是任何数据类型，不是必须唯一。

实例 1：添加字典元素

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
dictionary['email'] = "175412345@qq.com"
print(dictionary)
```

运行结果：

```
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678', 'email': '175412345@qq.com'}
```

实例 2：删除字典元素

注意：要删除的字典元素必须存在，否则会报错，可以结合 if 判断应用。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
if "zyf" in dictionary:
    del dictionary["zyf"]
else:
    print("要删除的 key 不存在！")
print(dictionary)
```

实例 3：修改字典元素

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

dictionary =
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '134012345678'}
dictionary["zyf"] = "13907110429"
print(dictionary)
```

运行结果：

```
{'qq': '175412345', 'wx': 'ZYF0429', 'zyf': '13907110429'}
```

1.5、字典推导式

使用字典推导式可以快速生成一个字典，它的表现形式和列表推导式类似。

实例 1：生成一个包含 4 个随机数的字典，键使用数字表示

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
rand = {i: random.randint(10, 100) for i in range(1, 5)}
print(rand)
```

运行结果：

```
{1: 46, 2: 25, 3: 56, 4: 95}
```

2、集合

在 Python 中集合有可变集合（set）和不可变集合（frozenset）两种概念。本章只介绍 set 无序可变序列集合。

2.1、集合的创建

在 Python 中提供了两种创建集合的方法：

- 1、直接使用“{}”创建
- 2、通过 set() 函数将列表、元组等可迭代对象转换为集合。（推荐使用）

1、直接使用“{}”创建

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 中，创建 set 集合也可以像列表、元组和字典一样，直接将集合赋值给变量从而实现创建集合。语法格式如下：

```
setname = {element 1, element 2, element 3, ..., element n}
```

参数说明：

- setname: 表示集合的名称，可以是任何符合 Python 命名规则的标识符。
- element 1, element 2, element 3, ..., element n: 表示集合中的元素，个数没有限制，数据类型只要是 Python 支持的就可以。

提示：在创建集合时，如果输入了重复的元素，Python 会自动只保留一个。

实例 1：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

set1 = {'Linux', 'Ubuntu', 'Debian'}
set2 = {1, 2, 3, 4, 5, 1, 4, 6}
print(set1)
print(set2)
```

运行结果：

```
{'Debian', 'Linux', 'Ubuntu'}
{1, 2, 3, 4, 5, 6}
```

2、使用 set() 函数创建

在 Python 中，可以使用 set() 函数将列表、元组等其他可迭代对象转换为集合。

set() 函数的语法格式如下：

```
setname = set(iteration)
```

参数说明：

- setname: 表示集合的名称
- iteration: 表示要转换为集合的可迭代对象，可以是列表、元组、range 对象等。另外，也可以是字符串，如果是字符串，返回的集合将是包含全部不重复字符的集合。

实例 2：下面的每一行代码都可以创建一个集合

代码如下：

```
#!/usr/bin/env python
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# *_ coding:utf-8 *_  
  
set1 = set("十年磨剑无人知，一朝出剑震天下！")  
set2 = set(["人生苦读","我用 Python"])  
print(set1)  
print(set2)
```

运行结果：

```
{'人', '震', '!', '剑', '出', '十', '天', ' ', ' ', '无', '磨', '朝',  
'一', '下', '年', '知'}  
{'我用 Python', '人生苦读'}
```

提示：python 中的集合是无序的，所以每次输出时元素的排列顺序可能会与上面的不同。

上面的输出，如果有相同字符，那么只会保留一个。

在 Python 中，创建集合时推荐采用 set() 函数实现。

2.2、集合的添加和删除

集合是可变序列，所以在创建集合后，还可以对创建的集合添加或删除元素。

1、向集合中添加元素

向集合中添加元素可以使用 add() 方法实现。

add() 语法如下：

```
setname.add(element)
```

参数说明：

- setname：表示要添加元素的集合。
- element：表示要添加的元素内容。

实例 1：

代码如下：

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_  
  
set1 = set(['Unix', 'Centos', 'Redhat', 'Debian'])  
print(set1)  
set1.add('Ubuntu')  
print(set1)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
{'Unix', 'Redhat', 'Debian', 'Centos'}  
{'Unix', 'Ubuntu', 'Debian', 'Redhat', 'Centos'}
```

2、向集合中删除元素

在 Python 中，可以使用 del 命令删除整个集合，也可以使用集合的 pop() 方法或者 remove() 方法删除一个元素，或者使用集合对象的 clear() 方法清空集合，即删除集合中的全部元素。

实例 2：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
set1 = set(['Unix', 'Centos', 'Redhat', 'Debian'])  
set1.remove('Unix') #移除指定元素  
print(set1)  
set1.pop() #移除一个元素  
print(set1)  
set1.clear() #清空集合  
print(set1)
```

运行结果：

```
{'Debian', 'Centos', 'Redhat'}  
{'Centos', 'Redhat'}  
set()
```

2.3、集合的交集、并集和差集运算

集合最常用的操作就是进行交集、并集和差集运算。

进行交集运算时使用“&”符号。进行并集运算时使用“|”符号。进行差集运算时使用“-”符号。

实例 1：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
set1 = set(['1', '2', '3', '4', '5'])  
set2 = set(['3', '4', '5', '6', '7'])  
print('交集运算：', set1 & set2)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(' 并集运算: ', set1 | set2)
print(' 差集运算: ', set1 - set2)
```

运行结果:

交集运算: {'5', '4', '3'}

并集运算: {'1', '3', '7', '2', '5', '4', '6'}

差集运算: {'2', '1'}

3、列表、元组、字典和集合的区别

列表、元组、字典和集合的区别如下表所示:

数据结构	是否可变	是否重复	是否有序	定义符号
列表	可变	可重复	有序	[]
元组	不可变	可重复	有序	()
字典	可变	可重复	无序	{key:value}
集合	可变	不可重复	无序	{}

第 10 章：用函数实现模块化程序设计

前面所学习的知识,都是从上到下依次执行的,如果某段代码需要多次使用,那么则需要将该段代码多次复制。

这种做法会很大程度上影响开发效率,在实际项目开发中是不可取的。
那么如果想要多次使用某段代码,我们应该怎么办呢?

在 Python 中,可以使用函数来达到这个目的。我们把实现某一功能的代码定义为一个函数,在需要使用时,随时调用即可,十分方便。对于函数,简单理解就是可以完成某项工作的代码块,类似于积木块,可以反复地使用。

1、函数的创建和调用

在前面,我们学习的输出的 print() 函数、用于输入的 input() 函数,以及用于生成一系列整数的 range() 函数。这些都是 Python 内置的标准函数,可以直接使用。

除了可以直接使用的标准函数外,Python 还支持自定义函数。即通过将一段有规律的、重复的代码定义为函数,来达到一次编写多次调用的目的。使用函数可以提高代码的重复利用率。

1.1、函数的创建

创建函数也称为定义函数,可以理解为创建一个具有某种用途的工具。使用 def 关键字实现。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

def 关键字的语法格式如下：

```
def functionname([parameterlist]):  
    '''comments'''  
    [functionbody]
```

参数说明：

- functionname: 函数名称，在调用函数时使用
- parameterlist: 可选参数，用于指定函数中传递的参数。如果有多个参数，各参数间使用逗号“,”分隔。如果不指定，则表示该函数没有参数。在调用时，也不指定参数。
- '''comments''': 可选参数，表示为函数指定注释，注释的内容通常是说明该函数的功能、要传递的参数的作用等，可以为用户提供友好提示和帮助的内容。
- functionbody: 可选参数，用于指定函数体，即该函数被调用后，要执行的功能代码。如果函数有返回值，可以使用 return 语句返回。

提示：即使函数没有参数时，也必须保留一对空的小括号“()”，否则会报错。

实例 1:

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
def fun_bmi(name,height,weight):  
    '''  
    功能：根据身高和体重计算 BMI 指数  
    name: 姓名  
    height: 身高，单位：米  
    weight: 体重，单位：千克  
    '''  
    print(name + "的身高：" + str(height) + "米 \t 体重：" +  
str(weight) + "千克")  
    bmi = weight/(height*height)  
    print(name + "的 BMI 指数为：" + str(bmi))  
    # 判断身材是否合理  
    if bmi<18.5:  
        print("您的身体过轻 ~@_@~")  
    if bmi>=18.5 and bmi<24.9:  
        print("正常范围，注意保持 (-_-)")  
    if bmi>=24.9 and bmi<29.9:  
        print("您的体重过重 ~@_@~")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
if bmi>=29.9:  
    print("肥胖 ^@_@^")
```

运行上面的代码，将不显示任何内容，也不会抛出异常，因为 fun_bmi() 函数还没有被调用。

1.2、函数的调用

调用函数也就是执行函数。如果把创建的函数理解为创建一个具有某种用途的工具，那么调用函数就相当于使用该工具。

调用函数的语法格式为：

```
functionname([parametersvalue])
```

参数说明：

- functionname：函数名称，要调用的函数名称，必须是已经创建好的。
- parametersvalue：可选参数，用于指定各个参数的值。如果需要传递多个参数值，则各参数值间使用逗号“,”分隔。如果该函数没有参数，则直接写一对小括号即可。

实例 1：调用 1.1 小节的代码

代码如下：

```
#!/usr/bin/env python  
# *_ coding:utf-8 *_  
  
def fun_bmi(name,height,weight):  
    ,,,  
    功能：根据身高和体重计算 BMI 指数  
    name：姓名  
    height：身高，单位：米  
    weight：体重，单位：千克  
    ,,,  
    print(name + "的身高：" + str(height) + "米 \t 体重：" +  
str(weight) + "千克")  
    bmi = weight/(height*height)  
    print(name + "的 BMI 指数为：" + str(bmi))  
    # 判断身材是否合理  
    if bmi<18.5:  
        print("您的身体过轻 ^@_@~")  
    if bmi>=18.5 and bmi<24.9:  
        print("正常范围，注意保持 (-_-)")  
    if bmi>=24.9 and bmi<29.9:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print("您的体重过重 ~@_@~")
if bmi>=29.9:
    print("肥胖 ^@_@^")
fun_bmi("张岩峰",1.83,60)
```

运行结果：

```
张岩峰的身高：1.83 米      体重：60 千克
张岩峰的 BMI 指数为：17.916330735465376
您的身体过轻 ~@_@~
```

提示：输入一对括号之后，就可以看到参数提示，如下图所示。



1.3、pass 空语句

在 Python 中有一个 pass 语句，表示空语句，它不做任何使用，一般起到占位作用。

实例 1：创建一个函数，但我们暂时不知道该函数要实现什么功能，这时就可以使用 pass 语句填充函数的主体。

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

def fun_bmi(name,height,weight):
    pass
```

2、函数的参数传递

在调用函数时，大多数情况下，主调函数和被调用函数之间有数据传递关系，这就是有参数的函数形式。函数参数的作用是传递数据给函数使用，函数利用接收到的数据进行具体的操作处理。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2.1、形式参数和实际参数

在使用函数是，经常会用到形式参数（形参）和实际参数（实参）。两者都叫做参数。有什么区别呢？如下看：

1、通过作用理解

形式参数和实际参数在作用上的区别如下：

- 形式参数：在定义函数时，函数名后面括号中的参数为“形式参数”，也称形参。

- 实际参数：在调用一个函数时，函数名后面括号中的参数为“实际参数”。也就是将函数的调用者提供给函数的参数称为实际参数，也称实参。

例：

```
def test(hello):    <==定义或创建函数，此时的函数参数 hello 为形式参数。
    print(hello)
    test = "hello"
    demo(test)      <==调用函数，此时的函数参数 test 是实际参数。
```

根据实参的类型不同，可以分为将实参的值传递给形参，和将实参的引用传递给形参两种情况。

当实参为不可变对象时，进行的是值传递。

当实参为可变对象时，进行的是引用传递。

实际上，值传递和引用传递的基本区别就是，进行值传递后，改变形参的值，实参的值不变。而进行引用传递后，改变形参的值，实参的值也一同改变。

实例 1：

定义一个名称为 demo 的函数，然后为 demo() 函数传递一个字符串类型的变量作为参数（代表值传递），并在函数调用前后分别输出该字符串变量。再为 demo() 函数传递一个列表类型的变量作为参数（代表引用传递），并在函数调用前后分别输出该列表。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

# 定义函数
def demo(obj):
    print("原值：", obj)
    obj += obj

# 调用函数
print("=====值传递=====")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
mot = "十年磨剑无人知，一朝出剑震天下！"
print("函数调用前：", mot)
demo(mot)      # 采用不可变对象(字符串)
print("函数调用后：", mot)

print("=====引用传递=====")
list = ['CentOS', 'Ubuntu', 'Debian']
print("函数调用前：", list)
demo(list)     # 采用可变对象(列表)
print("函数调用后：", list)
```

运行结果：

```
=====值传递=====
函数调用前： 十年磨剑无人知，一朝出剑震天下！
原值： 十年磨剑无人知，一朝出剑震天下！
函数调用后： 十年磨剑无人知，一朝出剑震天下！
=====引用传递=====
函数调用前： ['CentOS', 'Ubuntu', 'Debian']
原值： ['CentOS', 'Ubuntu', 'Debian']
函数调用后： ['CentOS', 'Ubuntu', 'Debian', 'CentOS', 'Ubuntu',
'Debian']
```

从上面的执行结果可以看出：

在进行值传递时，改变形参的值后，实参的值不改变。

在进行引用传递时，改变形参的值后，实参的值也会发生改变。

2、通过一个比喻来理解形参和实参

函数定义时参数列表中的参数就是形参，而函数调用时传递进来的参数就是实参，就像剧本选主角一样，剧本的角色相当于形参，而演角色的演员就相当于实参。

2.2、位置参数

位置参数也称必备参数，必须按照正确的顺序传到函数中。即调用时的数量和位置必须和定义时是一样的。

1、数量必须与定义时一致

在调用函数时，指定的实参数量必须与形参数量一致，否则将抛出 `TypeError` 异常，提示缺少必要的位置参数。

比如：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

定义了一个函数 `fun_bmi(name,height,weight)`，这个函数它有 3 个参数，但是如果在调用时，只传递了两个参数，就会出现 `TypeError` 异常。

2、位置必须与定义的一致

在调用函数时，指定的实参位置必须与形参位置一致，否则将产生以下两种结果：

抛出异常的情况主要是因为在调用函数时，如果指定的实参与形参的位置不一样，但是他们的数据类型一致，那么就不会抛出异常，而是产生结果与预期不符的问题。

例如：

```
fun_bmi(name,height,weight)
fun_bmi("张岩峰",60,1.83)
```

2.3、关键字参数

关键字参数是值使用形参的名字来确定输入的参数值。通过这种方式指定参数时，不需要再与形参的位置完全一致，只要将参数名写正确即可。这样可以避免用户需要牢记参数位置的麻烦。

例如：

```
fun_bmi(name,height,weight)
fun_bmi(name = "张岩峰", height = 1.83, weight = 60)
```

2.4、为参数设置默认值

调用函数时，如果没有指定某个参数将抛出异常，在定义函数时，我们可以直接指定参数的默认值。

这样，当没有传入参数时，则直接使用定义函数时设置的默认值。

默认值参数的函数语法格式如下：

```
def functionname([parameter1 = defaultvalue1]):
    [functionbody]
```

参数说明：

- `functionname`：函数名称，在调用函数时使用
- `parameter1 = defaultvalue1`：可选参数，用于指定向函数中传递的参数，并且为该参数设置默认值为 `defaultvalue1`。
- `functionbody`：可选参数，用于指定函数体，即该函数备调用后，要执行的功能代码。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 1:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def demo(test = 'hello'):
    print(test)
demo()
demo('你好')
```

运行结果:

```
hello
你好
```

3、函数的返回值

在 Python 中，可以在函数体内使用 `return` 语句为函数指定返回值。该返回值可以是任意类型，并且无论 `return` 语句出现在函数的什么位置，只要得到执行，就会直接结束函数的执行。

`return` 语句的语法格式如下:

```
result = return [value]
```

参数说明:

- `result`: 用于保存返回结果，如果返回一个值，那么 `result` 中保存的就是返回的一个值，该值可以是任意类型。如果返回多个值，那么 `result` 中保存的就是一个元组。
- `value`: 可选参数，用于指定要返回的值，可以返回一个值，也可以返回多个值。

提示: 当函数中没有 `return` 语句时，或者省略了 `return` 语句的参数时，将返回 `None`，即返回空值。

实例 1: 根据用户名查找对应用户的手机号

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def fun_checkout(name):
    checkname = ""
    if name == "张岩峰":
        number = "13403991234"
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
elif name == "张三":
    number = "13523351234"
elif name == "李四":
    number = "12306"
else:
    number = "没有找到此人的联系方式"
return number

while True:
    name = input("请输入要查询的用户名称：")
    number = fun_checkout(name)
    print("姓名：", name, "手机号：", number)
```

运行结果：

```
请输入要查询的用户名称：张岩峰
姓名： 张岩峰 手机号： 13403991234
请输入要查询的用户名称：张三
姓名： 张三 手机号： 13523351234
请输入要查询的用户名称：李四
姓名： 李四 手机号： 12306
```

4、变量的作用域

变量的作用域是指程序代码能够访问该变量的区域，如果超出该区域，再访问时就会出现错误。

在程序中，一般会根据变量的“有效范围”，将变量分为“局部变量”和“全局变量”。

4.1、局部变量

局部变量是指在函数内部定义并使用的变量，它只是在函数内部有效。函数内定义的变量，在函数运行之前或者运行完毕之后，所有的变量就都不存在了。所以，如果在函数外部使用函数内部定义的变量，就会出现抛出 NameError 异常。

实例 1：在函数外部调用函数内的变量

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def fun_demo():
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
message = "经常不断地学习，你就什么都知道。你知道得越多，你就越有力量。"
```

```
print('局部变量 message: ', message)
```

```
fun_demo()
```

```
print('局部变量 message: ', message)
```

运行结果：

局部变量 message: 经常不断地学习，你就什么都知道。你知道得越多，你就越有力量。

Traceback (most recent call last):

File "C:\Users\岩峰\Desktop\test\test.py", line 8, in <module>

```
print('局部变量 message: ', message)
```

NameError: name 'message' is not defined

4.2、全局变量

与局部变量对应，全局变量是能够作用于函数内外的变量。全局变量主要有以下两种情况：

1、如果一个变量，在函数外定义，那么不仅可以在函数外可以访问到，在函数内也可以访问到。提示：在函数体以外定义的变量是全局变量。

实例 1：在函数外部调用函数内的变量

代码如下：

```
#!/usr/bin/env python
```

```
# -*- coding:utf-8 -*-
```

```
message = "与其用华丽的外衣装饰自己，不如用知识武装自己。"
```

```
def fun_demo():
```

```
    print('函数体内: ', message)
```

```
fun_demo()
```

```
print('函数体外: ', message)
```

运行结果：

函数体内: 与其用华丽的外衣装饰自己，不如用知识武装自己。

函数体外: 与其用华丽的外衣装饰自己，不如用知识武装自己。

提示：当局部变量与全局变量重名时，对函数体的变量进行赋值后，不影响函数体外的变量。

2、在函数体内定义，并且使用 `global` 关键字修饰后，该变量也可以变成全局变量。在函数体外也可以访问到该变量，并且在函数体内还可以对其进行修改。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实例 2:

