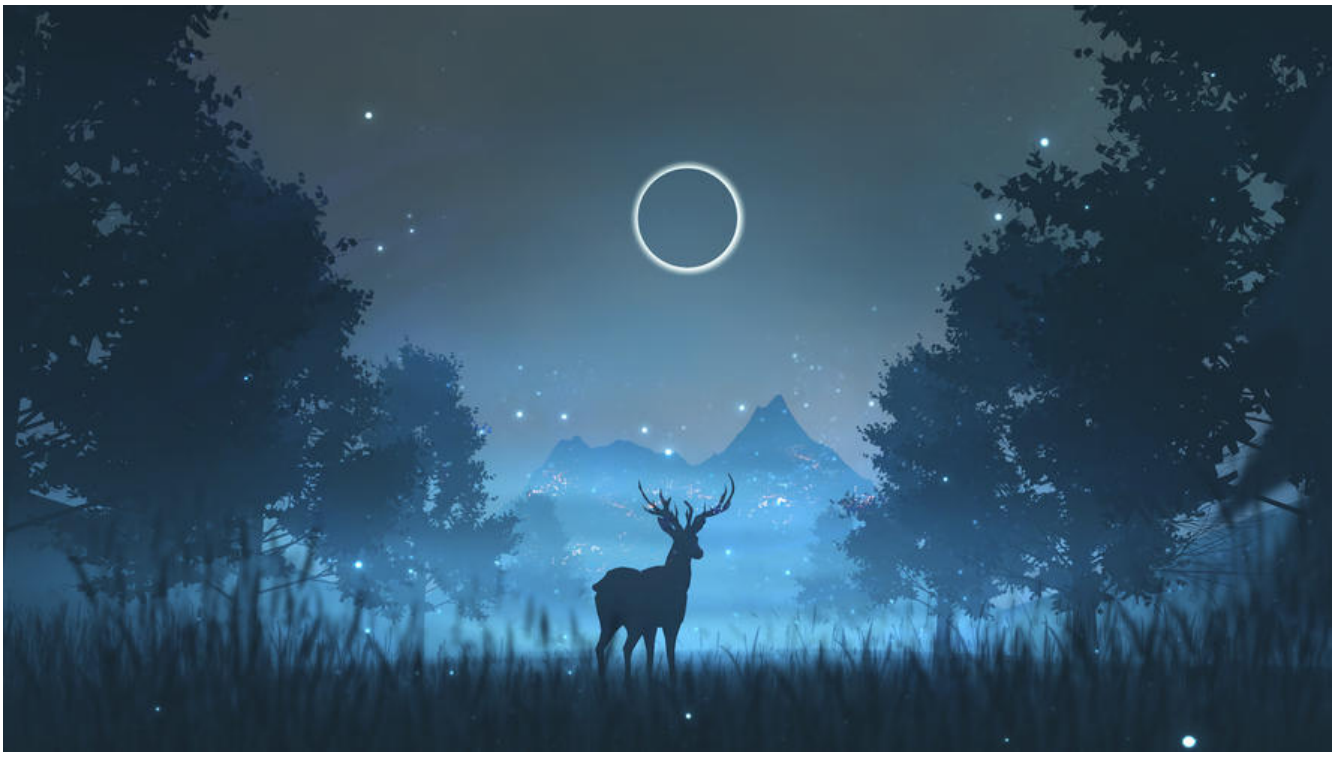
链滴

小白的 Python 修炼手册：入门篇

作者：jingqueyimu

原文链接：https://ld246.com/article/1628256554210

来源网站：链滴



Life is short, you need Python。（人生苦短，我用 Python。）<p align="right">——Bruce Eckel /p>

前言

听说现在是全民 Python 的时代，虽然不知道事实如何，但学会 Python 的确可以做很多事。据我了，Python 目前主要有五大用途：网站开发、网络爬虫、人工智能、数据分析、自动化运维。而对于场人士来说，Python 则可以用来进行自动化办公。除此之外，如果你想自己开发一些小游戏、小工，Python 也是一个不错的选择。