代码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

message = "与其用华丽的外衣装饰自己，不如用知识武装自己."
print('函数体外：', message)
def fun_demo():
    global message
    message = "十年磨剑无人知，一朝出剑震天下！"
    print('函数体内：', message)
fun_demo()
print('函数体外：', message)
```

运行结果:

```
函数体外： 与其用华丽的外衣装饰自己，不如用知识武装自己.
函数体内： 十年磨剑无人知，一朝出剑震天下！
函数体外： 十年磨剑无人知，一朝出剑震天下！
```

从上面的结果中可用看出，在函数体内修改了全局变量的值。

注意：在实际开发中，不建议这么做，因为这样容易让代码混乱，很难分清哪些是全局变量、哪些是局部变量。

第 11 章：模块

Python 提供了强大的模块支持，主要提现为不仅在 Python 标准库中包含了大量的模块（称为标准模块），而且还有很多第三方模块，另外开发者自己也可以开发自定义模块。

1、模块概述

模块的英文是 Modules，可以认为是一盒（箱）主体积木，通过它可以拼出某一主体的东西。这与函数不同，一个函数相当于一块积木，而一个模块中可以包括很多函数，也就是很多积木，所以也可以说模块相当于一盒积木。

在 Python 中，一个扩展名为“.py”的文件就称之为一个模块。通常情况下，我们把能够实现某一特定功能的代码放置在一个文件中作为一个模块，从而方便其他程序和脚本导入并使用。另外，使用模块也可以避免函数名和变量名冲突。

经过前面的学习，我们知道对于 Python 代码可以写在一个文件中。但是随着程序不断变大，为了便于维护，需要将其分为多个文件，这样可以提高代码的可维护性。另外，使用模块可以提高代码的可重用性。即编写好一个模块后，只要是实现该功能的程序，都可以导入到这个模块来实现。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2、自定义模块

在 Python 中，自定义模块有两个作用，一个是规范代码，另一个是方便其他程序使用已经编写好的代码，提高开发效率。

要实现自定义模块主要分为两部分，一部分是创建模块，另一部分是导入模块。

2.1、创建模块

创建模块可以将模块中相关的代码（变量定义和函数定义等）编写在一个单独的文件中，并且将该文件命名为“模块名 + .py”的形式，也就是说，创建模块，实际就是创建一个.py 文件。

注意：

- 1、创建模块时，设置的模块名尽量不要与 Python 自带的标准模块名称相同。
- 2、模块文件的扩展名必须是 “.py”

2.2、使用 import 语句导入模块

要使用模块需要先以模块的形式加载模块中的代码，可以使用 import 语句实现。

import 语句的语法格式如下：

```
import modulename [as alias]
```

参数说明：

- modulename：为要导入模块的名称
- [as alias]：给模块起的别名，通过该别名也可以使用模块。

实例 1：

同级目录存在 bmi.py 文件，文件内容如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def fun_bmi(name,height,weight):
    """
    功能：根据身高和体重计算 BMI 指数
    name：姓名
    height：身高，单位：米
    weight：体重，单位：千克
    """
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(name + "的身高：" + str(height) + "米 \t 体重：" +
str(weight) + "千克")
bmi = weight/(height*height)
print(name + "的BMI 指数为：" + str(bmi))
# 判断身材是否合理
if bmi<18.5:
    print("您的身体过轻 ~@@~")
if bmi>=18.5 and bmi<24.9:
    print("正常范围，注意保持 (-_-)")
if bmi>=24.9 and bmi<29.9:
    print("您的体重过重 ~@@~")
if bmi>=29.9:
    print("肥胖 ^@@^")
```

导入 bmi.py 模块

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import bmi as m
m.fun_bmi("张岩峰", 1.83, 60)
```


运行结果：

张岩峰的身高：1.83 米 体重：60 千克
张岩峰的BMI 指数为：17.916330735465376
您的身体过轻 ~@@~

测试完毕之后，会发现当前目录出现了一个__pycache__文件夹

名称	修改日期	类型	大小
 __pycache__	2022/2/9 8:46	文件夹	
 bmi.py	2022/2/9 8:44	Python File	1 KB
 test.py	2022/2/9 8:45	Python File	1 KB

文件夹内有一个叫 bmi.cpython-37.pyc 的文件

 bmi.cpython-37.pyc	2022/2/9 8:46	Compiled Pytho...	1 KB
--	---------------	-------------------	------

拆解 bmi.cpython-37.pyc 文件名：.cpython-37.pyc 其中 cpython 代表的是 c 语言实现的 Python 解释器，-37 代表采用的 Python 版本为 3.7。

pyc 是一种二进制文件，是由 py 文件经过编译后获得的。py 文件变成 pyc 文件后，加载的速度有所提高，而且 pyc 是一种跨平台的字节码，是由 python 的虚拟机来执行的，这个是类似于 JAVA 虚拟机的概念。而且如果采用 Python 开发商业软件的话，就需要编译为 pyc 后，再发布出去以免源码泄露。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

__pycache__文件夹的意义何在呢？

因为第一次执行代码的时候，Python 解释器已经把编译的字节码放在 __pycache__ 文件夹中，这样以后再次运行的话，如果被调用的模块未发生改变，那就直接跳过编译这一步，直接去 __pycache__ 文件夹中去运行相关的 *.pyc 文件，大大缩短了项目运行前的准备时间。

2.3、使用 from...import 语句导入模块

在使用 import 语句导入模块时，每执行一条 import 语句都会创建一个新的命名空间（namespace），并且在该命名空间中执行与 .py 文件相关的所有语句。在执行时，需要在具体的变量、函数和类名前加上“模块名.”前缀。

如果不想在每次导入模块时都创建一个新的命名空间，而是将具体的定义导入到当前的命名空间中。这时可以使用 from...import 语句。使用 from...import 语句导入模块后，不需要再添加前缀，直接通过具体的变量、函数和类名等访问即可。

提示：

命名空间可以理解为记录对象名字和对象之间对应关系的空间。目前 Python 的命名空间大部分都是通过字典（dict）来实现的。其中，key 是标识符；value 是具体的对象。例如，key 是变量的名字，value 则是变量的值。

from...import 语句的语法格式如下：

```
from modulename import member
```

参数说明：

- modulename：为要导入模块的名称，区分字母大小写。

- member：用于指定导入的变量、函数或者类等。可以同时导入多个定义，各个定义之间使用逗号“，”分隔。如果想导入全部定义，也可以使用通配符星号“*”代替。

实例 1：

在导入模块时，如果使用通配符“*”导入全部定义后，想查看具体导入了哪些定义，可以通过显示 dir() 函数的值来查看。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

from bmi import *
print(dir())
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
['__annotations__', '__builtins__', '__doc__', '__file__',  
'__loader__', '__name__', '__package__', '__spec__', 'fun_bmi']
```

其中 fun_bmi 就是我们导入的定义。

注意：

在使用 from...import 语句导入模块中的定义时，需要保证所导入的内容在当前的命名空间中是唯一的，否则将出现冲突，后导入的同名变量、函数或者类会覆盖先导入的。这时就需要使用 import 语句进行导入。

2.4、模块搜索目录

当使用 import 语句导入模块时，默认情况下，会按照以下顺序进行查找：

- (1) 在当前目录（即执行的 Python 脚本文件所在目录）下查找。
- (2) 到 PYTHONPATH（环境变量）下的每个目录中查找。
- (3) 到 Python 的默认安装目录下查找。

实例 1：以上各个目录的具体位置保存在标准模块 sys 的 sys.path 变量中。可以通过以下代码输出具体的目录：

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
import sys  
print(sys.path)
```

运行结果：

```
['I:\\soft\\1', 'H:\\ZYF\\Lib\\idlelib', 'H:\\ZYF\\python37.zip',  
'H:\\ZYF\\DLLs', 'H:\\ZYF\\lib', 'H:\\ZYF',  
'H:\\ZYF\\lib\\site-packages']
```

1、临时添加

提示：通过该方法添加的目录只在执行当前文件的窗口中有效，窗口关闭后即失效。

实例 2：

临时添加即在导入模块的 Python 文件中添加。例如，需要将

“I:\soft\Python 安装位置\Lib” 目录添加到 sys.path 中，代码如下所示

代码如下：

```
#!/usr/bin/env python  
# -*- coding:utf-8 -*-  
  
import sys
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
sys.path.append('I:\soft\Python 安装位置\Lib')
print(sys.path)
```

运行结果：

```
['I:\\soft\\1', 'H:\\ZYF\\Lib\\idlelib', 'H:\\ZYF\\python37.zip',
'H:\\ZYF\\DLLs', 'H:\\ZYF\\lib', 'H:\\ZYF',
'H:\\ZYF\\lib\\site-packages', 'I:\\soft\\Python 安装位置\\Lib']
```

2、增加.pth 文件（推荐）

在 Python 安装目录下的“Lib\\site-packages”子目录中（我的在“H:\\ZYF\\Lib\\site-packages”），创建一个扩展名为.pth 的文件，文件名任意。文件内容为要导入模块所在的目录。

实例 3：

在“H:\\ZYF\\Lib\\site-packages”下创建一个 test_import.pth 文件，内容为“E:\\LINUX”

cmd 下操作步骤如下：

```
H:>cd H:\\ZYF\\Lib\\site-packages
H:\\ZYF\\Lib\\site-packages>echo E:\\LINUX > test_import.pth
H:\\ZYF\\Lib\\site-packages>more test_import.pth
E:\\LINUX
```

使用 sys 模块输出具体目录：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import sys
print(sys.path)
```

运行结果：

```
['I:\\soft\\1', 'H:\\ZYF\\python37.zip', 'H:\\ZYF\\DLLs',
'H:\\ZYF\\lib', 'H:\\ZYF', 'H:\\ZYF\\lib\\site-packages', 'E:\\LINUX']
```

提示：创建完.pth 文件后，需要重新打开要执行的导入模块的 Python 文件，否则新添加的目录不起作用。

提示：实验操作做完记得还原，这里已经删除 test_import.pth 文件

3、以主程序的形式执行

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

这里先创建一个模块，名称为 bmi，在该模块中，首先定义一个全局变量，然后创建一个名称为 fun_bmi() 的函数，最后再通过 print() 函数输出一些内容，代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

bmi = "根据身高和体重计算 BMI 指数"

def fun_bmi(name,height,weight):
    """
    功能：根据身高和体重计算 BMI 指数
    name：姓名
    height：身高，单位：米
    weight：体重，单位：千克
    """
    print(name + "的身高：" + str(height) + "米 \t 体重：" +
str(weight) + "千克")
    bmi = weight/(height*height)
    print(name + "的 BMI 指数为：" + str(bmi))
    # 判断身材是否合理
    if bmi<18.5:
        print("您的身体过轻 ~@_@~")
    if bmi>=18.5 and bmi<24.9:
        print("正常范围，注意保持 (-_-)")
    if bmi>=24.9 and bmi<29.9:
        print("您的体重过重 ~@_@~")
    if bmi>=29.9:
        print("肥胖 ^@_@^")
    print(bmi)
fun_bmi("张岩峰",1.83,60)
```

在与 bmi 模块同级目录下，创建一个名称为 test.py 的文件，在该文件中，导入 bmi 模块，在通过 print() 语句输出模块中的全局变量 bmi 的值，代码如下：

```
代码如下：
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import bmi
print("全局变量的值为：",bmi.bmi)
```

执行结果：

根据身高和体重计算 BMI 指数
张岩峰的身高：1.83 米 体重：60 千克

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
张岩峰的 BMI 指数为: 17.916330735465376
您的身体过轻 ~@@~
全局变量的值为: 根据身高和体重计算 BMI 指数
```

从上面的执行结果可以看出，导入模块后，不仅输出了全局变量的值，而且模块中原有的测试代码也被执行了。这个结果显然不是我们想要的。那么如何只输出全局变量的值呢？

实际上，可以在模块中，将原本直接执行的测试代码放在一个 if 语句中。因此，可以将模块 bmi 的代码修改为以下内容：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

bmi = "根据身高和体重计算 BMI 指数"

def fun_bmi(name,height,weight):
    """
    功能：根据身高和体重计算 BMI 指数
    name: 姓名
    height: 身高，单位：米
    weight: 体重，单位：千克
    """

    print(name + "的身高：" + str(height) + "米 \t 体重：" +
str(weight) + "千克")
    bmi = weight/(height*height)
    print(name + "的 BMI 指数为：" + str(bmi))
    # 判断身材是否合理
    if bmi<18.5:
        print("您的身体过轻 ~@@~")
    if bmi>=18.5 and bmi<24.9:
        print("正常范围，注意保持 (-_-)")
    if bmi>=24.9 and bmi<29.9:
        print("您的体重过重 ~@@~")
    if bmi>=29.9:
        print("肥胖 ^@@^")

if __name__ == "__main__":
    print(bmi)
    fun_bmi("张岩峰",1.83,60)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
Python 3.7.5 Shell
File Edit Shell Debug Options Window Help
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
根据身高和体重计算BMI指数
张岩峰的身高: 1.83米 体重: 60千克
张岩峰的BMI指数为: 17.916330735465376
您的身体过轻 ~0_0~
>>>

bmi.py - I:\soft\1\bmi.py (3.7.5)
File Edit Format Run Options Window Help
#!/usr/bin/env python
# -*- coding:utf-8 -*-

bmi = "根据身高和体重计算BMI指数"

def fun_bmi(name,height,weight):
    """
    功能: 根据身高和体重计算BMI指数
    name: 姓名
    height: 身高, 单位: 米
    weight: 体重, 单位: 千克
    """
    print(name + "的身高: " + str(height) + "米 \t 体重: " + str(weight) + "千克")
    bmi = weight/(height*height)
    print(name + "的BMI指数为: " + str(bmi))
    # 判断身材是否合理
    if bmi<18.5:
        print("您的身体过轻 ~0_0~")
    if bmi>=18.5 and bmi<24.9:
        print("正常范围, 注意保持 (-_-)")
    if bmi>=24.9 and bmi<29.9:
        print("您的体重过重 ~0_0~")
    if bmi>=29.9:
        print("肥胖 ~0_0~")

if __name__ == "__main__":
    print(bmi)
    fun_bmi("张岩峰", 1.83, 60)
```

为什么要判断 `__name__ == "__main__"` 呢？
不着急，先做个输出查看，如下图：

```
Python 3.7.5 Shell
File Edit Shell Debug Options Window Help
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: I:\soft\1\bmi.py =====
根据身高和体重计算BMI指数
张岩峰的身高: 1.83米 体重: 60千克
张岩峰的BMI指数为: 17.916330735465376
您的身体过轻 ~0_0~
__main__
>>>

bmi.py - I:\soft\1\bmi.py (3.7.5)
File Edit Format Run Options Window Help
#!/usr/bin/env python
# -*- coding:utf-8 -*-

bmi = "根据身高和体重计算BMI指数"

def fun_bmi(name,height,weight):
    """
    功能: 根据身高和体重计算BMI指数
    name: 姓名
    height: 身高, 单位: 米
    weight: 体重, 单位: 千克
    """
    print(name + "的身高: " + str(height) + "米 \t 体重: " + str(weight) + "千克")
    bmi = weight/(height*height)
    print(name + "的BMI指数为: " + str(bmi))
    # 判断身材是否合理
    if bmi<18.5:
        print("您的身体过轻 ~0_0~")
    if bmi>=18.5 and bmi<24.9:
        print("正常范围, 注意保持 (-_-)")
    if bmi>=24.9 and bmi<29.9:
        print("您的体重过重 ~0_0~")
    if bmi>=29.9:
        print("肥胖 ~0_0~")

if __name__ == "__main__":
    print(bmi)
    fun_bmi("张岩峰", 1.83, 60)
print(__name__)
```

可以看到使用 `print(__name__)` 输出的结果为 `__main__`，这样判断就是满足条件的，所以就执行了。

那么 `__name__` 是怎么来的呢？

在每个模块的定义中都包括一个记录模块名称的变量 `__name__`，程序可以检查该变量，以确定它们在哪个模块中执行。如果一个模块不是被导入到其他程序中执行，那么它可能在解释器的顶级模块中执行。顶级模块的 `__name__` 变量的值为 `__main__`。

再次执行 `test.py` 文件，输出内容如下：

全局变量的值为： 根据身高和体重计算 BMI 指数

从执行结果中可以看出测试代码没有执行。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

4、引用其他模块

在 Python 中，除了可以自定义模块外，还可以引用其他模块，主要包括使用标准模块和第三方模块。

4.1、导入和使用标准模块

在 Python 中，自带了很多实用的模块，称为标准模块（也可以称为标准库），对于标准模块，我们可以直接使用 import 语句导入到 Python 文件中使用。

实例 1：

导入 random 模块，生成一个 0~10 之间（包括 0 和 10）的随机整数的代码如下：

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import random
print(random.randint(0,10))
```

运行结果：

4

Python 常用的内置标准模块及描述如下所示：

模块名	描述
sys	与 Python 解释器及其环境操作相关的标准库
time	提供与时间相关的各种函数的标准库
os	提供了访问操作系统服务功能的标准库
calendar	提供了与日期相关的各种函数的标准库
urllib	用于读取来自网上（服务器上）的数据的标准库
json	用于使用 JSON 序列化和反序列化对象
re	用于在字符串中执行正则表达式匹配和替换
math	提供标准算术运算符函数的标准库
decimal	用于进行精确控制运算精度、有效数位和四舍五入操作的十进制运算
shutil	用于进行高级文件操作，如复制、移动和重命名等
logging	提供了灵活的记录事件、错误、警告和调试信息等日志信息的功能
tkinter	使用 Python 进行 GUI 编程的标准库

4.2、第三方模块的下载与安装

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在进行 Python 程序开发时，除了可以使用 Python 内置的标准模块外，还有很多第三方模块可以所使用。对于这些第三方模块，可以在 Python 官方推出的“<https://pypi.org/>”中找到。

在使用第三方模块时，需要先下载并安装该模块，然后就可以像使用标准模块一样导入并使用了。下载和安装第三方模块可以使用 Python 提供的 pip 命令。

pip 命令语法格式如下：

```
pip <command> [modulename]
```

参数说明：

- command：用于指定要执行的命令。
- modulename：可选参数，用于指定要安装或者卸载或其他操作的模块名。

实例 1：

安装 kubernetes 第三方模块，两种安装方式，自选：

项目地址：<https://github.com/kubernetes-client/python>



自述文件.md

Kubernetes Python 客户端

build passing | pypi package 21.7.0 | codecov 0% | python 3.6 | 3.7 | 3.8 | 3.9 | 3.10 | Kubernetes client Silver

kubernetes client beta

用于kubernetes API的 Python 客户端。

安装

从来源：

```
git clone --recursive https://github.com/kubernetes-client/python.git
cd python
python setup.py install
```