相较于其他编程语言，Python 不仅上手容易，而且由于代码库（就是别人已经写好的代码，我们可直接拿来用）丰富，我们在实现功能时需要写的代码也更少。

目前，Python 有 2.x 和 3.x 两个版本，且两个版本不兼容。本文主要针对 3.x 版本。

正文

从小白的日常说起

在学习 Python 编程之前，我们先来看一个现实生活中可能出现的场景——

某天，小白下班回到家里，想起今天的《斗罗大陆》更新了。于是赶紧兴冲冲地跑到电脑桌前，打开记本电脑，然后登录腾讯视频观看最新一集的《斗罗大陆》。但是，看视频怎么能少得了瓜子呢？于，小白又从柜子里拿出了一包瓜子，磕着瓜子看着《斗罗大陆》，小白露出了满意的笑容。

看到这里，你可能会一脸懵逼——这傻逼作者在讲什么，这跟 Python 编程有什么关系吗？但我要说是，上述场景其实就是一个程序的执行过程，它和计算机内的程序执行类似，只不过这个程序的执行是发生在我们所处的现实世界当中。至于为什么这么说，我们下面就来看看——

一、现实世界 VS 计算机世界

在计算机的世界里，一个程序的执行过程可以简单地理解为：计算机从内存中的指定地址获取到所需据，然后对该数据进行某种操作，如此循环往复。

现在我们把小白回家日常的场景描述稍微改造一下：小白从家里的电脑桌上获取到笔记本电脑，然后开笔记本电脑观看《斗罗大陆》。再从柜子里获取到瓜子，开始边嗑瓜子边看《斗罗大陆》。

怎么样？是不是跟计算机内程序的执行过程很相似。其实计算机从内存中的指定地址获取数据，就相于小白从家里的某个位置获取某个物品。而计算机对数据的操作，就相当于小白对物品的操作。

计算机世界	现实世界
计算机	小白
计算机内存	小白的家
内存地址	家里的位置
数据	物品
从内存中指定地址获取数据	从家里的某个位置获物品
对数据的操作	对物品的操作

二、小白执行手册 VS Python 程序代码

现在，假设小白是一个听话的小白，无论你让他做什么，他都会按你的要求一丝不苟的照做。于是你嘿一笑，立马给小白安排了 10 套暑假作业、20 套网络课程、30 套健身计划。小白很开心，并对你示万分感谢。

但是，你不可能一直呆在小白身边，给他讲解每个步骤。于是，你想到可以写一份执行手册，然后让白按照上面的步骤去执行。这份执行手册其实就相当于 Python 程序代码。只不过你的执行手册是写小白看的，而 Python 程序代码是写给计算机看的。换句话说，Python 程序代码就是写给计算机的行手册。

三、执行手册书写格式 VS Python 编程语法

假设你是一个强迫症患者，为了写出一份美观、简洁的执行手册，你规定了一些书写格式——

1、给物品取别名

可以给某个位置的物品取一个独一无二的别名。当需要用到该物品时，可以使用别名代替。其书写格为：

```
别名 = 某个位置的物品
```

2、给一系列步骤取行为名

在给小白写某个行为的具体步骤时，可以给这一系列步骤取个名称，该名称我们称为行为名。当需要到这些步骤时，可以使用行为名代替。其书写格式为：

```
行为名():
    步骤1
    步骤2
    ...
    步骤n
```

注意：行为名后面带了括号和冒号。

3、使用行为名代替一系列步骤

由于一些步骤可能被多次用到，如果每次都要写一大堆步骤就太麻烦了。因此，我们可以在给这些步取完行为名后，用行为名来代替这些步骤。其书写格式为：

```
步骤1
步骤2
行为名()
...
步骤n
```

注意：使用行为名代替一系列步骤时，要在行为名后面加括号，以表明这是一个行为名。

4、条件判断的写法

当需要根据具体情况（即满足某种条件）来决定是否进行某种操作时，使用以下书写格式：

```
如果 某种条件:
    某种操作
```

注意：这里的"如果"，是固定写法。

5、重复操作的写法

对于当满足某种条件时，需要一直重复进行某种操作的情况，使用以下书写格式：

```
当 某种条件:
    某种操作
```

注意：这里的"当"，是固定写法。

规定好了执行手册的书写格式后，只要让小白学会怎么看懂这些书写格式，那他就能根据你的执行手自己去执行了。

这些书写格式就相当于 Python 编程语言的语法。我们在编写 Python 程序时，只要严格按照 Python 语法去编写程序代码，计算机就能理解并按你的要求去执行了。