直接从PyPI：

```
pip install kubernetes
```

这里以 pip 方式进行安装：（推荐）（在 Linux 系统安装方法也是一样的）

```
C:\Users\zhang>pip install kubernetes
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
C:\Users\zhang\pip install kubernetes
Collecting kubernetes
  Downloading kubernetes-21.7.0-py2.py3-none-any.whl (1.8 MB)
    | 1.8 MB 85 kB/s
Collecting google-auth>=1.0.1
  Downloading google_auth-2.6.0-py2.py3-none-any.whl (156 kB)
    | 156 kB 60 kB/s
Collecting requests-oauthlib
  Downloading requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: setuptools>=21.0.0 in g:\py\lib\site-packages (from kubernetes) (58.1.0)
Collecting pyyaml>=5.4.1
  Downloading PyYAML-6.0-cp310-cp310-win_amd64.whl (151 kB)
    | 151 kB 36 kB/s
Collecting python-dateutil>=2.5.3
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
    | 247 kB 39 kB/s
Collecting websocket-client!=0.40.0,!=0.41.*,!=0.42.*,>=0.32.0
  Downloading websocket_client-1.2.3-py3-none-any.whl (53 kB)
    | 53 kB 57 kB/s
Collecting six>=1.9.0
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting certifi>=14.05.14
  Downloading certifi-2021.10.8-py2.py3-none-any.whl (149 kB)
    | 149 kB 27 kB/s
Collecting urllib3>=1.24.2
  Downloading urllib3-1.26.8-py2.py3-none-any.whl (138 kB)
    | 138 kB 21 kB/s
Collecting requests
  Downloading requests-2.27.1-py2.py3-none-any.whl (63 kB)
    | 63 kB 42 kB/s
Collecting cachetools<6.0,>=2.0.0
  Downloading cachetools-5.0.0-py3-none-any.whl (9.1 kB)
Collecting pyasn1-modules>=0.2.1
  Downloading pyasn1_modules-0.2.8-py2.py3-none-any.whl (155 kB)
    | 155 kB 28 kB/s
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.8-py3-none-any.whl (39 kB)
Collecting pyasn1<0.5.0,>=0.4.6
  Downloading pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
    | 77 kB 53 kB/s
Collecting idna<4,>=2.5
  Downloading idna-3.3-py3-none-any.whl (61 kB)
    | 61 kB 44 kB/s
Collecting charset-normalizer>=2.0.0
  Downloading charset_normalizer-2.0.11-py3-none-any.whl (39 kB)
Collecting oauthlib>=3.0.0
  Downloading oauthlib-3.2.0-py3-none-any.whl (151 kB)
    | 151 kB 26 kB/s
Installing collected packages: urllib3, pyasn1, idna, charset-normalizer, certifi, six, rsa, requests, pyasn1-modules, oauthlib, cachetools, websocket-client, requests-oauthlib, pyyaml, python-dateutil, google-auth, kubernetes
Successfully installed cachetools-5.0.0 certifi-2021.10.8 charset-normalizer-2.0.11 google-auth-2.6.0 idna-3.3 kubernetes-21.7.0 oauthlib-3.2.0 pyasn1-0.4.8 pyasn1-modules-0.2.8 python-dateutil-2.8.2 pyyaml-6.0 requests-2.27.1 requests-oauthlib-1.3.1 rsa-4.8 six-1.16.0 urllib3-1.26.8 websocket-client-1.2.3
WARNING: You are using pip version 21.2.4, however, version 22.0.3 is available.
You should consider upgrading via the 'G:\PY\python.exe -m pip install --upgrade pip' command.
```

补充：默认使用 pip install 安装第三方库时，默认走的是 python 官网地址，下载会比较慢，可以使用如下方法，配置下载加速：

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple [软件] #清华源地址

# 将加速配置写到配置文件中
C:\Users\张岩峰>pip config set global.index-url
https://mirrors.aliyun.com/pypi/simple/
Writing to C:\Users\张岩峰\AppData\Roaming\pip\pip.ini

C:\Users\张岩峰>pip config set install.trusted-host mirrors.aliyun.com
Writing to C:\Users\张岩峰\AppData\Roaming\pip\pip.ini
```

第 12 章：文件及目录操作

在变量、序列和对象中存储的数据是暂时的，程序运行结束后就会消失。为了能够长时间地保存程序中的数据，需要将程序中的数据保存到磁盘文件中。

Python 提供了内置的文件对象和对文件、目录进行操作的内置模块。通过这些技术可以很方便地将数据保存到文件中，以达到长时间保存数据的目的。

1、基本文件操作

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

在 Python 中，内置了文件（File）对象。在使用文件对象时，首先需要通过内置的 open() 方法创建一个文件对象，然后通过该对象提供的方法进行一些基本文件操作。

1.1、创建和打开文件

在 Python 中，想要操作文件需要先创建或者打开指定的文件并创建文件对象。可以通过内置的 open() 函数实现。

open 函数的基本语法格式如下：

```
file = open(filename[,mode[,buffering]])
```

参数说明：

- file：被创建的文件对象。
- filename：要创建或打开文件的文件名称，需要使用单引号或双引号括起来。如果要打开的文件和当前文件在同一个目录下，那么直接写文件名即可，否则需要指定完整路径。
- mode：可选参数，用于指定文件的打开模式。
- buffering：可选参数，用于指定读写文件的缓存模式，值为 0 表示表达式不缓存。值为 1 表示表达式缓存。如果大于 1，则表示缓冲区的大小。默认缓存模式。

mode 参数的参数值如下所示：

值	说明	注意
r	以只读模式打开文件。文件的指针将会放在文件的开头。	文件必须存在
rb	以二进制格式打开文件，并且采用只读模式。文件的指针将会放在文件的开头。一般用于非文本文文件，如图片、声音等。	
r+	打开文件后，可以读取文件内容，也可以写入新的内容覆盖原有的内容（以文件开头进行覆盖）。	
rb+	以二进制格式打开文件，并且采用读写模式。文件的指针将会放在文件的开头。一般用于非文本文文件，如图片、声音等。	
w	以只写模式打开文件。	文件存在，则将其覆盖，否则创建新文件
wb	以二进制格式打开文件，并且采用只写模式。一般用于非文本文文件，如图片、声音等。	
w+	打开文件后，先清空原有内容，使其变为一个空的文件，对这个空文件有读写权限。	
wb+	以二进制格式打开文件，并且采用读写模式。一般用于非文本文文件，如图片、声音等。	
a	以追加模式打开一个文件。如果该文件已经存在，文件指针将放在文件末尾（即新内容会被写入到已有内容之后），否则，创建新文件用于写入。	

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

ab	以二进制格式打开文件，并且采用追加模式。如果该文件已经存在，文件指针将放在文件末尾（即新内容会被写入到已有内容之后），否则，创建新文件用于写入。	
a+	以读写模式打开文件。如果该文件已经存在，文件指针将放在文件末尾（即新内容会被写入到已有内容之后），否则，创建新文件用于读写。	
ab+	以二进制格式打开文件，并且采用追加模式。如果该文件已经存在，文件指针将放在文件末尾（即新内容会被写入到已有内容之后），否则，创建新文件用于读写。	

实例 1：创建一个空文件

说明：只能在当前目录创建一个文件

代码如下：

```
#!/usr/bin/env python
# *_ coding:utf-8 *_

file = open("time.txt","w")    #创建一个 time.txt 文件
```

运行结果：



注意：在调用 open() 函数时，指定 mode 的参数值为 w、w+、a、a+。这样，当要打开的文件不存在时，就可以创建新的文件了。如果当文件不存在，并且 mode 参数值为 r 这样的，就会发生错误。（见错误 1）

实例 2：打开文件时指定编码方式

说明：在使用 open() 函数打开文件时，默认采用 GBK 编码，当被打开的文件不是 GBK 编码时，将抛出异常。

代码如下：

```
file = open("doc.txt","r",encoding="utf-8")
#读取 doc.txt 文件，doc.txt 文件编码为 UTF-8，内容为张岩峰
print(file.read())
```

运行结果：

张岩峰

实例 3：以二进制形式打开文件

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

说明：使用 `open()` 函数不仅可以以文本的形式打开文件，而且还可以以二进制形式打开非文本文件，如图片文件、音频文件、视频文件等。

代码如下：

```
file = open("pub.png", "rb")          #以二进制格式打开 pub.png 图片文件
print(file)
```

运行结果：

```
<_io.BufferedReader name='pub.png'>
```

错误 1：打开一个不存在的文件

代码如下：

```
file = open("doc.txt", "r")
```

运行结果：

```
Traceback (most recent call last):
  File "E:\test.py", line 1, in <module>
    file = open("doc.txt", "r")
FileNotFoundError: [Errno 2] No such file or directory: 'doc.txt'
```

1.2、关闭文件

打开文件后，需要及时关闭，以免对文件造成不必要的破坏。关闭文件可以使用文件对象的 `close()` 方法实现。

`close()` 方法的语法格式如下：

```
file.close()
```

参数说明：

● `file`：file 为打开的文件对象。

提示：

`close()` 方法先刷新缓存区中还没有写入的信息，然后再关闭文件，这样可以将没有写入到文件的内容写入到文件中。在关闭文件后，便不能再进行写入操作了。

实例 1：读取 doc.txt 文件，并关闭。

代码如下：

```
file = open("doc.txt", "r", encoding = "utf-8")          # doc.txt 文件
                                                           编码为 UTF-8，内容为好好学习
print(file.read())
print("关闭前：", file.closed)
file.close()
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print("关闭后：",file.closed)
运行结果：
好好学习
关闭前： False
关闭后： True
# True 表示文件已经关闭，False 表示没有关闭。
```

1.3、打开文件时使用 with 语句

打开文件后，要及时将其关闭，如果忘记关闭可能会带来意想不到的问题。另外，如果在打开文件时抛出了异常，那么将导致文件不能被及时关闭。为了更好地避免此类问题发生，可以使用 Python 提供的 with 语句，从而实现在处理文件时，无论是否抛出异常，都能保证 with 语句执行完毕后关闭已经打开的文件。

with 语句的基本语法格式如下：

```
with expression as target:
    with-body
参数说明：
● expression：用于指定一个表达式，这里可以是打开文件的 open() 函数。
● target：用于指定一个变量，并且将 expression 的结果保存到该变量中。
● with-body：用于指定 with 语句体，其中可以是执行 with 语句后相关的一些操作语句。如果不想执行任何语句，可以直接使用 pass 语句代替。
```

实例 1：使用 with 打开 doc.txt 文件。

```
代码如下：
with open("doc.txt","r",encoding = "utf-8") as file:           #
doc.txt 文件编码为 UTF-8，内容为好好学习
    print(file.read())
print("关闭了吗？",file.closed)

运行结果如下：
好好学习
关闭了吗？ True
```

1.4、写入文件内容

我们虽然创建并打开一个文件，但是该文件中并没有任何内容，它的大小是 0KB。Python 的文件对象提供了 write() 方法，可以向文件中写入内容。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

write() 方法的语法格式如下：

```
file.write(string)
```

参数说明：

- file 为打开的文件对象。
- string 为要写入的字符串。

注意：在调用 write() 方法向文件中写入内容的前提是在打开文件时，指定的打开模式为 w(可写) 或者 a(追加)，否则，将抛出如下异常错误：

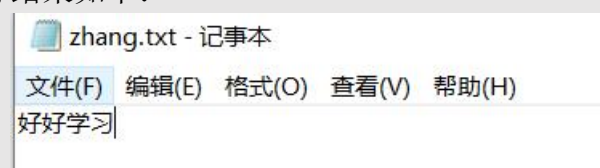
```
Traceback (most recent call last):
  File "E:\test.py", line 2, in <module>
    file.write("天天向上")
io.UnsupportedOperation: not writable
```

实例 1：创建 zhang.txt 文件，并写一条“好好学习”的信息

源码如下：

```
file = open("zhang.txt", "w", encoding = "utf-8")
file.write("好好学习")
file.close()
```

执行结果如下：



注意：在写入文件后，一定要调用 close() 方法关闭文件，否则写入的内容不会保存到文件中。这是因为当我们在写入文件内容时，操作系统不会立即把数据写入磁盘，而是先缓存起来，只有调用 close() 方法时，操作系统才会保证把没有写入的数据全部写入磁盘。

在向文件中写入内容后，如果不想马上关闭文件，也可以调用文件对象提供的 flush() 方法，把缓冲区的内容写入文件，这样也能保证数据全部写入磁盘。

实例 2：创建 zhang.txt 文件，并写一条“好好学习”的信息，使用 flush() 把缓冲区的内容写入文件

源码如下：

```
file = open("zhang.txt", "w", encoding = "utf-8")
file.write("好好学习")
file.flush()
```

执行结果如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



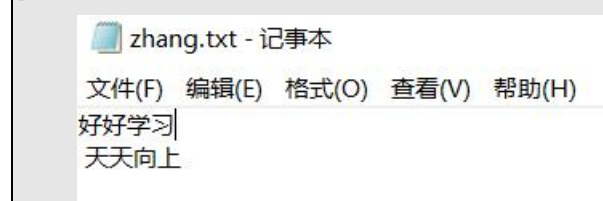
注意：向文件中写内容时，如果打开文件采用 w(写入) 模式，则先清空原文件中的内容，再写入新的内容；而如果打开文件采用 a(追加) 模式，则不覆盖原有文件的内容，只是在文件的结尾处增加新的内容。

实例 3：根据实例 1，向结尾追加“天天向上”的信息。

源码如下：

```
file = open("zhang.txt", "a", encoding = "utf-8")
file.write("\n 天天向上")
file.close()
```

执行结果如下：



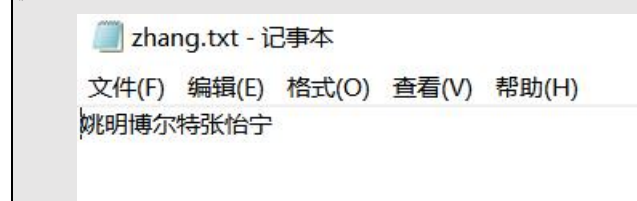
在 Python 的文件对象中除了提供了 write() 方法，还提供了 writelines() 方法，可以实现把字符串列表写入文件，但是不添加换行符。

实例 4：使用 writelines() 方法，把字符串列表写入文件。

源码如下：

```
list1 = ["姚明", "博尔特", "张怡宁"]
with open("zhang.txt", "w", encoding = "utf-8") as file:
    file.writelines(list1)
```

执行结果如下：



实例 5：根据实例 4，把字符串列表导入文件，添加换行符。

源码如下：

```
list1 = ["姚明", "博尔特", "张怡宁"]
with open("zhang.txt", "w", encoding = "utf-8") as file:
    file.writelines([line + "\n" for line in list1])
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

执行结果如下：



1.5、读取文件

在 Python 中打开文件后，除了可以向其写入或追加内容，还可以读取文件中的内容。读取文件内容主要分为以下几种情况：

1、读取指定字符

文件对象提供了 `read()` 方法读取指定个数的字符

`read()` 方法语法格式如下：

```
file.read(size)
参数说明：
● file：为打开的文件对象。
● size：可选参数，用于指定要读取的字符个数，如果省略，则一次性读取所有内容。
```

注意：在调用 `read()` 方法读取文件内容的前提是在打开文件时，指定的打开模式为 `r` (只读) 或者 `r+` (读写)，否则，将抛出如下所示的异常：

```
Traceback (most recent call last):
  File "E:\test.py", line 2, in <module>
    string = file.read(2)
io.UnsupportedOperation: not readable
```

实例 1：读取 `zhang.txt` 文件中的前 5 个字符

```
E:\>more zhang.txt
好好学习，天天向上

源码如下：
with open("zhang.txt","r") as file:
    string = file.read(5)
    print(string)

执行结果如下：
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

好好学习，

实例 2：读取 zhang.txt 文件的全部内容

```
E:\>more zhang.txt  
好好学习，天天向上
```

源码如下：

```
with open("zhang.txt","r") as file:  
    string = file.read()  
    print(string)
```

执行结果如下：

```
好好学习，天天向上
```

使用 read(size) 方法读取文件时，是从文件的开头读取的。如果想要读取部分内容，可以先使用文件对象的 seek() 方法将文件的指针移动到新的位置，然后再应用 read(size) 方法读取。

seek() 方法的基本语法格式如下：

```
file.seek(offset[, whence])
```

参数说明：

- file：表示已经打开的文件对象。
- offset：用于指定移动的字符个数，其具体位置与 whence 参数有关。
- whence：用于指定从什么位置开始计算。值为 0 表示从文件头开始计算，值为 1 表示从当前位置开始计算，值为 2 表示从文件尾开始计算，默认为 0。

注意：对于 whence 参数，如果在打开文件时，没有使用 b 模式（即 rb），那么只允许从文件头开始计算相对位置，从文件尾计算时就会抛出如下异常：

```
Traceback (most recent call last):  
  File "E:\test.py", line 2, in <module>  
    file.seek(8, 2)  
io.UnsupportedOperation: can't do nonzero end-relative seeks
```

说明：在使用 seek() 方法时，如果采用 GBK 编码，那么 offset 的值是按一个汉字（包括中文标点符号）占两个字符计算，而采用 UTF-8 编码，则一个汉字占 3 个字符，不过无论采用何种编码英文和数字都是按一个字符计算的。这与 read(size) 方法不同。

实例 1：从第 4 个字符后开始读取，读取 5 个字符

```
E:\>more zhang.txt  
好好学习，天天向上
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

源码如下：

```
with open("zhang.txt", "r") as file:
    file.seek(8)
    string = file.read(5)
    print(string)
```

执行结果如下：

，天天向上

2、读取一行

在使用 read() 方法读取文件时，如果文件很大，一次读取全部内容到内存，容易造成内存不足，所以通常会采用逐行读取。文件对象提供了 readline() 方法用于每次读取一行数据。

readline() 方法的基本语法格式如下：

```
file.readline()
```

参数说明：

- file 为打开的文件对象。同 read() 方法一样，打开文件时，也需要指定打开模式为 r（只读）或者 r+（读写）

实例 1：逐行读取

```
E:\>more zhang.txt
好好学习
天天向上
```

源码如下：

```
with open("zhang.txt", "r") as file:
    number = 0
    while True:
        number += 1
        line = file.readline()
        if line == '':
            break
        print(number, line, end="\n")
```

执行结果如下：

1 好好学习

2 天天向上

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

3、读取全部行

读取全部行的作用同调用 `read()` 方法时不指定 `size` 类似，只不过读取全部行时，返回的是一个字符串列表，每个元素为文件的一行内容。

读取全部行，使用的是文件对象的 `readlines()` 方法，其语法格式如下：

```
file.readlines()
```

参数说明：

- `file` 为打开的文件对象。同 `read()` 方法一样，打开文件时，也需要指定打开模式为 `r`（只读）或者 `r+`（读写）。

实例 1：读取 `zhang.txt` 文件的全部内容

```
E:\>more zhang.txt  
好好学习  
天天向上
```

源码如下：

```
with open("zhang.txt","r") as file:  
    message = file.readlines()  
    print(message)
```

执行结果如下：

```
['好好学习\n', '天天向上']
```

实例 2：逐行读取 `zhang.txt` 文件的全部内容

```
E:\>more zhang.txt  
好好学习  
天天向上
```

源码如下：

```
with open("zhang.txt","r") as file:  
    message = file.readlines()  
    for mess in message:  
        print(mess)
```