四、编写小白执行手册 VS 编写 Python 程序代码

现在，你怀着激动而又忐忑的心情开始给小白编写执行手册了。最终你的《小白执行手册》是这样的—

```
当 暑假作业没写完10套:
    写暑假作业()

当 网络课程没学完20套:
    观看网络课程()

当 健身不足30套:
    健身()
    喝水()

写暑假作业():
```

```
    暑假作业 = 书桌抽屉里的白色封面暑假作业
    铅笔 = 书桌上笔筒里的黑色铅笔
    打开暑假作业
    如果 铅笔尖钝了:
        削铅笔
    用铅笔写暑假作业

观看网络课程():
    笔记本电脑 = 电脑桌上的ThinkPad笔记本电脑
    打开笔记本电脑
    登录网易云课堂
    观看Python课程

健身():
    哑铃 = 南边墙角下的15公斤哑铃
    哑铃弯举100下
    哑铃划船100下
    哑铃卧推100下

喝水():
    矿泉水 = 冰箱里的娃哈哈矿泉水
    打开矿泉水瓶盖
    喝矿泉水
```

看到如此简单、优雅的执行手册，是不是很令人身心舒畅？我们接下来要学习的 Python 编程语言就一门简单、优雅的编程语言。

而且，编写 Python 程序代码的过程，与编写《小白执行手册》的过程也很类似。只不过，编写 Python 程序代码时，涉及到语法会更加丰富，对一些格式的要求也更加严格。

现在开始学 Python

我们已经知道了编写 Python 程序的过程，其实就是给计算机编写执行手册的过程。也知道了一个程的执行过程，就是不断地根据地址从内存中取数据，然后操作数据的过程。而 Python 程序在执行过中所涉及的东西，有些是计算机本身已有的，有些则是需要我们通过 Python 语言告诉计算机的。

1、程序执行涉及的东西

接下来，我们就来简单地了解一下 Python 程序执行过程中所涉及到的一些东西。同时明确一下哪些计算机本身已有的，哪些我们需要告诉计算机的——

（1）内存

内存是计算机内用于存放程序运行时所需数据的地方，我们在编写 Python 程序时涉及到的数据，在用前都会存放在内存中。

内存由计算机自己管理，我们无法直接控制（可以通过一些优化技巧来间接控制，但我们目前先不考这个），因此我们在编写 Python 程序时不用管这个。

其实计算机会将内存分配给 Python 虚拟机，然后由 Python 虚拟机负责管理内存，但我们可以把 Python 虚拟机简单地理解为计算机的一部分。

（2）数据

数据可以简单地理解为存储在计算机内的东西。

计算机存储数据的地方有磁盘和内存：磁盘就是我们平常在 Windows 电脑上看到的 C 盘、D 盘等，永久保存数据；内存则负责临时保存程序运行时所需要的数据，它无法永久保存数据。我们后面提到据时，如果没有特殊说明就是指内存中的数据。

程序运行时，计算机会给这个程序分配一个内存块，这块内存一开始是没有任何数据的。这就跟小白天搬了新家一样，一开始家里是没有任何东西的。因此我们在编写 Python 程序时，就需要告诉计算如何在内存中生成数据，一般有以下三种生成数据的方式——

● 自己创建：后面我们会学到如何创建基本类型的数据以及自定义类型的数据。基本类型是 Python 身提供的数据类型，而自定义类型则通过类和对象来创建（类和对象是面向对象编程中的概念，这个们后面会学到）。