执行结果如下：

```
好好学习  
  
天天向上
```

2、目录操作

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

目录也称文件夹，用于分层保存文件。通过目录可以分门别类地存放文件。我们也可以通过目录快速找到想要的文件。在 Python 中，并没有提供直接操作目录的函数或者对象，而是需要使用内置的 os 和 os.path 模块实现。

注意：os 模块是 Python 内置的与操作系统功能和文件系统相关的模块。该模块中的语句的执行结果通过与操作系统有关，在不同操作系统上运行，可能会得到不一样的结果。

2.1、os 和 os.path 模块

在使用 os 模块或者 os.path 模块时，需要先应用 import 语句将其导入，然后才可以应用它们提供的函数或者变量。

导入 os 模块可以使用下面的代码：

```
import os
```

导入 os 模块后，也可以使用其子模块 os.path。

实例 1：获取操作系统类型

源码如下：

```
import os
print(os.name)
```

执行结果如下：

```
nt
```

说明：os.name 的输出结果为 nt，则表示是 Windows 操作系统；如果是 posix，则表示是 Linux、Unix 或 Mac OS 操作系统。

实例 2：获取当前操作系统所使用的路径分隔符

源码如下：

```
import os
print(os.sep)
```

执行结果如下：

```
\
```

os 模块提供的与目录相关的函数，如下表所示：

函数	说明
getcwd()	返回当前的工作目录
listdir(path)	返回指定路径下的文件和目录信息
makedirs(path [, mode])	创建目录
makedirs(path1/path2... [, mode])	创建多级目录

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

rmdir(path)	删除目录
removedirs(path1/path2……)	删除多级目录
chdir (path)	把 path 设置为当前工作目录
walk(top[, topdown[, onerror]])	遍历目录树，该方法返回一个元组，包括所有路径名、所有目录列表和文件列表 3 个元素

os.path 模块提供的与目录相关的函数，如下表所示：

函数	说明
abspath(path)	用于获取文件或目录的绝对路径
exists(path)	用于判断目录或者文件是否存在，如果存在则返回 True，否则返回 False
join(path, name)	将目录与目录或者文件名拼接起来
splitext()	分离文件名和扩展名
basename(path)	从一个目录中提取文件名
dirname(path)	从一个路径中提取文件路径，不包括文件名
isdir(path)	用于判断是否为路径

2.2、路径

用于定位一个文件或者目录的字符串被称为一个路径。在程序开发时，通常涉及两种路径，一种是相对路径，另一种是绝对路径。

1、相对路径

在学习相对路径之前，首先需要了解什么是当前工作目录。当前工作目录是指当前文件所在的目录。

在 Python 中，可以通过 os 模块提供的 getcwd() 函数获取当前工作目录。

实例 1：获取当前工作目录

代码如下：

```
import os
print(os.getcwd())
```

执行结果如下：

```
E:\
```

如果在当前工作目录下，有一个子目录 demo，并且在该子目录下保存着文件 more.txt，那么在打开并访问这个文件。

实例 2：打开并访问文件

代码如下：

```
import os
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
with open("demo/more.txt") as file:      # 通过相对路径打开文件
    string = file.read()
    print(string)
```

执行结果如下：

好好学习，天天向上

说明：在 Python 中，指定文件路径时需要对路径分隔符“\”进行转义，即将路径中的“\”替换为“\\”。例如对于相对路径“demo\more.txt”需要使用“demo\\more.txt”代替。另外，也可以将路径分隔符“\”采用“/”代替。

在指定文件路径时，也可以在表示路径的字符串前面加上字母 r（或 R），那么该字符串原样输出，这时路径中的分隔符就不需要转义了。例如，上面的代码可以修改为以下内容：

```
import os
with open(r"demo\more.txt") as file:      # 通过相对路径打开文件
    string = file.read()
    print(string)
```

2、绝对路径

绝对路径是指在使用文件时指定文件的实际路径。它不依赖于当前工作目录。在 Python 中，可以通过 os.path 模块提供的 abspath() 函数获取一个文件的绝对路径。

abspath() 函数的基本语法格式如下：

```
os.path.abspath(path)
```

参数说明：

- path 为要获取绝对路径的相对路径，可以是文件也可以是目录。

实例 1：要获取相对路径“demo\more.txt”的绝对路径，可以使用下面的代码：

代码如下：

```
import os
print(os.path.abspath(r"demo\more.txt"))
```

执行结果如下：

E:\demo\more.txt

3、拼接目录

如果想要将两个或者多个路径拼接到一起组成一个新的路径，可以使用 os.path 模块提供的 join() 函数实现。

join() 函数基本语法格式如下：

```
os.path.join(path1[, path2[, .....]])
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

其中，path1、path2 用于代表要拼接的文件路径，这些路径间使用逗号进行分隔。如果在要拼接的路径中，没有一个绝对路径，那么最后拼接出来的将是一个相对路径。

注意：使用 os.path.join() 函数拼接路径是，并不会检测该路径是否真实存在。

实例 1：将“E:\program\Python\Code”和“demo\more.txt”路径拼接到一起源码如下：

```
import os
print(os.path.join(r"E:\program\Python\Code", r"demo\more.txt"))
```

执行结果如下：

```
E:\program\Python\Code\demo\more.txt
```

实例 2：拼接多个绝对路径

说明：在使用 join() 函数时，如果要拼接的路径中，存在多个绝对路径，那么以从左到右为序最后一次出现的路径为准，并且该路径之前的参数都将被忽略。

源码如下：

```
import os
print(os.path.join("E:\\code", "E:\\python\\mr", "code", "C:\\", "demo"))
```

执行结果如下：

```
C:\demo
```

2.3、判断目录或文件是否存在

在 Python 中，有时需要判断给定的目录是否存在，这时可以使用 os.path 模块提供的 exists() 函数实现。

exists() 函数的基本语法格式如下：

```
os.path.exists(path)
```

其中，path 为要判断的目录，可以采取绝对路径，也可以采取相对路径。

返回值：如果给定的路径存在，则返回 True，否则返回 False。

提示：os.path.exists() 函数除了可以判断目录是否存在，还可以判断文件是否存在。

实战 1：判断 C:\demo 是否存在

源码如下：

```
import os
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(os.path.exists(r"C:\demo"))
```

执行结果如下：

```
False
```

实战 2：判断目录或文件是否存在，如果文件存在则输出“该文件或目录存在”，如果文件不存在则输出“该文件或目录不存在”

源码如下：

```
if os.path.exists(r"D:\k8s 架构师课程\第二章、Python 入门到高级进阶\python\13"):  
    print("该文件或目录存在")  
else:  
    print("该文件或目录不存在")
```

2.4、创建目录

在 Python 中，os 模块提供了两个创建目录的函数，一个用于创建一级目录，另一个用于创建多级目录。

1、创建一级目录

创建一级目录是指一次只能创建一级目录。在 Python 中，可以使用 os 模块提供的 mkdir() 函数实现。通过该函数只能创建指定路径中的最后一级目录，如果该目录的上一级不存在，则抛出 FileNotFoundError 异常。

mkdir() 函数的基本语法格式如下：

```
os.mkdir(path, mode=0777)
```

参数说明：

- path：用于指定要创建的目录，可以使用绝对路径，也可以使用相对路径。
- mode：用于指定数值模式，默认值为 0777. 该参数在非 UNIX 系统上使用会被无效或被忽略。

实战 1：创建 C:\\demo

源码如下：

```
import os  
os.mkdir("C:\\demo")
```

实战 2：判断文件夹是否存在，如果存在就不创建，如果不存在就创建

说明：根据实战 1，如果实战 1 的代码重复执行，就会发生以下错误：

```
Traceback (most recent call last):
```

```
File "E:\test.py", line 2, in <module>
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
os.mkdir("C:\\demo")
FileExistsError: [WinError 183] 当文件已存在时，无法创建该文件。:
'C:\\demo'
```

源码如下：

```
import os
path = "C:\\demo"          # 指定要创建的目录
if os.path.exists(path):    # 判断目录是否存在
    print("目录已经存在")
else:
    os.mkdir(path)
    print("目录创建成功")
```

2、创建多级目录

使用 `mkdir()` 函数只能创建一级目录，如果想要创建多级目录，可以使用 `os` 模块提供的 `makedirs()` 函数，该函数用于采用递归的方式创建目录。

`makedirs()` 函数的基本语法格式如下：

```
os.makedirs(name, mode=0777)
```

参数说明：

- `name`：用于指定要创建的目录，可以使用绝对路径，也可以使用相对路径。
- `mode`：用于指定数值模式，默认值为 0777. 该参数在非 UNIX 系统上使用会被无效或被忽略。

实战 1：递归创建 E：

源码如下：

```
import os
os.makedirs(r"E:\q1\q2\q3")
```

2.5、删除目录

删除目录可以通过使用 `os` 模块提供的 `rmdir()` 函数实现。通过 `rmdir()` 函数删除目录时，只有当要删除的目录为空时才起作用。

`rmdir()` 函数的基本语法如下：

```
os.rmdir(path)
```

其中，`path` 为要删除的目录，可以使用相对路径，也可以使用绝对路径

实例 1：删除 "c:\q1"

源码如下：

```
import os
os.rmdir(r"C:\q1")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

实战 2：判断文件是否存在，如果存在再删除“c:\q1”

源码如下：

```
import os
path = (r"C:\q1")
if os.path.exists(path):
    os.rmdir(path)
    print("目录已经删除")
else:
    print("该目录不存在！")
```

使用 rmdir() 函数只能删除空的目录，如果想要删除非空目录，则需要使用 Python 内置的标准模块 shutil 的 rmtree() 函数实现

实例 3：删除“C:\q1\q2\q3”删除 q1 包括下面的子目录

源码如下：

```
import shutil
import os
path = (r"C:\q1")
if os.path.exists(path):
    shutil.rmtree(path)
    print("目录已经删除")
else:
    print("该目录不存在！")
```

2.6、遍历目录

在 Python 中，遍历是将指定的目录下的全部目录（包括子目录）及文件访问一遍。在 Python 中，os 模块的 walk() 函数用于实现遍历目录的功能。

walk() 函数的基本语法格式如下：

```
os.walk(top[, topdown][, onerror][, followlinks])
```

参数说明：

- top：用于指定要遍历内容的根目录。
- topdown：可选参数，用于指定遍历的顺序，如果值为 True，表示自上而下遍历（即先遍历根目录）；如果值为 False，表示自下而上遍历（即先遍历最后一级的子目录）。默认值为 True。
- onerror：可选参数，用于指定错误处理方式，默认为忽略，如果不想忽略也可以指定一个错误处理函数。通常情况下采用默认设置。
- followlinks：可选参数，默认情况下，walk() 函数不会向下转换成解析到目录的符号链接，将该参数设置为 True，表示用于指定在支持的系统上访问由符号链接指向的目录。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

● 返回值：返回一个包括 3 个元素（dirpath, dirnames, filenames）的元组生成器对象。其中，dirpath 表示当前遍历的路径，是一个字符串；dirnames 表示当前路径下包含的子目录，是一个列表；filenames 表示当前路径下包含的文件，也是一个列表。

实例 1：遍历 E:\zhang

源码如下：

```
import os
tuples = os.walk(r"E:\zhang")
for tuple1 in tuples:
    print(tuple1, "\n")
```

执行结果如下：

```
('E:\zhang', ['q1'], ['more.txt', 'qq.txt', 'text.txt'])
#遍历的文件夹    遍历文件夹下的文件夹    遍历文件夹下的文件
('E:\zhang\q1', [], ['more.txt', 'test.txt'])
```

实例 2：遍历 E:\zhang

源码如下：

```
import os    # 导入 os 模块
path = r"E:\zhang"    # 指定要遍历的根目录
print("【", path, "】 目录下包括的文件和目录：")
for root, dirs, files in os.walk(path, topdown=True):    # 遍历指定
目录
    for name in dirs:    # 循环输出遍历到的子目录
        print("●", os.path.join(root, name))
    for name in files:    # 循环输出遍历到的文件
        print("◎", os.path.join(root, name))
```

执行结果如下：

```
【 E:\zhang 】 目录下包括的文件和目录：
● E:\zhang\q1
◎ E:\zhang\more.txt
◎ E:\zhang\qq.txt
◎ E:\zhang\text.txt
◎ E:\zhang\q1\more.txt
◎ E:\zhang\q1\test.txt
```

3、高级文件操作

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Python 内置的 os 模块除了可以对目录进行操作，还可以对文件进行一些高级操作，具体函数如下表所示：

os 模块提供的与文件相关的函数	
函数	说明
access(path, accessmode)	获取对文件是否有指定的访问权限（读取 / 写入 / 执行权限）。accessmode 的值是 R_OK（读取）、W_OK（写入）、X_OK（执行）或 F_OK（存在）。如果有指定的权限，则返回 1，否则返回 0
chmod(path, mode)	修改 path 指定文件的访问权限
remove(path)	删除 path 指定的文件路径
rename(src, dst)	将文件或目录 src 重命名为 dst
stat(path)	返回 path 指定文件的信息
startfile(path[, operation])	使用关联的应用程序打开 path 指定的文件

3.1、删除文件

Python 没有内置删除文件的函数，但是在内置的 os 模块中提供了删除文件的函数 remove()

remove() 函数的基本语法格式如下：

<pre>os.remove(path)</pre> <p>其中，path 为要删除的文件路径，可以使用相对路径，也可以使用绝对路径。</p>

实战 1：删除"E:\zhang\qq.txt"

<p>源码如下：</p> <pre>import os path = r"E:\zhang\qq.txt" if os.path.exists(path): os.remove(path) print("文件删除完毕！") else: print("文件不存在")</pre> <p>执行结果如下：</p> <p>文件删除完毕！</p>
--

3.2、重命名文件或目录

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

os 模块提供了重命名文件和目录的函数 `rename()`，如果指定的路径是文件的，则重命名文件，如果指定的路径是目录，则重命名目录。

`rename()` 函数的基本语法格式如下：

```
os.rename(src, dst)
其中，src 用于指定要进行重命名的目录或文件；dst 用于指定重命名后的目录或文件。
```

实战 1：将“E:\zhang”修改为“E:\zhang1”

```
源码如下：
import os
src = r"E:\zhang"
dst = r"E:\zhang1"
if os.path.exists(src):
    os.rename(src, dst)
    print("文件重命名完毕")
else:
    print("文件不存在")

执行结果如下：
文件重命名完毕
```

3.3、获取文件基本信息

在计算机上创建文件后，该文件本身就会包含一些信息。例如，文件的最后一次访问时间、最后一次修改时间、文件大小等基本信息。通过 os 模块的 `stat()` 函数可以获取到文件的这些基本信息。

`stat()` 函数的基本语法如下：

```
os.stat(path)
其中，path 为要获取文件基本信息的文件路径，可以是相对路径，也可以是绝对路径。
```

`stat()` 函数的返回值是一个对象，该对象包含如下表的属性。通过访问这些属性可以获取文件的基本信息：

属性	说明
st_mode	保护模式
st_ino	索引号
st_nlink	硬链接号（被连接的数目）
st_size	文件大小，单位为字节
st_mtime	最后一次修改时间
st_dev	设备名

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

st_uid	用户 ID
st_gid	组 ID
st_atime	最后一次访问时间
st_ctime	最后一次状态变化的时间（系统不同返回结果也不同，例如，在 Windows 操作系统下返回的是文件的创建时间）

实例 1：获取 test.py 基本信息

源码如下：

```
import os    # 导入 os 模块
fileinfo = os.stat(r"E:\test.py") # 获取文件的基本信息
print("文件完整路径：", os.path.abspath("test.py")) # 获取文件的完整路径
# 输出文件的基本信息
print("索引号：", fileinfo.st_ino)
print("设备名：", fileinfo.st_dev)
print("文件大小：", fileinfo.st_size, " 字节")
print("最后一次访问时间：", fileinfo.st_atime)
print("最后一次修改时间：", fileinfo.st_mtime)
print("最后一次状态变化时间：", fileinfo.st_ctime)
```

执行结果如下：

```
文件完整路径： E:\test.py
索引号： 2251799813694899
设备名： 3932587287
文件大小： 2155 字节
最后一次访问时间： 1564193457.41277
最后一次修改时间： 1564193457.41277
最后一次状态变化时间： 1564013209.067524
```

实例 2：

由于上面的结果中的时间和字节数都是一长串的整数，与我们平时见到的有所不同，所以一般情况下，为了显示更加直观，还需要对这样的数字进行格式化。

源码如下：

```
import os    # 导入 os 模块

def formatTime(longtime):
    ''' 格式化日期时间的函数
        longtime: 要格式化的时间
    '''
    import time # 导入时间模块
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
        return
time.strftime('%Y-%m-%d %H:%M:%S',time.localtime(longtime))

if __name__ == '__main__':
    fileinfo = os.stat(r"I:\soft\1\bmi.py") # 获取文件的基本信息
    print("文件完整路径:", os.path.abspath(r"I:\soft\1\bmi.py")) #
获取文件的完整数路径
    # 输出文件的基本信息
    print("索引号:",fileinfo.st_ino)
    print("设备名:",fileinfo.st_dev)
    print("文件大小:",fileinfo.st_size)
    print("最后一次访问时间:",formatTime(fileinfo.st_atime))
    print("最后一次修改时间:",formatTime(fileinfo.st_mtime))
    print("最后一次状态变化时间:",formatTime(fileinfo.st_ctime))
```

执行结果如下:

文件完整路径: E:\test.py
索引号: 2251799813694899
设备名: 3932587287
文件大小: 1450
最后一次访问时间: 2019-07-27 10:16:16
最后一次修改时间: 2019-07-27 10:16:16
最后一次状态变化时间: 2019-07-25 08:06:49

第 13 章：使用 Python 操作数据库

1、数据库编程接口

在项目开发中，数据库应用必不可少。虽然数据库的种类有很多，如 SQLite、MySQL、Oracle 等，但是它们的功能基本都是一样的，为了对数据库进行统一的操作，大多数语言都提供了简单的、标准化的数据库接口（API）。在 Python Database API 2.0 规范中，定义了 Python 数据库 API 接口的各个部分，如模块接口、连接对象、游标对象、类型对象和构造器、DB API 的可选扩展以及可选的错误处理机制等。

1.1、连接对象

数据库连接对象（Connection Object）主要提供获取数据库游标对象和提交、回滚事务的方法，以及关闭数据库连接。

1、获取连接对象

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

如何获取连接对象呢？这就需要使用 connect() 函数。该函数有多个参数，具体使用哪个参数，取决于使用的数据库类型。例如，需要访问 Oracle 数据库和 MySQL 数据库，则必须同时下载 Oracle 和 MySQL 数据库模块。这些模块在获取连接对象时，都需要使用 connect() 函数。

connect() 函数常用的参数及说明如下表所示：

参数	说明
dsn	数据源名称，给出该参数表示数据库依赖
user	用户名
password	用户密码
host	主机名
database	数据库名称

2、连接对象的方法

Connect() 函数返回连接对象。这个对象表示目前和数据库的会话，连接对象支持的方法如下表：

方法名	说明
close()	关闭数据库连接
commit()	提交事务
rollback()	回滚事务
cursor()	获取游标对象，操作数据库，如执行 DML 操作，调用存储过程等

commit() 方法用于提交事务，事务主要用于处理数据量大、复杂度高的数据。如果操作的是一系列动作，比如张三给李四转账，有如下两个操作：

- 张三账户金额减少
- 李四账户金额增加

这时使用事务可以维护数据库的完整性，保证 2 个操作要么全部执行，要么全部不执行。

1.2、游标对象

游标对象（Cursor Object）代表数据库中的游标，用于指示抓取数据操作的上下文，主要提供执行 SQL 语句、调用存储过程、获取查询结果等方法。

如何获取游标对象呢？通过使用连接对象的 cursor() 方法，可以获取到游标对象。游标对象的属性如下所示：

- description：数据库列类型和值的描述信息。
- rowcount：回返结果的行数统计信息，如 SELECT、UPDATE、CALLPROC 等。

游标对象的方法如下表：

方法名	说明
callproc(procname, [, parameters])	调用存储过程，需要数据库支持

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

<code>close()</code>	关闭当前游标
<code>execute(operation[, parameters])</code>	执行数据库操作，SQL 语句或者数据库命令
<code>executemany(operation, seq_of_params)</code>	用于批量操作，如批量更新
<code>fetchone()</code>	获取查询结果集中的下一条记录
<code>fetchmany(size)</code>	获取指定数量的记录
<code>fetchall()</code>	获取结构集的所有记录
<code>nextset()</code>	跳至下一个可用的结果集
<code>arraysize</code>	指定使用 <code>fetchmany()</code> 获取的行数，默认为 1
<code>setinputsizes(sizes)</code>	设置在调用 <code>execute*()</code> 方法时分配的内存区域大小
<code>setoutputsize(sizes)</code>	设置列缓冲区大小，对大数据列（如 LONGS 和 BLOBS）尤其有用

2、操作 SQLite

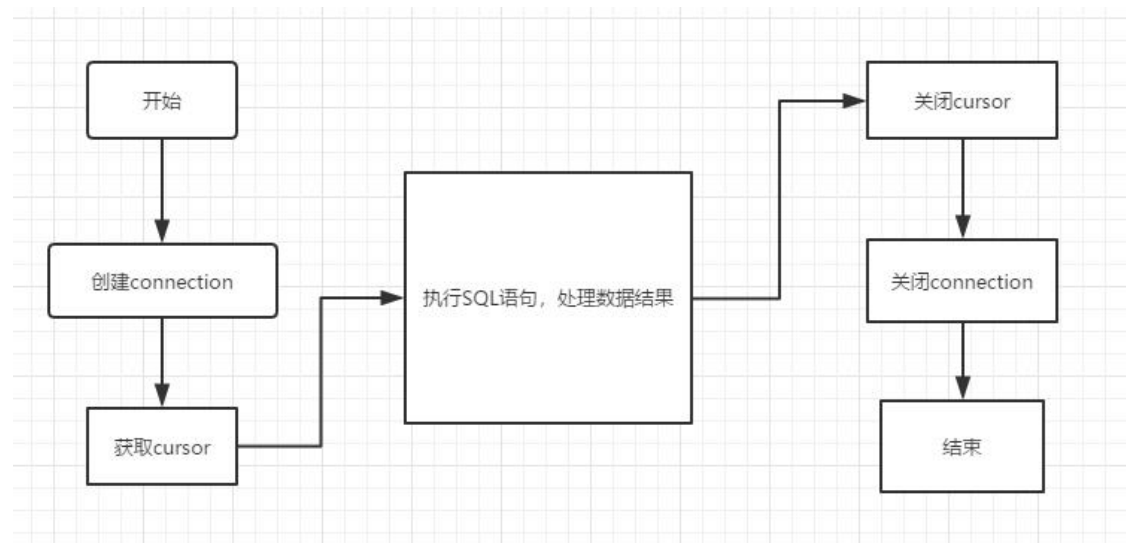
与许多其他数据库管理系统不同，SQLite 不是一个客户端/服务器结果的数据库引擎，而是一种嵌入式数据库，它的数据库就是一个文件。SQLite 将整个数据库，包括定义、表、索引以及数据本身，作为一个单独的、可跨平台使用的文件存储在主机中。由于 SQLite 本身是 C 写的，而且体积很小，所以，经常被集成到各种应用程序中。Python 就内置了 SQLite3，所以在 Python 中使用 SQLite，不需要安装任何模块，直接使用即可。

2.1、创建数据库文件

由于 Python 中已经内置了 SQLite3，所以可以直接使用 `import` 语句导入 SQLite3 模块。Python 操作数据库的通用的流程图如下所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



实例 1: 创建 SQLite 数据库文件

创建一个 mrsoft.db 的数据库文件，然后执行 SQL 语句创建一个 user（用户表），user 表包含 id 和 name 两个字段。

源码如下：

```
import sqlite3
# 连接到 SQLite 数据库
# 数据库文件是 mrsoft.db
# 如果文件不存在，会自动在当前目录创建:
conn = sqlite3.connect('mrsoft.db')
# 创建一个 Cursor:
cursor = conn.cursor()
# 执行一条 SQL 语句，创建 user 表:
cursor.execute('create table user (id int(10) primary key, name
varchar(20))')
# 关闭游标
cursor.close()
# 提交事务:
conn.commit()
# 关闭 Connection:
conn.close()
```

运行结果如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

名称	修改日期	类型	大小
__pycache__	2022/2/9 8:46	文件夹	
bmi.py	2022/2/9 14:11	Python File	1 KB
mrsoft.db	2022/2/11 16:21	Data Base File	12 KB
test.py	2022/2/11 16:21	Python File	1 KB

在当前目录生成了一个 mrsoft.db

2.2、操作 SQLite

1、新增用户数据信息

为了向数据表中新增数据，可以使用如下 SQL 语句：

```
insert into 表名(字段名 1, 字段名 2, ... , 字段名 n) values (字段值 1, 字段值 2, ... , 字段值 n)
```

实例 1：新增用户数据信息

我们已经创建了一个 mrsoft.db，所以本实战可以直接操作 user 表，向 user 表中插入 3 条用户信息。此外，由于是新增数据，需要使用 commit() 方法提交事务。因为对于增加、修改和删除操作，使用 commit() 方法提交事务后，如果相应操作失败，可以使用 rollback() 方法回滚到操作之前的状态。

源码如下：

```
import sqlite3
# 连接到 SQLite 数据库
# 数据库文件是 mrsoft.db
# 如果文件不存在，会自动在当前目录创建：
conn = sqlite3.connect('mrsoft.db')
# 创建一个 Cursor:
cursor = conn.cursor()
# 继续执行一条 SQL 语句，插入一条记录：
cursor.execute('insert into user (id, name) values ("1", "Feng")')
cursor.execute('insert into user (id, name) values ("2", "zyf")')
cursor.execute('insert into user (id, name) values ("3", "Zhang")')
# 关闭游标
cursor.close()
# 提交事务：
conn.commit()
# 关闭 Connection:
conn.close()
```

运行该实例，会向 user 表中插入 3 条记录。为验证程序是否正常运行，可以再次运行，如果提示如下信息，说明插入成功（因为 user 表中已经保存了上一次插入的记录，所以再次插入会报错）。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
sqlite3.IntegrityError: UNIQUE constraint failed: user.id
```

2、查看用户数据信息

查找 user 表中的数据可以使用如下 SQL 语句：

```
select 字段名 1, 字段名 2, 字段名 3... from 表名 where 查询条件
```

查看用户信息的代码与插入数据信息大致相同，不同点在于使用的 SQL 语句不同。此外，查询数据时通常使用如下 3 种方式：

- fetchone()：获取查询结果集中的下一条记录。
- fetchmany(size)：获取指定数量的记录。
- fetchall()：获取结果集的所有记录。

实例 1：使用 3 种方式查询用户数据信息

分别使用 fetchone、fetchmany、fetchall 这 3 种方式查询用户信息

源码如下：

```
import sqlite3
# 连接到 SQLite 数据库, 数据库文件是 mrsoft.db
conn = sqlite3.connect('mrsoft.db')
# 创建一个 Cursor:
cursor = conn.cursor()
# 执行查询语句:
cursor.execute('select * from user')
# 获取查询结果:
result1 = cursor.fetchone()
print(result1)

# 关闭游标
cursor.close()
# 关闭 Connection:
conn.close()
```

使用 fetchone() 方法返回的 result1 为一个元组，执行结果如下：

```
(1, 'Feng')
```

(1) 修改黄色区域获取查询结果的语句块代码为如下内容：

```
result2 = cursor.fetchmany(2)
print(result2)
```

使用 fetchmany() 方法传递一个参数，其值为 2，默认为 1。返回的 result2 为一个列表，列表中包含 2 个元组，运行结果如下：

```
[(1, 'Feng'), (2, 'zyf')]
```

(2) 修改黄色区域获取查询结果的语句块代码为如下内容：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
result3 = cursor.fetchall()
print(result3)
```

使用 `fetchall()` 方法返回的 `result3` 为一个列表，列表中包含所有 `user` 表中数据组成的元组，运行结果如下：

```
[(1, 'Feng'), (2, 'zyf'), (3, 'Zhang')]
```

(3) 修改黄色区域获取查询结果的语句块代码为如下内容：

```
cursor.execute('select * from user where id > ?', (1,))
result3 = cursor.fetchall()
print(result3)
```

在 `select` 查询语句中，使用问号作为占位符代替具体的数值，然后使用一个元组来替换问号（注意，不要忽略元组中最后的逗号）。上述查询结果等价于：

```
cursor.execute('select * from user where id > 1')
```

执行结果如下：

```
[(2, 'zyf'), (3, 'Zhang')]
```

3、修改用户数据信息

修改 `user` 表中的数据可以使用如下 SQL 语句：

```
update 表名 set 字段名 = 字段值 where 查询条件
```

实战 1：修改用户数据信息

源码如下：

```
import sqlite3
# 连接到 SQLite 数据库, 数据库文件是 mrsoft.db
conn = sqlite3.connect('mrsoft.db')
# 创建一个 Cursor:
cursor = conn.cursor()
cursor.execute('update user set name = ? where id = ?', ('Me', 1))
cursor.execute('select * from user')
result = cursor.fetchall()
print(result)
# 关闭游标
cursor.close()
# 提交事务
conn.commit()
# 关闭 Connection:
conn.close()
```

执行结果如下：

```
[(1, 'Me'), (2, 'zyf'), (3, 'Zhang')]
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

4、删除用户数据信息

删除 user 表中的数据可以使用如下 SQL 语句：

```
delete from 表名 where 查询条件
```

实战 1：删除用户数据信息

源码如下：

```
import sqlite3
# 连接到 SQLite 数据库, 数据库文件是 mrsoft.db
conn = sqlite3.connect('mrsoft.db')
# 创建一个 Cursor:
cursor = conn.cursor()
cursor.execute('delete from user where id = ?', (1,))
cursor.execute('select * from user')
result = cursor.fetchall()
print(result)
# 关闭游标
cursor.close()
# 提交事务
conn.commit()
# 关闭 Connection:
conn.close()
```

执行结果如下：

```
[(2, 'zyf'), (3, 'Zhang')]
```

3、Python 操作 MySQL

3.1、安装 MySQL

本文采用的是 Linux 系统下的 MySQL

操作方式如下：

```
[root@localhost ~]# yum -y install mariadb-server
[root@localhost ~]# systemctl enable mariadb --now
[root@localhost ~]# mysql
MariaDB [(none)]> create user root@'192.168.%.%' identified by
'zzl23..';
MariaDB [(none)]> grant all PRIVILEGES ON *.* to zyf@'192.168.%.%'
with grant option;
```

3.2、安装 PyMySQL

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

由于 MySQL 服务器以独立的进程运行，并通过网络对外服务，所以，需要支持 Python 的 MySQL 驱动来连接到 MySQL 服务器。在 Python 中支持 MySQL 的数据库模块有很多，我们选择使用 PyMySQL。

PyMySQL 的安装比较简单，在 cmd 执行如下命令：

```
pip install PyMySQL
```

运行结果如下图所示：

```
C:\Users\zhang>pip install PyMySQL
Collecting PyMySQL
  Downloading PyMySQL-1.0.2-py3-none-any.whl (43 kB)
    |#####| 43 kB 100 kB/s
Installing collected packages: PyMySQL
Successfully installed PyMySQL-1.0.2
WARNING: You are using pip version 21.2.4; however, version 22.0.3 is available.
You should consider upgrading via the 'G:\PY\python.exe -m pip install --upgrade pip' command.
```

3.3、程序环境安装数据库

MySQL 是一款开源的数据库软件，由于其免费特性得到了全世界用户的喜爱，是目前使用人数最多的数据库。

1、下载 MySQL

在浏览器的地址栏中输入

“<https://dev.mysql.com/downloads/installer/8.0.html>”，并按下回车键，将进入到最新版本 MySQL 8.0 的下载页面，选择离线下载，如下图所示：

MySQL Community Downloads

MySQL Installer

[General Availability \(GA\) Releases](#) [Archives](#) [Download](#)

MySQL Installer 8.0.28

Select Operating System:
Microsoft Windows

[Looking for previous GA versions?](#)

Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-8.0.28.0.msi)	8.0.28	2.3M	Download
Windows (x86, 32-bit), MSI Installer (mysql-installer-community-8.0.28.0.msi)	8.0.28	435.7M	Download

[MD5: 514567a7503999d271a20b86057f15d0](#) | [Signature](#)

[MD5: e1223cb4d7873a057d5aa0ab0fa596201](#) | [Signature](#)

[We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.](#)

ORACLE © 2022, Oracle Corporation and/or its affiliates

[Legal Policies](#) | [Your Privacy Rights](#) | [Terms of Use](#) | [Trademark Policy](#) | [Contributor Agreement](#) | [Cookie 喜好设置](#)

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

单击“download”按钮下载，进入开始下载页面，如果有 MySQL 的账户，可以单击 Login 按钮，登录账户后下载，如果没有可以直接单击下方的“No thanks, just start my download.”超链接，跳过注册步骤，直接下载，如下图所示：

MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

Login »
using my Oracle Web account

Sign Up »
for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

2、安装 MySQL

下载完成以后，开始安装 MySQL。双击安装文件，在安装界面中勾选“I accept the license terms”，点击“next”，进入选择设置类型界面。在选择设置中有 5 种类型，说明如下：

● **Developer Default**：安装 MySQL 服务器以及开发 MySQL 应用所需的工具。工具包括开发和管理服务器的 GUI 工作台、访问操作数据的 Excel 插件、与 Visual Studio 集成开发的插件、通过 NET/Java/C/C++/ODBC 等访问数据的连接器、例子和教程、开发文档。

● **Server only**：仅安装 MySQL 服务器，适用于部署 MySQL 服务器。

● **Client only**：仅安装客户端，适用于基于已存在的 MySQL 服务器进行 MySQL 应用开发的情况。

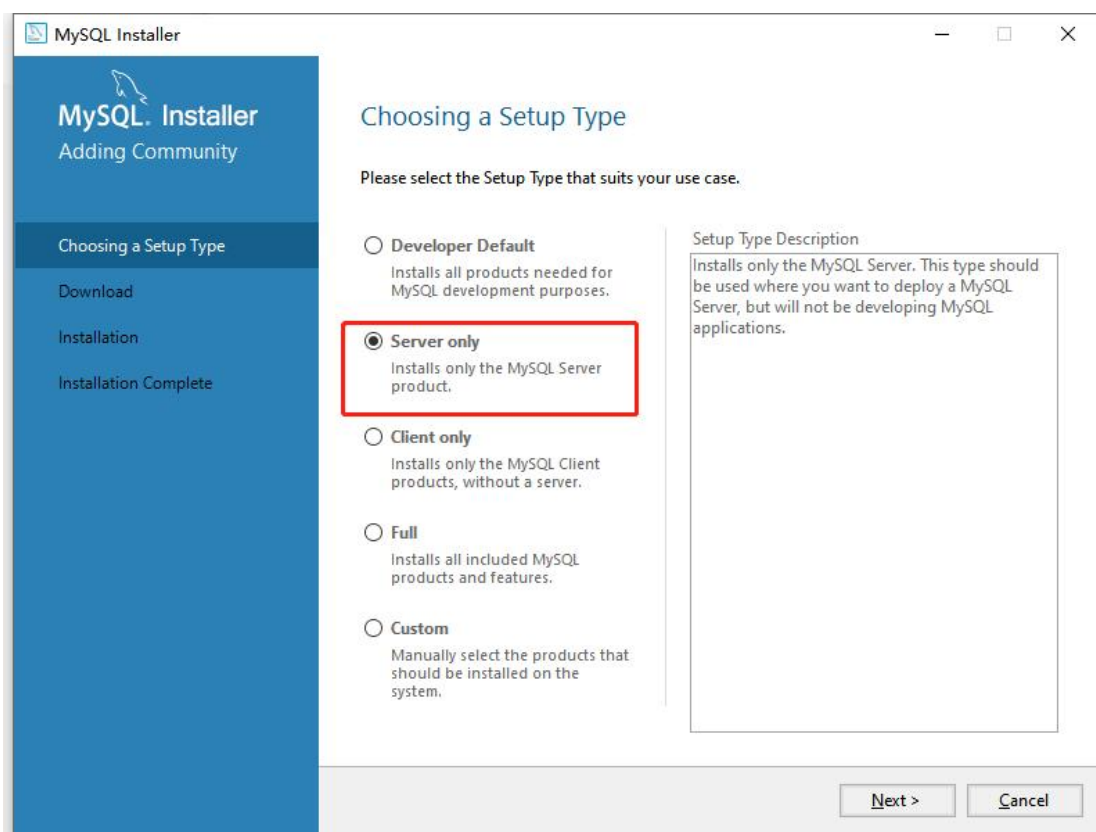
● **Full**：安装 MySQL 所有可用组件。

● **Custom**：自定义需要安装的组件。

MySQL 会默认选择“Developer Default”类型，这里选择“Server only”类型，如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



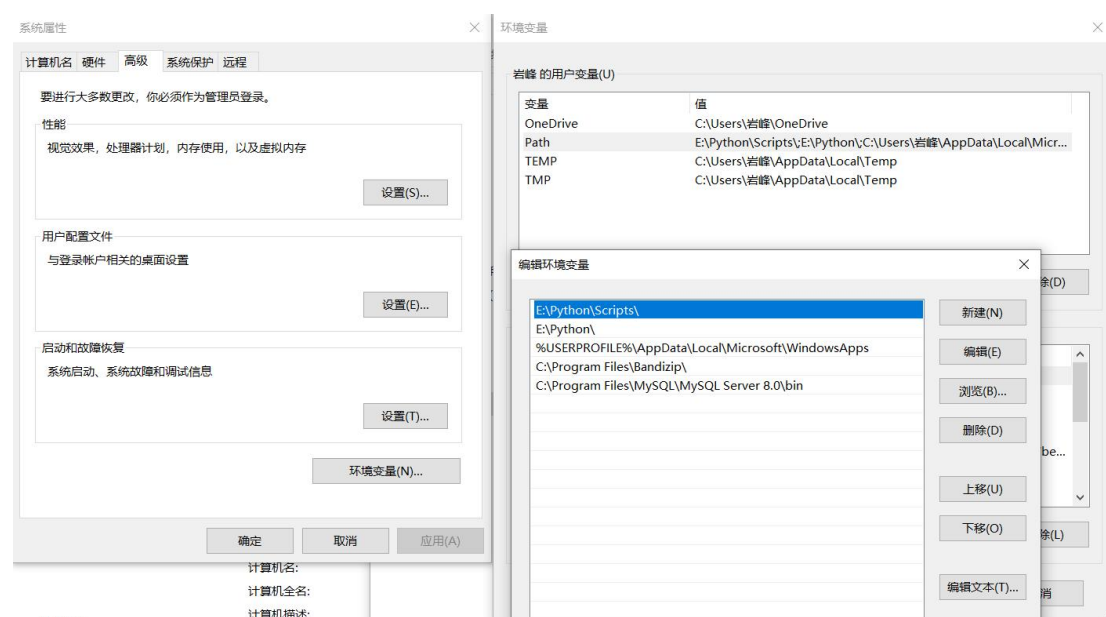
这里可以选择为“Client only”，开发环境连接数据库，是需要一个数据库的客户端工具的，否则连接会报错。

3、设置环境变量

安装完成以后，默认的安装路径是“C:\Program Files\MySQL\MySQL Server 8.0\bin”。下面设置环境变量，以便在任意目录下使用 MySQL 命令。右键单击“计算机”-》选择“属性”，打开控制面板主页，选择“高级系统设置”-》选择“环境变量”-》选择“PATH”-》选择“编辑”。将“C:\Program Files\MySQL\MySQL Server 8.0\bin”写在变量值中。如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



3.3、连接数据库

使用数据库的第一步是连接数据库。接下来使用 PyMySQL 连接数据库。由于 PyMySQL 也遵循 Python Database API 2.0 规范，所以操作 MySQL 数据库的方式与 SQLite 相似。

实例 1：使用 PyMySQL 连接数据库

源码如下：