● 从计算机本地磁盘读取：后面我们要学的文件读写，就是用来处理来自计算机本地磁盘的数据的。

● 从网络中读取：由于本文只是带大家入门的，因此我们不讲如何处理来自网络中的数据，有兴趣的以自己百度下“Python 网络编程”。

无论是自己创建的、从磁盘读取的、还是从网络中读取的数据，最终都会存放在内存中。

（3）数据在内存中的地址

Python 程序在执行过程中，所涉及到的数据都会在内存中有一个对应的地址。一旦需要这个数据，能根据这个地址从内存中获取该数据。

也就是说，我们在编写 Python 程序时，如果需要某个数据的话，只需要告诉计算机这个数据所在的址就行了。但由于我们不像计算机一样，可以根据地址就能知道里面存的是什么数据，因此，为了方编写 Python 程序，我们可以给这个地址取个别名，这个别名在编程语言中称为变量名，而取别名则为赋值。

● 变量：本质上是一小块内存空间。在 Python 中，变量里存储的是地址，而变量具有变量名，这个量名就是地址的别名。由于计算机会自动将地址转换成数据（从地址中获取数据），因此，我们可以单地理解为变量里存储的是数据，而变量名就是数据的别名。

● 赋值：将某一数值赋给某个变量的过程。在 Python 中，就是将地址值赋给某个变量的过程，这个程相当于给地址取别名。由于计算机会自动将地址转换成数据（从地址中获取数据），因此，我们可把赋值简单理解为给数据取别名。

（4）如何根据地址从内存中获取数据

计算机会自动根据变量名找到对应的地址，再从该地址获取数据。我们在编写 Python 程序时不用管个。

（5）对数据进行何种操作

由于对数据进行什么操作，是由我们的意愿决定的。因此，我们在编写 Python 程序时，就必须告诉算机如何操作数据。

一般操作数据有两种方式——

● 运算符：也叫操作符，其实就跟数学里的加、减、乘、除等运算符一样。

● 函数：函数中封装了一系列代码片段，该代码片段一般用于实现特定功能。跟我们前面说的给一系步骤取一个行为名类似，函数就相当于给一系列代码片段取一个名称（函数名）。

（6）当存在多个操作时，如何控制操作的流程

当存在多个操作时，我们必须告诉计算机哪个操作先执行，哪个后执行。否则，计算机将无法正常执程序。

在编程语言中，一般有三种控制流程的方式——

- 顺序控制：从上往下依次执行代码。程序执行时，默认就是顺序执行。
- 选择控制：根据不同条件，执行不同代码。后面我们要学的条件语句，就是用来进行选择控制的。
- 循环控制：根据指定条件，重复执行某段代码。后面我们要学的循环语句，就是用来进行循环控制。

2、编写 Python 程序需要做的事

现在，我们来总结一下，在编写 Python 程序时，一般需要做哪些事情——

1. 生成数据：创建基本类型数据、创建自定义数据（类、对象）、从磁盘中读取、从网络中读取。
2. 给数据取别名：变量、赋值。
3. 操作数据：运算符（操作符）、函数。
4. 控制操作数据的流程：顺序、选择、循环。