```
# 1 导包
# 注意如果导入的 pymysql 是黄色的，那么有可能你的项目名称和 python
# 文件名称是 pymysql 导致导包导入错误
# 如果是一个波浪线，有可能没有选择运行环境和没有安装 pymysql
import pymysql
# 2 建立连接
db = pymysql.connect(host='192.168.13.253', user='root',
password='zz123..', port=3306)
# 3 获取游标
cursor = db.cursor()
# 4 执行 SQL 语句：执行查询数据库版本的 SQL 语句
# 注意 SQL 语句当中的符号都必须是英文的符号
cursor.execute("select version();")
# 打印执行的结果
# 获取执行结果
data = cursor.fetchone()
# 打印查询结果
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print ("Database version : %s " % data)
# 5 关闭游标
cursor.close()
# 6 关闭连接
db.close()
```

执行结果如下：

```
Database version : 5.7.37
```

3.4、创建数据库

创建数据库语法为：

```
create database [name]
```

实例 1：连接并创建 books 数据库

源码如下：

```
import pymysql

# 连接数据库
db = pymysql.connect(host='192.168.13.253', user='root',
password='zz123..', port=3306)
# 获取游标
cursor = db.cursor()

# 查询数据库，并获取执行结果
cursor.execute("show databases;")
data_1 = cursor.fetchall()
print (data_1)

# 创建 books 数据库，并获取执行结果
cursor.execute("create database books;")
data_2 = cursor.fetchall()
print (data_2)

# 查询数据库，并获取执行结果
cursor.execute("show databases;")
data_3 = cursor.fetchall()
print (data_3)

# 关闭游标，关闭数据库连接
cursor.close()
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
db.close()
```

执行结果如下：

```
Database version : 5.7.37
```

3.5、创建数据表

数据库连接成功以后，我们就可以为数据库创建数据表了。使用 `execute()` 方法来为数据库创建表 `books` 图书表。

实战 1：创建 `books` 图书表

`books` 表包含 `id`（主键）、`name`（图书名称），`category`（图书分类），`price`（图书价格）和 `publish_time`（出版时间）5 个字段。创建 `books` 表的 SQL 语句如下：

```
CREATE TABLE books (  
    id int(8) NOT NULL AUTO_INCREMENT,  
    name varchar(50) NOT NULL,  
    category varchar(50) NOT NULL,  
    price decimal(10,2) DEFAULT NULL,  
    publish_time date DEFAULT NULL,  
    PRIMARY KEY (id)  
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

在创建数据表前，使用如下语句：

```
DROP TABLE IF EXISTS 'books';
```

如果 `books` 数据库中已经存在 `books`，那么先删除 `books`，然后再创建 `books` 数据表。

实例 1：连接并创建 `books` 数据库

源码如下：

```
import pymysql  
  
# 打开数据库连接  
db = pymysql.connect(host='192.168.13.253', user='root',  
password='zz123..', port=3306)  
# 使用 cursor() 方法创建一个游标对象 cursor  
cursor = db.cursor()  
# 连接到 books 库  
cursor.execute("use books")  
# 使用 execute() 方法执行 SQL，如果表存在则删除  
cursor.execute("DROP TABLE IF EXISTS books")  
# 使用预处理语句创建表
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
sql = """
CREATE TABLE books (
    id int(8) NOT NULL AUTO_INCREMENT,
    name varchar(50) NOT NULL,
    category varchar(50) NOT NULL,
    price decimal(10,2) DEFAULT NULL,
    publish_time date DEFAULT NULL,
    PRIMARY KEY (id)
) ENGINE=MyISAM AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
"""

# 执行 SQL 语句
cursor.execute(sql)
# 关闭数据库连接
db.close()
```

执行结果如下：

```
mysql> describe books.books;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id     | int(8) | NO | PRI | NULL | auto_increment |
| name   | varchar(50) | NO | | NULL | |
| category | varchar(50) | NO | | NULL | |
| price  | decimal(10,2) | YES | | NULL | |
| publish_time | date | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

3.6、操作 MySQL 数据表

MySQL 数据表的操作主要包括数据的增删改查，与操作 SQLite 类似。

实战 1：向 books 图书表添加图书数据

在向 books 图书表中插入图书数据时，可以使用 `execute()` 方法添加一条记录，也可以使用 `executemany()` 方法批量添加多条记录，`executemany()` 方法的语法格式如下：

```
executemany(operation, seq_of_params)
● operation: 操作的 SQL 语句。
● seq_of_params: 参数序列。
```

`executemany()` 方法批量添加多条记录的具体代码如下：

```
源码如下：
import pymysql
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
# 打开数据库连接
db = pymysql.connect(host='192.168.13.253', user='root',
password='zzl23..', port=3306, database='books', charset='utf8')
# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# 数据列表
data = [("Kubernetes 权威指南", 'Linux', '79.80', '2018-5-20'),
        ("Nginx 中文指南", 'Linux', '69.80', '2018-6-18'),
        ("鸟哥的Linux私房菜", 'Linux', '69.80', '2017-5-23'),
        ("Python 运维开发指南", 'Python', '79.80', '2016-5-23'),
        ("Python Web 开发指南", 'Python', '69.80', '2017-5-23'),
        ]

try:
    # 执行 sql 语句，插入多条数据
    cursor.executemany("insert into books(name, category, price,
publish_time) values (%s,%s,%s,%s)", data)
    # 提交数据
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

执行结果如下：

```
mysql> use books;
Database changed
mysql> select * from books;
```

id	name	category	price	publish_time
1	Kubernetes权威指南	Linux	79.80	2018-05-20
2	Nginx中文指南	Linux	69.80	2018-06-18
3	鸟哥的Linux私房菜	Linux	69.80	2017-05-23
4	Python运维开发指南	Python	79.80	2016-05-23
5	Python Web开发指南	Python	69.80	2017-05-23

```
5 rows in set (0.00 sec)
```

第 14 章：进程和线程

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

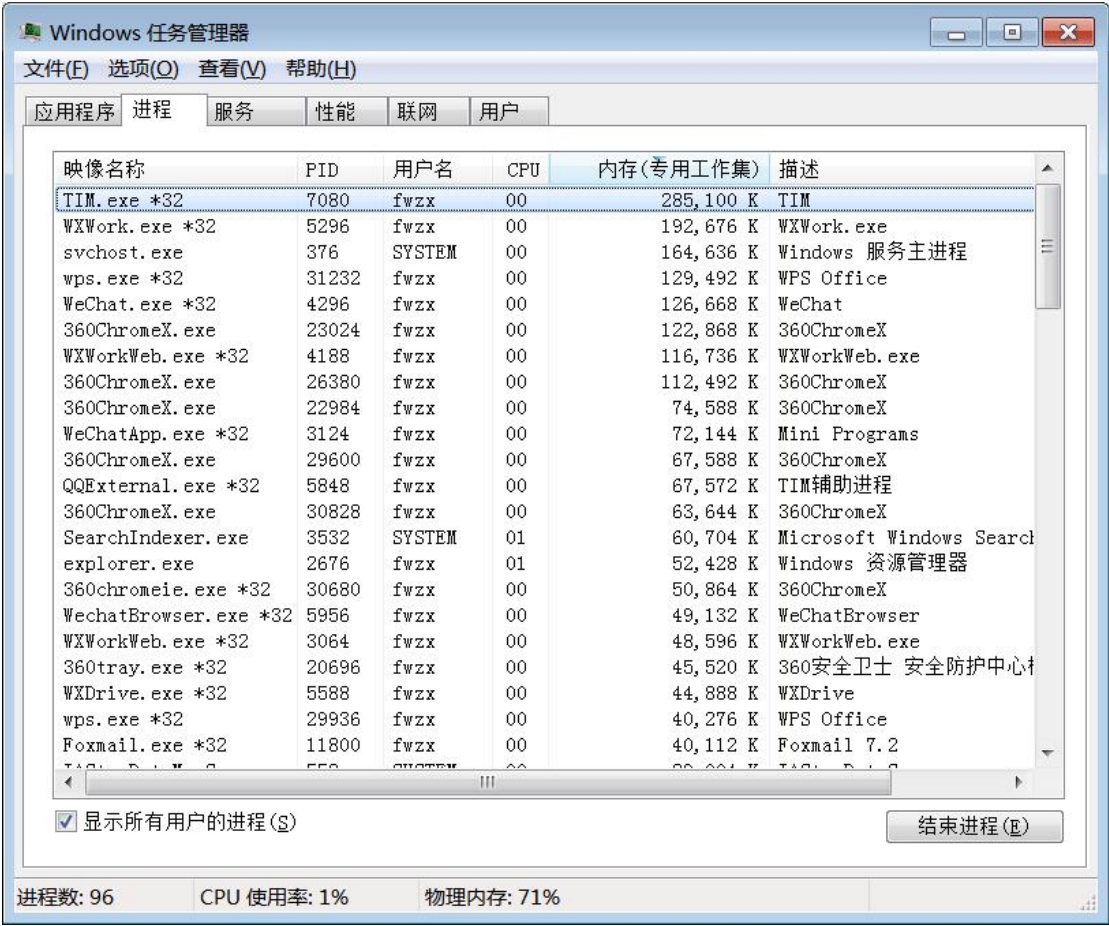
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

为了实现在同一时间运行多个任务,Python 引入了多线程的概念。在 Python 中可以通过方便、快捷的方式启动多线程模式。多线程常被应用在符合并发机制的程序中，例如网络程序等。为了再进一步将工作任务细分，在一个进程内可以使用多个线程。

1、什么是进程

在了解进程之前，我们需要知道多任务的概念。多任务，顾名思义，就是指操作系统能够执行多个任务。例如，使用 windows 或 Linux 操作系统可以同时看电影、聊天、查看网页等，此时，操作系统就是在执行多任务，而每一个任务就是一个进程。

我们可以使用 Windows 的任务管理器，查看一下操作系统正在执行的进程，如下图所示：



进程（process）是计算机中已运行程序的实体。进程与程序不同，程序本身只是指令、数据及其组织形式的描述，进程才是程序（指令和数据）的真正运行实例。

例如：在没有打开 QQ 时，QQ 只是程序。打开 QQ 后，操作系统就为 QQ 开启了一个进程。再打开一个 QQ，则又开启了一个进程。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2、创建进程的常用方式

在 Python 中有多个模块可以创建进程，比较常用的有 `os.fork()` 函数、`multiprocessing` 模块和 `pool` 进程池。

`os.fork()` 函数只适用于 Unix/Linux/Mac 系统上运行，在 Windows 操作系统中不可用，所以来讲解 `multiprocessing` 模块和 `pool` 进程池这两个跨平台模块。

2.1、使用 `multiprocessing` 模块创建进程

`multiprocessing` 模块提供了一个 `Process` 类来代表一个进程对象，语法如下：

```
Process([group [, target [, name [, args [, kwargs]]]])
```

`Process` 类的参数说明如下：

- `group`：参数未使用，值使用为 `None`。
- `target`：表示当前进程启动时执行的可调用对象。
- `name`：为当前进程实例的别名。
- `args`：表示传递给 `target` 函数的参数元组。
- `kwargs`：表示传递给 `target` 函数的参数字典。

实例 1：

实例化 `Process` 类，执行子进程。

代码如下：

```
from multiprocessing import Process
import time

#执行子进程代码
def test(interval):
    time.sleep(interval)
    print('我是子进程')

#执行主程序
def main():
    print('主进程开始')
    p = Process(target=test, args=(1,)) #实例化 Process 进程类
    p.start() #启动子进程
    print('主进程结束')

if __name__ == '__main__':
    main()
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
C:\Users\fwzx>H:\ZYF\python.exe I:\soft\1\test.py
主进程开始
主进程结束
我是子进程
```

注意：由于 IDLE 自身的问题，运行上述代码时，不会输出子进程内容，所以使用命令行方式运行 Python 代码。

提示：

上述代码中，先实例化 Process 类，然后使用 p.start() 方法启动子进程，开始执行 test() 函数。

Process 的实例 p 常用的方法除 start() 外，还有如下常用方法：

- is_alive()：判断进程实例是否还在执行。
- join([timeout])：是否等待进程实例执行结束，或等待多少秒。
- start()：启动进程实例（创建子进程）。
- run()：如果没有给定 target 参数，对这个对象调用 start() 方法时，就将执行对象中的 run() 方法。
- terminate()：不管任务是否完成，立即终止。

Process 类还有如下常用属性

- name：当前进程实例别名，默认未 Process-N，N 为从 1 开始递增的整数。
- pid：当前进程实例的 PID 值。

实例 2：

使用 Process 子类创建 2 个子进程，并记录子进程运行时间

代码如下：

```
from multiprocessing import Process
import time
import os

#两个子进程将会调用的两个方法
def child_1(interval):
    print("子进程(%s)开始执行，父进程为(%s)" % (os.getpid(),
os.getpid()))
    t_start = time.time() #计时开始
    time.sleep(interval) #程序将会被挂起 interval 秒
    t_end = time.time() #计时结束
    print("子进程(%s)执行时间为'%.2f' 秒" % (os.getpid(), t_end -
t_start))
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
def child_2(interval):
    print("子进程(%s)开始执行，父进程为(%s)" % (os.getpid(),
os.getpid()))
    t_start = time.time() #计时开始
    time.sleep(interval) #程序将会被挂起 interval 秒
    t_end = time.time() #计时结束
    print("子进程(%s)执行时间为'%.2f' 秒" % (os.getpid(), t_end -
t_start))

if __name__ == '__main__':
    print("-----父进程开始执行-----")
    print("父进程PID: %s" % os.getpid()) #输出当前程序的PID
    p1 = Process(target=child_1, args=(1,)) #实例化进程 p1
    p2 = Process(target=child_2, name="mrsoft", args=(2,)) #实例化
进程 p2
    p1.start() #启动进程 p1
    p2.start() #启动进程 p2
    #同时父进程仍然往下执行，如果 p2 进程还在执行，将会返回 True
    print("p1.is_alive=%s" % p1.is_alive())
    print("p2.is_alive=%s" % p2.is_alive())
    #输出 p1 和 p2 进程的别名和 PID
    print("p1.name=%s" % p1.name)
    print("p1.pid=%s" % p1.pid)
    print("p2.name=%s" % p2.name)
    print("p2.pid=%s" % p2.pid)
    print("-----等待子进程-----")
    p1.join() #等待 p1 进程结束
    p2.join() #等待 p2 进程结束
    print("-----父进程执行结束-----")
```

运行结果：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
C:\Users\fwzx>H:\ZYF\python.exe I:\soft\1\test.py
-----父进程开始执行-----
父进程PID: 28036
p1.is_alive=True
p2.is_alive=True
p1.name=Process-1
p1.pid=33436
p2.name=mrsoft
p2.pid=32292
-----等待子进程-----
子进程(33436)开始执行，父进程为(33436)
子进程(32292)开始执行，父进程为(32292)
子进程(33436)执行时间为' 1.00' 秒
子进程(32292)执行时间为' 2.00' 秒
-----父进程执行结束-----
```

2.2、使用进程池 Pool 创建进程

在上面，我们学习了使用 Process 类创建 2 个进程。如果要创建几十个或者上百个进程，则需要实例化更多个 Process 类。有没有更好的创建进程的方式解决这类问题呢？答案就是使用 multiprocessing 模块提供的 Pool 类，即 Pool 进程池。

为了更好的理解进程池，可以将进程池比作水池，如下图所示。



我们需要完成放满 10 个水盆的水的任务，而在这个水池中，最多可以安放 3 个水盆接水，也就是同时可以执行 3 个任务，即开启 3 个进程。为更快完成任务，现在打开 3 个水龙头开始放水，当有一个水盆的水接满时，即该进程完成 1 个任务，我们就将这个水盆的水导入水桶中，然后继续接水，即执行下一个任务。如果 3 个水盆每次同时装满水，那么在放满第 9 盆水后，系统会随机分配 1 个水盆接水，另外 2 个水盆空闲。

接下来，先来了解一下 Pool 类的常用方法。常用方法及说明如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

● `apply_async(func[, args[, kwds]])`: 使用非阻塞方式调用 `func()` 函数（并行执行，堵塞方式必须等待上一个进程退出才能执行下一个进程），`args` 为传递给 `func()` 函数的参数列表，`kwds` 为传递给 `func()` 函数的关键字参数列表。

● `apply(func[, args[, kwds]])`: 使用阻塞方式调用 `func()` 函数。

● `close()`: 关闭 Pool，使其不再接受新的任务。

● `terminate()`: 不管任务是否完成，立即终止。

● `join()`: 主进程阻塞，等待子进程的退出，必须在 `close` 或 `terminate` 之后使用。

什么是阻塞？什么是非阻塞

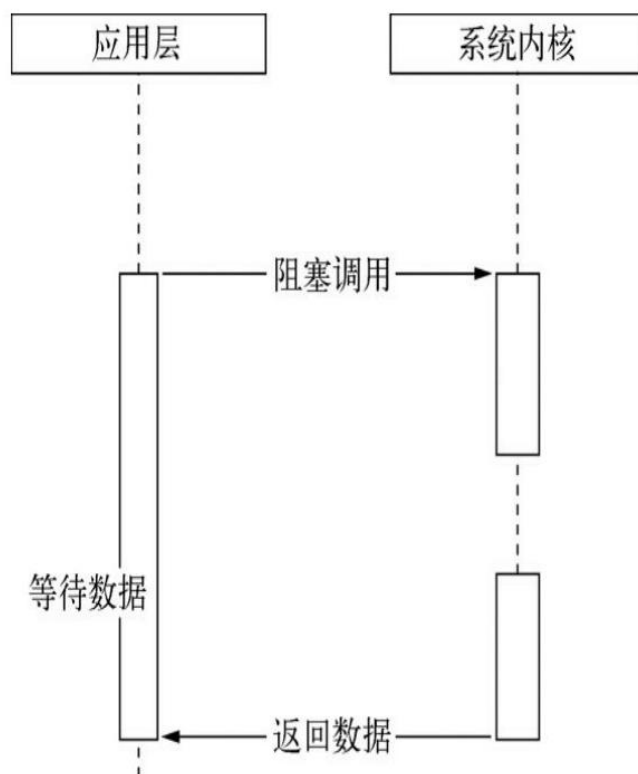
老张爱喝茶，废话不说，煮开水。出场人物：老张，水壶两把（普通水壶，简称水壶；会响的水壶，简称响水壶）。

阻塞：老张把水壶放到火上，立等水开。（同步阻塞）老张觉得自己有点傻

非阻塞：老张把水壶放到火上，去客厅看电视，时不时去厨房看看水开没有。（同步非阻塞）老张还是觉得自己有点傻，于是变高端了，买了把会响笛的那种水壶。水开之后，能大声发出嘀~~~~的噪音。

图解：

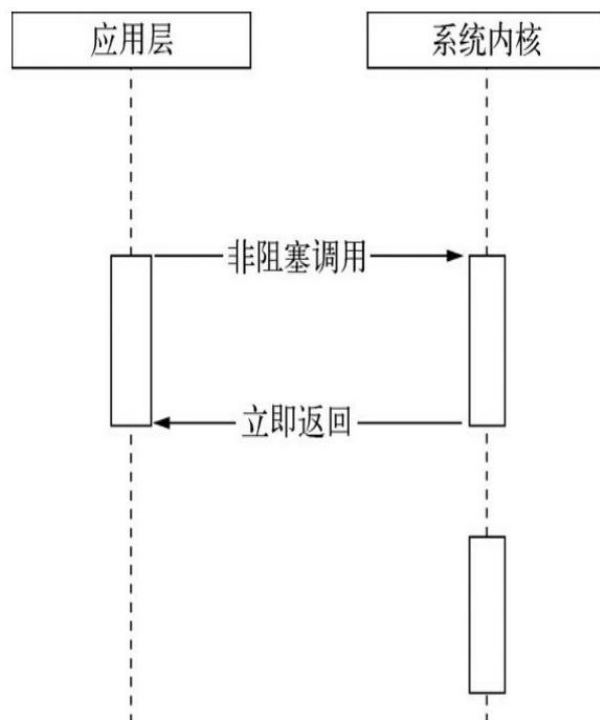
阻塞：



非阻塞：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



实例 1:

模拟水池放水的场景，定义一个进水池，设置最大进程数为 3。然后使用非阻塞方式执行 10 个任务，查看每个进程执行的任务。

代码如下：

```
from multiprocessing import Pool
import time
import os

def task(name):
    print('子进程 (%s) 执行 task %s ...' % (os.getpid(), name))
    time.sleep(1) #休眠 1 秒

if __name__ == "__main__":
    print('父进程 (%s) .' % os.getpid())
    p = Pool(3) #定义一个进程池，最大进程数 3
    for i in range(10): #从 0 开始循环 10 次
        p.apply_async(task, args=(i,)) #使用非阻塞方式调用 task()
```

函数

```
    print('等待所有子进程运行结束...')
    p.close() #关闭进程池，关闭后 p 不再接收新的请求
    p.join() #等待子进程结束
    print('所有子进程运行结束。')
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

运行结果：

```
K:\soft\1>python "test - 副本.py"
父进程 (12828) :
等待所有子进程运行结束...
子进程 (14292) 执行task 0 ...
子进程 (11960) 执行task 1 ...
子进程 (13576) 执行task 2 ...
子进程 (13576) 执行task 4 ...
子进程 (14292) 执行task 3 ...
子进程 (11960) 执行task 5 ...
子进程 (14292) 执行task 6 ...
子进程 (11960) 执行task 7 ...
子进程 (13576) 执行task 8 ...
子进程 (13576) 执行task 9 ...
所有子进程运行结束.
```

从上图可以看出 PID 为 13576 的子进程执行了 4 个任务，而 14292 和 11960 这两个子进程分别执行了 3 个任务。

3、通过队列实现进程间通信

通过上面我们已经学习了如何创建多进程，那么在多进程中，每个进程之间有什么关系呢？其实每个进程都有自己的地址空间、内存、数据栈以及其他记录其运行状态的辅助数据。通过如下实例 1，验证一下进程之间能否直接共享信息。

实例 1：

定义一个全局变量 `g_num`，分别创建 2 个子进程对 `g_num` 执行不同的操作，并输出操作后的结果。

代码如下：

```
from multiprocessing import Process

def plus():
    print("-----子进程 1 开始-----")
    global g_num
    g_num += 50
    print("g_num is %d" % g_num)
    print("-----子进程 1 结束-----")

def minus():
    print("-----子进程 2 开始-----")
    global g_num
    g_num -= 50
    print("g_num is %d" % g_num)
```

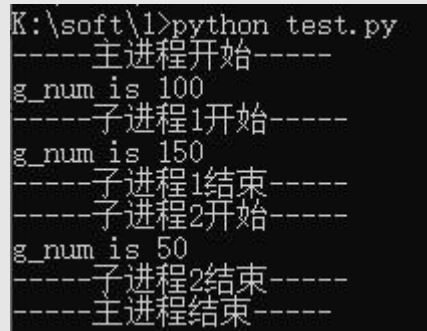
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print("-----子进程 2 结束-----")

g_num = 100
if __name__ == "__main__":
    print("-----主进程开始-----")
    print("g_num is %d" % g_num)
    p1 = Process(target=plus)
    p2 = Process(target=minus)
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    print("-----主进程结束-----")
```

运行结果：

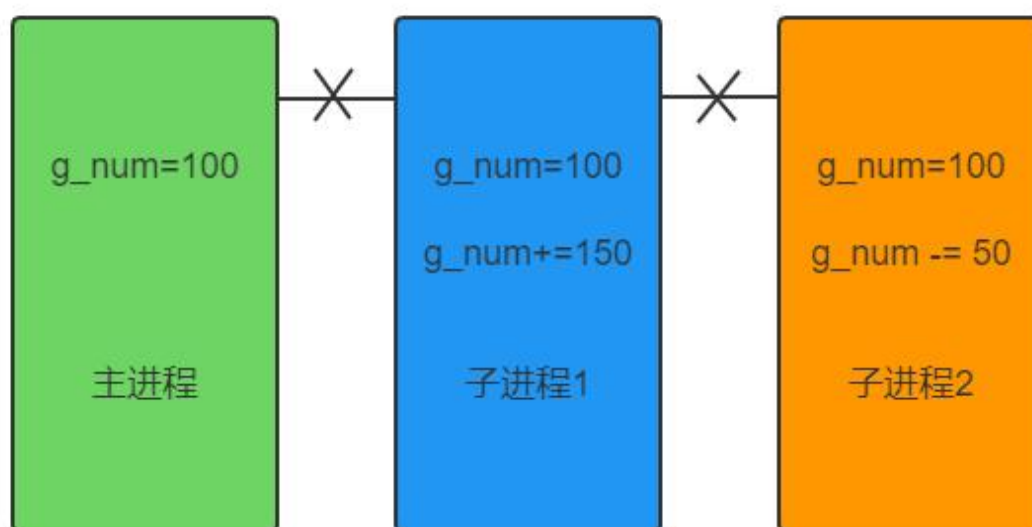


```
K:\soft\1>python test.py
-----主进程开始-----
g_num is 100
-----子进程1开始-----
g_num is 150
-----子进程1结束-----
-----子进程2开始-----
g_num is 50
-----子进程2结束-----
-----主进程结束-----
```

在上述代码中，分别创建了 2 个子进程，一个子进程中令 g_num 加上 50，另一个子进程令 g_num 减去 50。但是从运行结果中可用看出，g_num 在父进程和 2 个子进程中的初始值都是 100。也就是全局变量 g_num 在一个进程中的结果，没有传递到下一个进程中，即进程之间没有共享信息。进程间示意图如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



要如何才能实现进程之间的通信呢？

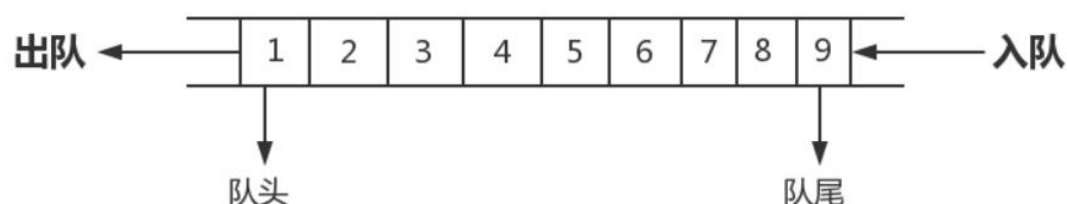
Python 的 multiprocessing 模块包装了底层的机制，提供了 Queue（队列）、Pipes（管道）等多种方式来交换数据。接下来讲解一下 Queue（队列）来实现进程间的通信。

3.1、队列简介

Queue（队列）就是模仿现实中的排队。例如学生在食堂排队买饭。新来的学生排到队伍最后，最前面的学生吃完饭走开，后面的学生跟上。可以看出队伍有两个特点：

- 新来的学生都在队尾。
- 最前面的学生完成后离队，后面一个跟上。

根据以上特点，可以归纳出队列的结构图如下所示：



3.2、多进程队列的使用

进程之间有时需要通信，操作系统提供了很多机制来实现进程间的通信。可以使用 multiprocessing 模块的 Queue 实现多进程之间的数据传递。Queue 本身是一个消息队列程序，下面介绍一下 Queue 的使用。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

初始化 Queue() 对象时（例如：q=Queue(num)），若括号中没有指定最大可以接受的消息数量，或数量为负值，那么就代表可接受的消息数量没有上限（直到内存的尽头）。

Queue 的常用方法如下：

- Queue.qsize()：返回当前队列包含的消息数量。
- Queue.empty()：如果队列为空，返回 True，反之返回 False。
- Queue.full()：如果队列满了，返回 True，反之返回 False。
- Queue.get([block[, timeout]])：获取队列中的一条消息，然后将其从队列中移除，block 默认值为 True。
 - 如果 block 使用默认值，且没有设置 timeout（单位秒），消息队列为空，此时程序将被阻塞（停留在读取状态），直到从消息队列读到消息为止。如果设置了 timeout，则会等待 timeout 秒，若还没读取到任何消息，则抛出“Queue.Empty”异常。
 - 如果 block 值为 False，消息队列为空，则会立即抛出“Queue.Empty”异常。
 - Queue.get_nowait()：相当于 Queue.get(False)。
- Queue.put(item, [block[, timeout]])：将 item 消息写入队列，block 默认值为 True。
 - 如果 block 使用默认值，且没有设置 timeout（单位秒），消息队列如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息队列腾出空间为止，如果设置了 timeout，则会等待 timeout 秒，若还没空间，则抛出“Queue.Full”异常。
 - 如果 block 值为 False，消息队列如果已经没有空间可写入，则会抛出“Queue.Full”异常。
 - Queue.put_nowait(item)：相当于 Queue.put(item, False)。

实例 1：

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

from multiprocessing import Queue

if __name__ == '__main__':
    q=Queue(3) #初始化一个 Queue 对象，最多可以接收三条 Put 消息
    q.put("消息 1")
    q.put("消息 2")
    print(q.full()) #返回 False
    q.put("消息 3")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print(q.full()) #返回 True

#因为消息队列已满，下面的 try 都会抛出异常
#第一个 try 会等待 2 秒后抛出异常，第二个 try 会立刻抛出异常
try:
    q.put("消息 4", True, 2)
except:
    print("消息队列已满，现有消息数量: %s" % q.qsize())

try:
    q.put_nowait("消息 4")
except:
    print("消息队列已满，现有消息数量: %s" % q.qsize())

#读取消息时，先判断消息队列是否为空，再读取
if not q.empty():
    print("-----从队列中获取消息-----")
    for i in range(q.qsize()):
        print(q.get_nowait())
#先判断消息队列是否已满，再写入
if not q.full():
    q.put_nowait("消息 4")
    print("消息 4，写入成功")
else:
    print("消息 4，写入失败")
```

执行结果如下：

```
F:\软件安装位置\Python\vern\Scripts\python.exe C:/Users/岩峰/Desktop/test/test.py
False
True
消息队列已满，现有消息数量: 3
消息队列已满，现有消息数量: 3
-----从队列中获取消息-----
消息1
消息2
消息3
消息4，写入成功
```

3.3、使用队列在进程间通信

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

我们知道使用 multiprocessing.Process 可以创建多进程，使用 multiprocessing.Queue 可以实现队列的操作。接下来，通过一个案例结合 Process 和 Queue 实现进程间的通信。

实例 1:

创建 2 个子进程，一个子进程负责向队列中写入数据，另一个子进程负责从队列中读取数据。为保证能够正确从队列中读取数据，设置读取数据的进程等待时间为 2 秒。如果 2 秒后仍然无法读取数据，则抛出异常。

源码如下:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

from multiprocessing import Process, Queue
import time

#向队列中写入数据
def write_task(q):
    if not q.full():
        for i in range(5):
            message = "消息" + str(i)
            q.put(message)
            print("写入: %s" % message)

#从队列读取数据
def read_task(q):
    time.sleep(1) #休眠 1 秒
    while not q.empty():
        print("读取: %s" % q.get(True, 2)) #等待 2 秒，如果还没有读
        取到任何信息，则抛出 Queue.Empty 异常

if __name__ == "__main__":
    print("-----父进程执行开始-----")
    q = Queue() #父进程创建 Queue，并传给各个子进程
    pw = Process(target=write_task, args=(q,)) #实例化写入队列的
    子进程，并且传递队列
    pr = Process(target=read_task, args=(q,)) #实例化读入队列的子
    进程，并且传递队列
    pw.start() #启动子进程 pw，写入
    pr.start() #启动子进程 pr，读取
    pw.join() #等待 pw 结束
    pr.join() #等待 pr 结束
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
print("-----父进程执行结束-----")
执行结果如下：
F:\软件安装位置\Python\vern\Scripts\python.exe C:/Users/岩峰/Desktop/test/test.py
-----父进程执行开始-----
写入：消息0
写入：消息1
写入：消息2
写入：消息3
写入：消息4
读取：消息0
读取：消息1
读取：消息2
读取：消息3
读取：消息4
-----父进程执行结束-----|
```

4、什么是线程

如果需要同时处理多个任务，一种是可以在一个应用程序内使用多个进程，每个进程负责完成一部分工作；另一种将工作细分为多个任务的方法是使用一个进程内的多个线程。那么，什么是线程呢？

线程（Thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。

例如，对于视频播放器，显示视频用一个线程，播放音频用另一个线程，只有 2 个线程同时工作，我们才能正常观看画面和声音同步的视频。

举一个生活中的例子来更好的理解进程和线程的关系。一个进程就像一座房子，它是一个容器，有着相应的属性，如：占地面积、卧室、厨房和卫生间等。房子本身并没有主动地做任何事情。而线程就是这座房子的居住者，他可以使用房子内每一个房间、做饭、洗澡等。

进程与线程的关系如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



5、创建线程

由于线程是操作系统直接支持的执行单元，因此，高级语言（如 Python、Java 等）通常都内置多线程的支持。Python 的标准库提供了两个模块：`_thread` 和 `threading`，`_thread` 是低级模块，`threading` 是高级模块，对 `_thread` 进行了封装。绝大多数情况下，我们只需要使用 `threading` 这个高级模块。

5.1、使用 `threading` 模块创建线程

`threading` 模块提供了一个 `Thread` 类来代表一个线程对象

`Thread` 类的语法如下：

```
Thread([group [, target [, name [, args [, kwargs]]]])
```

参数说明：

- `group`：值为 `None`，为以后版本而保留。
- `target`：表示一个可调对象，线程启动时，`run()` 方法将调用此对象，默认值为 `None`，表示不调用任何内容。
- `name`：表示当前线程名称，默认创建一个“Thread-N”格式的唯一名称。
- `args`：表示传递给 `target()` 函数的参数元组。
- `kwargs`：表示传递给 `target()` 函数的参数字典。

实例 1：

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
import threading
import time

def process():
    for i in range(3):
        time.sleep(1)
        print("thread name is %s" %
threading.current_thread().name )

if __name__ == "__main__":
    print("-----主进程开始执行-----")
    threads = [threading.Thread(target=process) for i in range(4)]
#创建 4 个线程，存入列表
    for t in threads:
        t.start() # 开启线程
    for t in threads:
        t.join() # 等待子线程结束
    print("-----主进程结束执行-----")
```

执行结果如下：

```
I:\soft\1>H:\ZYF\python test.py
-----主进程开始执行-----
thread name is Thread-1
thread name is Thread-2
thread name is Thread-4
thread name is Thread-3
thread name is Thread-1
thread name is Thread-2
thread name is Thread-4
thread name is Thread-3
thread name is Thread-1
thread name is Thread-2
thread name is Thread-4
thread name is Thread-3
-----主进程结束执行-----
```

上述代码中，创建了 4 个线程，然后分别使用 for 循环执行 start() 和 join() 方法。每个子线程分别执行输出 3 次。从执行结果中可以看出，线程的执行顺序是不确定的。

6、线程间通信

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

我们已经知道进程之间不能直接共享信息，那么线程之间可以共享信息吗？我们来实操验证一下。

实例 1：

定义一个全局变量 `g_num`，分别创建 2 个子线程对 `g_num` 执行不同的操作，并输出操作后的结果。

源码如下：

```
from threading import Thread
import time

def plus(): #第一个线程函数
    print("-----子线程 1 开始执行-----")
    global g_num #定义全局变量
    g_num += 50 #全局变量值加 50
    print("g_num is %d" % g_num)
    print("-----子线程 1 结束执行-----")

def minus(): #第二个线程函数
    time.sleep(1) #休息 1 秒
    print("-----子线程 2 开始执行-----")
    global g_num #定义全局变量
    g_num -= 50 #全局变量值减 50
    print("g_num is %d" % g_num)
    print("-----子线程 2 结束执行-----")

g_num = 100
if __name__ == "__main__":
    print("-----主进程开始执行-----")
    print("g_num is %d" % g_num)
    t1 = Thread(target=plus) #实例化线程 t1
    t2 = Thread(target=minus) #实例化线程 t2
    t1.start() #启动子线程 t1
    t2.start() #启动子线程 t2
    t1.join() #等待子线程 t1
    t2.join() #等待子线程 t2
    print("-----主进程结束执行-----")
```