一、Python 安装与运行

1、Python 安装

由于本人只有 Windows 电脑，因此这里只介绍 Windows 上安装 Python 的过程。

（1）下载 Python 安装包

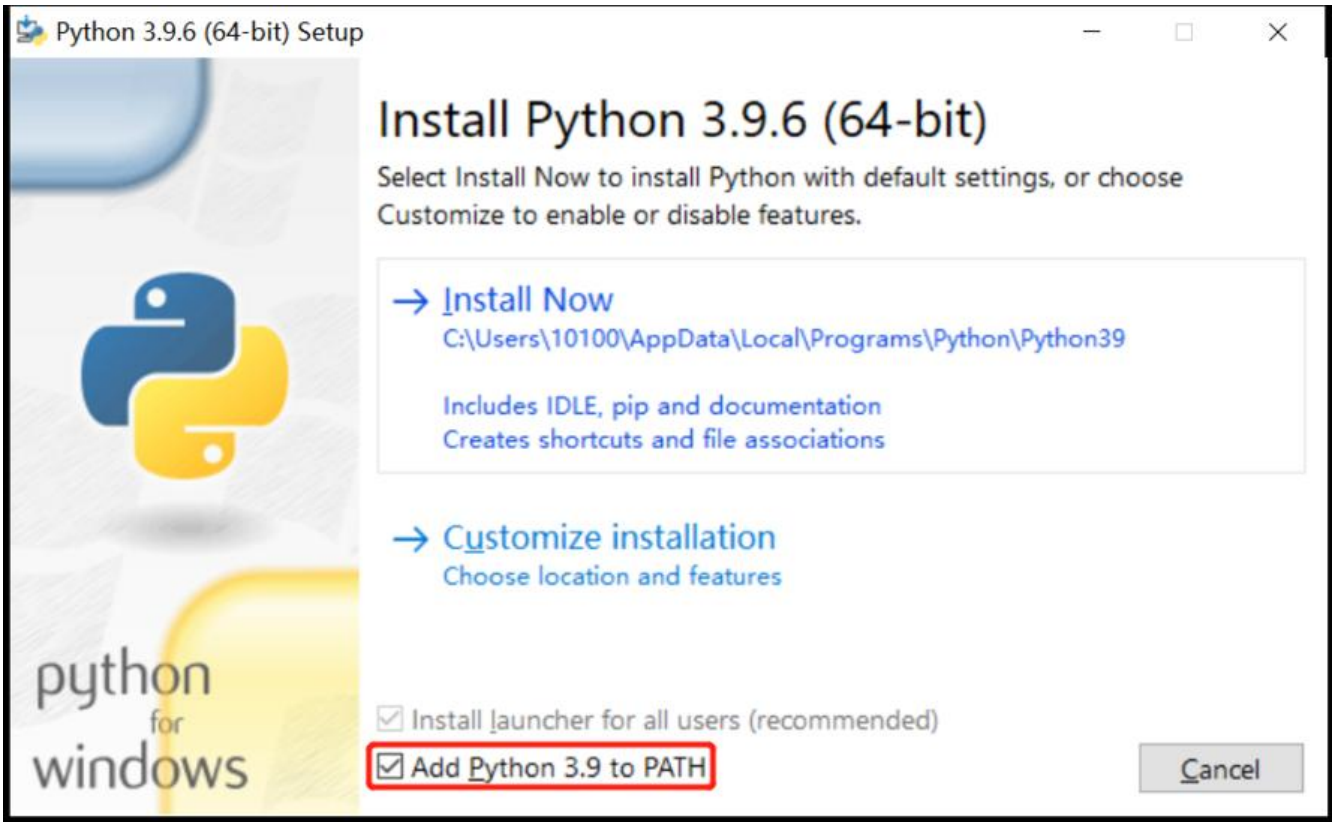
我们这边下载 Python 3.9.6 版本——

- 32 位 Windows 系统：https://www.python.org/ftp/python/3.9.6/python-3.9.6.exe
- 64 位 Windows 系统：https://www.python.org/ftp/python/3.9.6/python-3.9.6-amd64.exe

如果要下载其他平台或者其他版本的 Python 安装包，可以自己到官网去下载。官网下载地址：https://www.python.org/downloads/

（2）安装 Python

下载完安装包后，双击运行，勾选上“Add Python 3.9 to PATH”，然后点击“Install Now”即可。



如果不想使用默认安装，也可以选择“Customize installation”进行自定义安装。

（3）检查是否安装成功

安装完成后，打开命令提示符（按下 Windows 键 + R 键，在弹出的输入框内输入“cmd”，敲回，即可打开命令提示符），输入 python 后，敲回车，如果出现类似以下内容，则说明安装成功。

```
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\10100>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win
2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

否则，有可能是没有勾选上“Add Python 3.9 to PATH”，这时候就需要修改环境变量，把 python.xe 所在路径添加在 Path 中。如果不知道怎么修改环境变量，可以直接卸载重装。

2、Python 运行

下面介绍三种运行 Python 的方式——

（1）命令行模式运行 Python

按下 Windows 键 + R 键，在弹出的输入框内输入“cmd”，敲回车，即可进入命令行模式。

在命令行模式下，可进入 Python 文件（.py 文件）所在目录，输入 python 文件名.py 后敲回车，运行 Python 文件。

```
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\10100>cd /d D:\mypython

D:\mypython>python hello.py
hello world
```

也可以直接输入 python 文件完整路径，而不用进入 Python 文件所在目录。