执行结果如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
I:\soft\1>H:\ZYF\python test.py
-----主进程开始执行-----
g_num is 100
-----子线程1开始执行-----
g_num is 150
-----子线程1结束执行-----
-----子线程2开始执行-----
g_num is 100
-----子线程2结束执行-----
-----主进程结束执行-----
```

上述代码中，定义一个全局变量 `g_num`，赋值为 100，然后创建 2 个线程。一个线程将 `g_num` 增加 50，一个线程将 `g_num` 减少 50。如果 `g_num` 的最终结果为 100，则说明线程之间可以共享数据。

从上面的例子可以得出，在一个进程内的所有线程共享全局变量，能够在不使用其他方式的前提下完成多线程之间的数据共享。

6.1、什么是互斥锁

由于线程可以对全局变量随意修改，这就可能造成多线程之间对全局变量的混乱操作。

依然以房子为例，当房子内只有一个居住者时（单线程），他可以任意时刻使用任意一个房间，如厨房、卧室和卫生间等。但是，当这个房子有多个居住者时（多线程），他就不能在任意时刻使用某个房间，如卫生间，否则就好造成混乱。

如何解决这个问题呢？

一个防止他人进入的简单方法，就是门上加一把锁。先到的人锁上门，后到的人就在门口排队，等锁打开再进去。

这就是“互斥锁”（Mutual exclusion，缩写 Mutex）防止多个线程同时读写某一块内存区域。互斥锁为资源引入一个状态：锁定和非锁定。某个线程要更改共享数据时，先将其锁定，此时资源的状态为“锁定”，其他线程不能更改；直到该线程释放资源，将资源的状态变成“非锁定”时，其他的线程才能再次锁定该资源。互斥锁保证了每次只有一个线程进行写入操作，从而保证了多线程情况下数据的正确性。

6.2、使用互斥锁

在 `threading` 模块中使用 `Lock` 类可以方便地处理锁定。`Lock` 类有 2 个方法：`acquire()` 锁定和 `release()` 释放锁。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

acquire() 锁定和 release() 释放锁示例如下：

```
test = threading.Lock() #创建锁
test.acquire([blocking]) #锁定
test.release() #释放锁
```

语法如下：

- acquire([blocking])：获取锁定，如果有必要，需要阻塞到锁定释放为止。如果提供 blocking 参数并将它设置为 False，当无法获取锁定时将立即返回 False；如果成功获取锁定则返回 True。

- release()：释放一个锁定。当锁定处于未锁定状态时，或者从与原本调用 acquire() 方法的不同线程调用此方法，将出现错误。

实例 1：

通过一个案例来学习互斥锁。这里使用多线程和互斥锁模拟实现多人同时订购电影票的功能，假设电影院某个场次只有 10 张电影票，11 个用户同时抢购该电影票。每售出一张，显示一次剩余的电影票张数。

源码如下：

```
from threading import Thread, Lock
import time

n = 10

def task():
    global n
    mutex.acquire() #上锁
    temp = n #赋值给临时变量
    time.sleep(1) #休眠 1 秒
    n=temp - 1 #数量减 1
    if n>=0: #判断是否还有电影票
        print("购买成功，剩余%d 张电影票" % n)
        mutex.release() #释放锁
    else:
        print("购买失败，剩余电影票不足")
        mutex.release() #释放锁

if __name__ == "__main__":
    mutex = Lock() #实例化 Lock 类
    t_list = [] #初始化一个列表
    for i in range(11):
        t = Thread(target=task) #实例化线程类
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
t_list.append(t) #将线程实例存入列表中
t.start() #创建线程
for t in t_list:
    t.join() #等待子线程结束
```

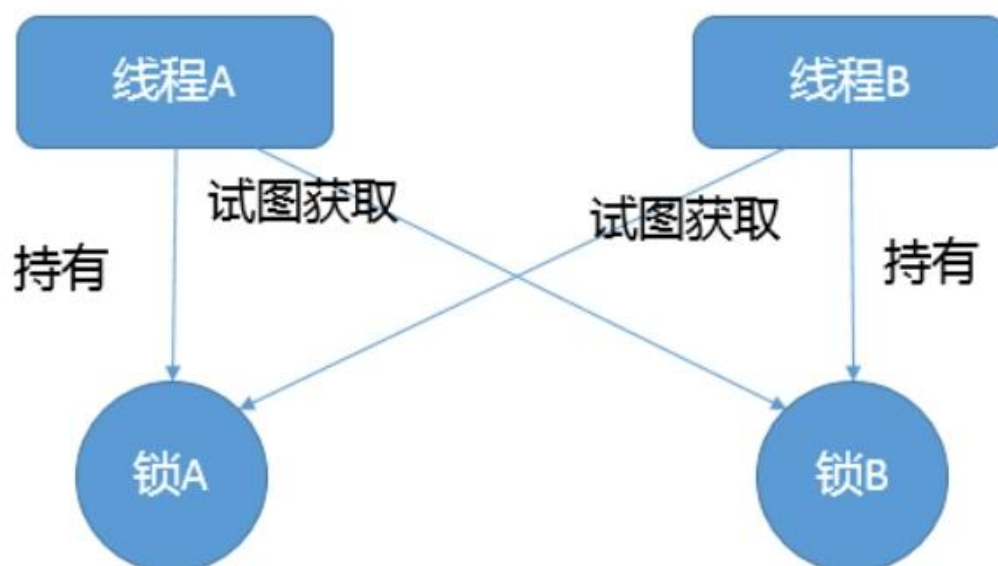
执行结果如下：

```
K:\soft\1>python test.py
购买成功, 剩余9张电影票
购买成功, 剩余8张电影票
购买成功, 剩余7张电影票
购买成功, 剩余6张电影票
购买成功, 剩余5张电影票
购买成功, 剩余4张电影票
购买成功, 剩余3张电影票
购买成功, 剩余2张电影票
购买成功, 剩余1张电影票
购买成功, 剩余0张电影票
购买失败, 剩余电影票不足
```

6.3、多线程死锁及解决方案

在多线程程序中，死锁问题很大一部分是由于线程同时获取多个锁造成的。举个例子：一个线程获取了第一个锁，然后在获取第二个锁的时候发生阻塞，那么这个线程就可能阻塞其他线程的执行，从而导致整个程序假死。

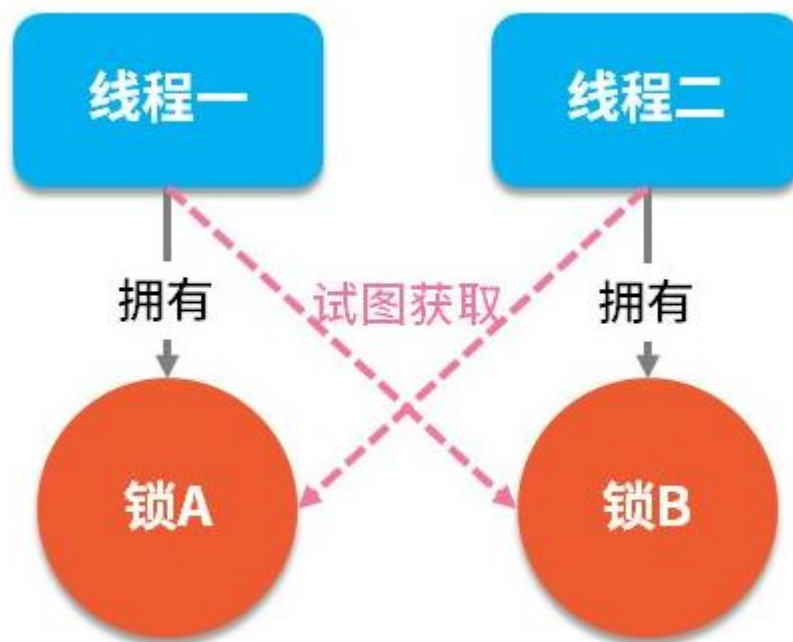
死锁，示例图 1：



死锁，示例图 1：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



所谓死锁：是指两个或两个以上的进程或线程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

在线程间共享多个资源的时候，如果两个线程分别占有一部分资源并且同时等待对方的资源时，就会造成死锁。尽管死锁很少发生，但一旦发生就会造成应用的停止响应。

实例 1：产生死锁

源码如下：

```
from threading import Thread, Lock
import time

def test_1():
    test_A.acquire()
    print("test_1--上锁 test_A")
    time.sleep(1)
    test_B.acquire()
    print("test_1--上锁 test_B")
    test_A.release()
    print("test_1--解锁 test_A")
    test_B.release()
    print("test_1--解锁 test_B")
```

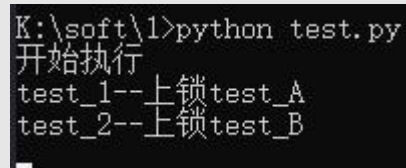
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
def test_2():
    test_B.acquire()
    print("test_2--上锁 test_B")
    time.sleep(1)
    test_A.acquire()
    print("test_2--上锁 test_A")
    test_B.release()
    print("test_2--解锁 test_B")
    test_A.release()
    print("test_2--解锁 test_A")

if __name__ == "__main__":
    print("开始执行")
    test_A = Lock() #实例化 Lock 类
    test_B = Lock() #实例化 Lock 类
    t1 = Thread(target=test_1) #实例化线程类
    t2 = Thread(target=test_2) #实例化线程类
    t1.start() #创建线程
    t2.start() #创建线程
    t1.join() #等待子线程结束
    t2.join() #等待子线程结束
    print("执行结束")
```

执行结果如下：



```
K:\soft\1>python test.py
开始执行
test_1--上锁test_A
test_2--上锁test_B
_
```

从上面的运行结果中可以看出，发生了死锁现象。

实例 2：解决死锁问题

源码如下：

```
from threading import Thread, Lock
import time

def test_1():
    print("test_1--上锁 test_A")
    test_A.acquire()
    time.sleep(1)
    print("test_1--上锁 test_B")
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

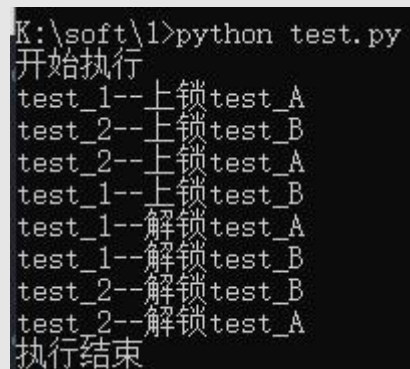
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
test_B.acquire(timeout=3) #在此处添加了超时时间，此处上锁超过
3 秒自动解锁
    print("test_1--解锁 test_A")
    test_A.release()
    print("test_1--解锁 test_B")
    #test_B.release() #因为是死锁，所有在上面代码一定会达到超时时间自动解锁，执行到此处时会报错
```

```
def test_2():
    print("test_2--上锁 test_B")
    test_B.acquire()
    time.sleep(1)
    print("test_2--上锁 test_A")
    test_A.acquire()
    print("test_2--解锁 test_B")
    test_B.release()
    print("test_2--解锁 test_A")
    test_A.release()

if __name__ == "__main__":
    print("开始执行")
    test_A = Lock() #实例化 Lock 类
    test_B = Lock() #实例化 Lock 类
    t1 = Thread(target=test_1) #实例化线程类
    t2 = Thread(target=test_2) #实例化线程类
    t1.start() #创建线程
    t2.start() #创建线程
    t1.join() #等待子线程结束
    t2.join() #等待子线程结束
    print("执行结束")
```

执行结果如下：



```
K:\soft\1>python test.py
开始执行
test_1--上锁test_A
test_2--上锁test_B
test_2--上锁test_A
test_1--上锁test_B
test_1--解锁test_A
test_1--解锁test_B
test_2--解锁test_B
test_2--解锁test_A
执行结束
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

7、关于线程需要注意的两点

7.1、进程和线程的区别

进程和线程的主要区别有：

- 进程是系统进行资源分配和调度的一个独立单位，线程是进程的一个实体，是 CPU 调度和分派的基本单位。

- 进程之间是相互独立的，多进程中，同一个变量，各自有一份备份存在于每个进程中，但互不影响；而同一个进程的多个线程是内存共享的，所有变量都由所有线程共享。

- 由于进程间是独立的，因此一个进程的崩溃不会影响到其他进程；而线程是包含在进程之内的，线程的崩溃就会引发进程的崩溃，继而导致同一进程内的其他线程也崩溃。

7.2、多线程非全局变量是否要加锁

在多线程开发中，全局变量是多个线程都共享的数据，为防止数据混乱，通常使用互斥锁。而局部变量等是各自线程的，是非共享的，所以不需要使用互斥锁。

第 15 章：异常处理及程序调试

本章着重讲解程序异常处理语句和程序调试方法。让同学们拥有排查错误能力。

1、异常概述

在程序运行过程中，经常会遇到各种各样的错误，这些错误统称为“异常”。这些异常有的是由于开发者一时疏忽将关键字敲错导致的，这类错误多数产生的是“SyntaxError: invalid syntax”（无效的语法），这将直接导致程序不能运行。这类异常是显示的，在开发阶段很容易发现。还有一类是隐式的，通常和使用者的操作有关。

实例 1：

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def division():
    num1 = int(input("请输入被除数："))
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
num2 = int(input("请输入除数："))
result = num1//num2
print(result)

if __name__ == "__main__":
    division()

执行结果如下：
请输入被除数：10
请输入除数：0
Traceback (most recent call last):
  File "I:\soft\1\server.py", line 11, in <module>
    division()
  File "I:\soft\1\server.py", line 7, in division
    result = num1//num2
ZeroDivisionError: integer division or modulo by zero
```

运行程序，如果在输入除数时，输入为 0，就会产生 ZeroDivisionError（除数为 0 错误），根源在于算术运算符“10/0”中，0 作为除数出现，所以正在执行的程序被中断，后面 print 的代码都不会被执行了。

Python 中常见的异常及描述

异常	描述
NameError	尝试访问一个没有声明的变量引发的错误
IndexError	索引超出序列范围引发的错误
IndentationError	缩进错误
ValueError	传入的值错误
KeyError	请求一个不存在的字典关键字引发的错误
IOError	输入输出错误（如要读取的文件不存在）
ImportError	当 import 语句无法找到模块或 from 无法在模块中找到相应的名称时引发的错误
AttributeError	尝试访问未知的对象属性引发的错误
TypeError	类型不合适引发的错误
MemoryError	内存不足
ZeroDivisionError	除数为 0 引发的错误

2、异常处理语句

在程序开发时，有些错误并不是每次运行都会出现。就比如上小节所讲的案例。只要输入的数据符合程序的要求，程序就可以正常运行。但如果输入的不符合要求，就会抛出异常并停止运行。这是就需要在开发程序时对可能出现异常的情况进行处理。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

2.1、try...except 语句

在 Python 中，提供了 try...except 语句捕获并处理异常。在使用时，把可能产生异常的代码放在 try 语句块中，把处理结果放在 except 语句块中，这样，当 try 语句块中的代码出现错误，就会执行 except 语句块中的代码，如果 try 语句块中的代码没有错误，那么 except 语句块将不会执行。

语法如下：

```
try:
    block1
except [ExceptionName [as alias]]:
    block2
```

参数说明：

- block1: 表示可能出现错误的代码块
- ExceptionName [as alias]: 可选参数，用于指定要捕获的异常。其中，ExceptionName 表示要捕获的异常名称，如果在其右侧加上“as alias”，则表示为当前异常指定一个别名，通过该别名，可以记录异常的具体内容。except 后面不知道异常名称，则表示捕获全部异常。
- block2: 表示进行异常处理的代码块。在这里可以输出固定的提示信息，也可以通过别名输出异常的具体内容。

实例 1:

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def division():
    num1 = int(input("请输入被除数："))
    num2 = int(input("请输入除数："))
    result = num1//num2
    print(result)

if __name__ == "__main__":
    try: #捕获异常
        division() #调用除法函数
    except: #处理异常
        print("输入错误：除数不能为 0") #输出错误原因
```

执行结果如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
请输入被除数：10
请输入除数：0
输入错误：除数不能为0
```

2.2、try...except...else 语句

try...except...else 语句就是在 try...except 语句的基础上再添加了一个 else 子句，用于指定当 try 语句块中没有发现异常时要执行的语句块。该语句块中的内容在 try 语句中发现异常时，将不被执行。

语法如下：

```
try:
    block1
except [ExceptionName [as alias]]:
    block2
else:
    block3
```

实例 1：

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def division():
    num1 = int(input("请输入被除数："))
    num2 = int(input("请输入除数："))
    result = num1//num2
    print(result)

if __name__ == "__main__":
    try: #捕获异常
        division() #调用除法函数
    except ZeroDivisionError: #处理异常
        print("输入错误：除数不能为0") #输出错误原因
    except ValueError as i:
        print("输入错误：",i)
    else:
        print("程序执行完成！")
```

执行结果如下：

```
请输入被除数：10
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
请输入除数：2
5
程序执行完成！
```

2.3、try...except...finally 语句

完整的异常处理语句应该包含 finally 代码块，通常情况下，无论程序中是否有异常产生，finally 代码块中的代码都会被执行。

语法格式如下：

```
try:
    block1
except [ExceptionName [as alias]]:
    block2
finally:
    block3
```

对于 try...except...finally 语句的理解并不复杂，它只是比 try...except 语句多了一个 finally 代码块，如果程序中有一些在任何情形中都必须执行的代码，那么就可以将它们放在 finally 语句的区块中。

使用 except 子句是为了允许处理异常。无论是否引发了异常，使用 finally 子句都可以执行。如果分配了有限的资源（如打开文件），则应将释放这些资源的代码放在 finally 块中。

实例 1：

源码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

def division():
    num1 = int(input("请输入被除数："))
    num2 = int(input("请输入除数："))
    result = num1//num2
    print(result)

if __name__ == "__main__":
    try: #捕获异常
        division() #调用除法函数
    except ZeroDivisionError: #处理异常
        print("输入错误：除数不能为 0") #输出错误原因
    except ValueError as i:
        print("输入错误：",i)
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
else:
    print("程序执行完成！")
finally:
    print("释放资源，并关闭！")
```

执行结果如下：

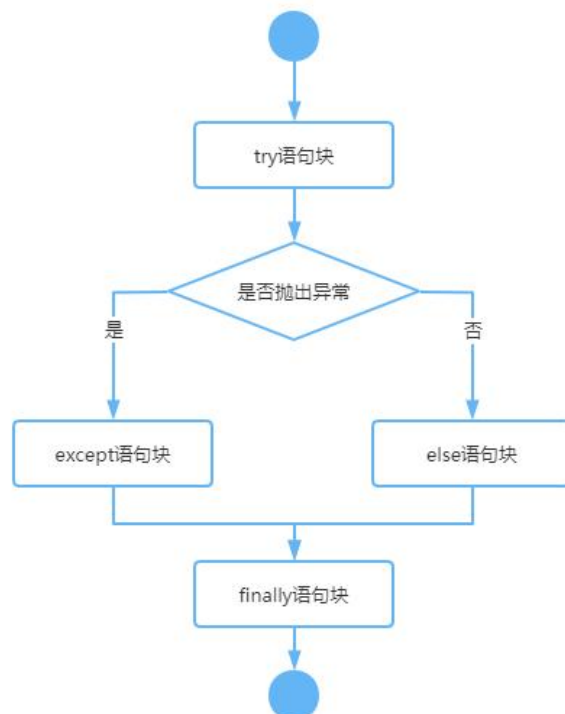
请输入被除数：10

请输入除数：0

输入错误：除数不能为0

释放资源，并关闭！

各个子句的执行关系如下图所示：



附录一：网址

Python 中文手册官网地址：<https://docs.python.org/zh-cn/3/>

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**