```
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\10100>python D:\mypython\hello.py
hello world
```

（2）交互模式运行 Python

进入命令行模式后，输入 python，敲回车，即可进入交互模式。输入 exit()，敲回车，即可退出交模式。

在交互模式下，会有 >>> 提示符，>>> 后可直接编写 Python 代码。编写完后敲回车，即可执行代。

```
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\10100>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win
2
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello world')
hello world
>>>
```

（3）集成开发环境（IDE）运行 Python

集成开发环境（IDE）可理解为用于编写 Python 程序的一整套工具，只不过这些工具被集中放在一软件内。比如 PyCharm 或者装了相关插件的 Visual Studio Code（简称 VS Code）。

我们在实际编写 Python 程序时，一般会使用 IDE。由于 IDE 自带了运行 Python 的功能，因此前面种运行 Python 的方式，我们了解即可。

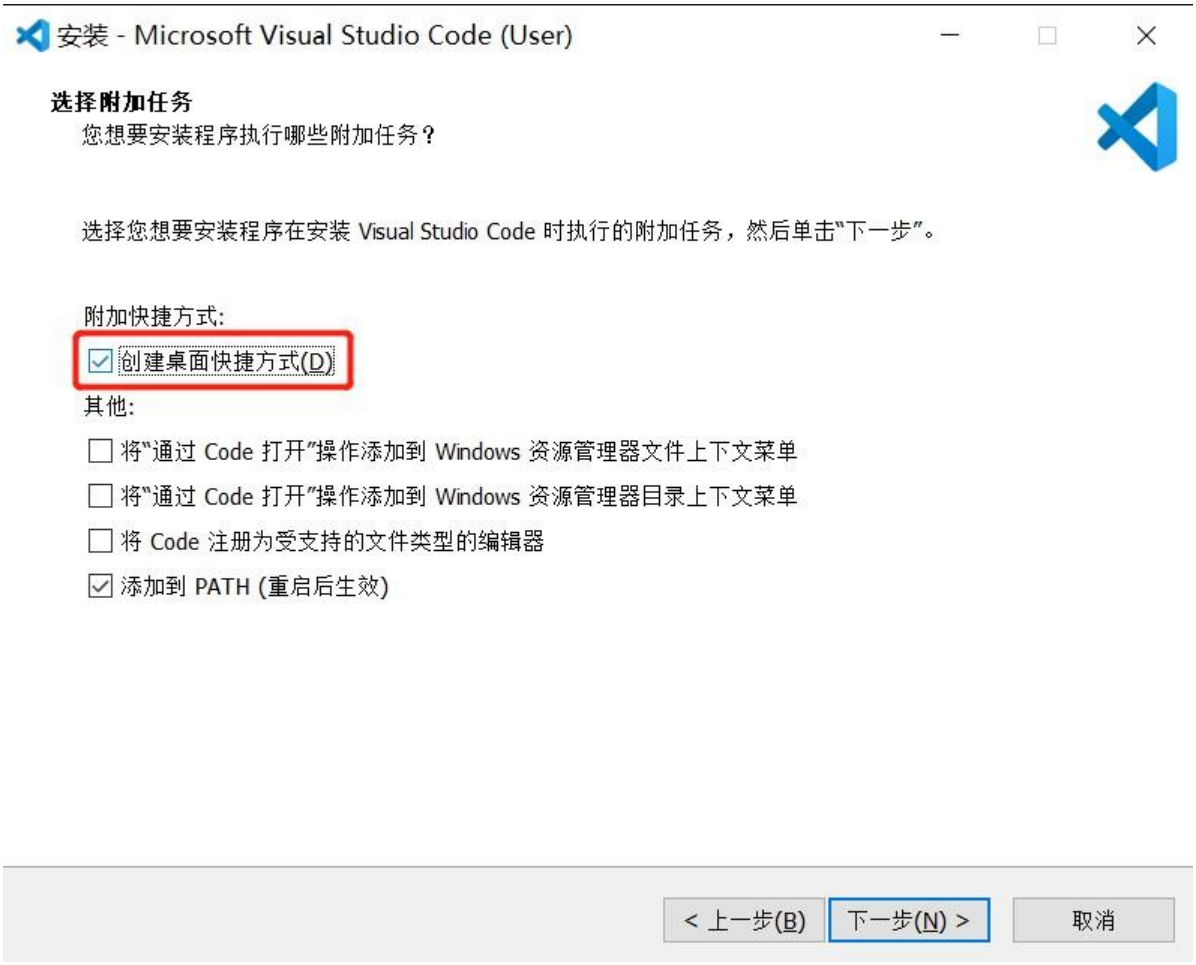
二、第一个 Python 程序

这里推荐使用 VS Code 来编写 Python 程序，推荐理由很简单——它的界面清爽、干净，看起来贼服（一个不太正经的理由，勿喷，谢谢）！除此之外，VS Code 也可以通过安装插件来集成更多的功。

1、安装 VS Code

下载地址：https://code.visualstudio.com/

安装过程中，注意勾选上“创建桌面快捷方式”的选项，其他按默认的来就行。

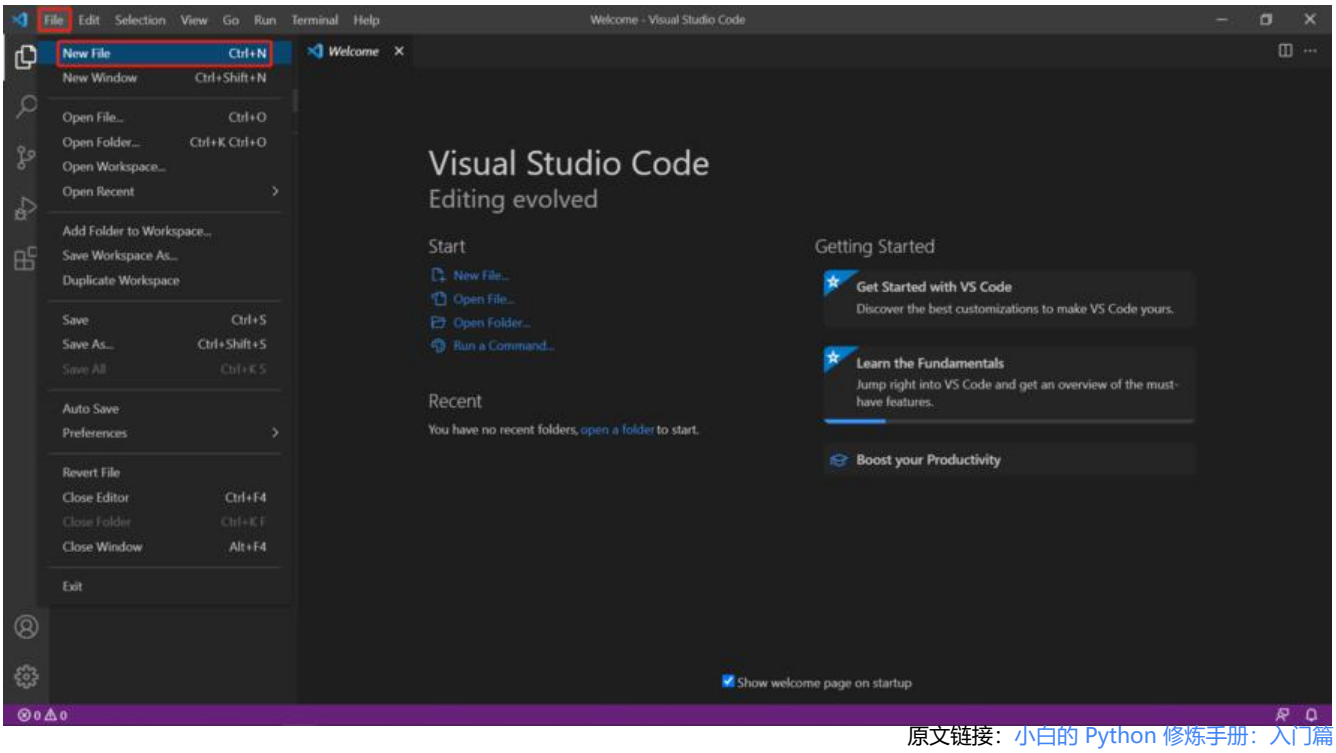


2、使用 VS Code 编写程序

现在，我们来使用 VS Code 编写一个打印“hello world”的 Python 程序——

（1）创建新文件

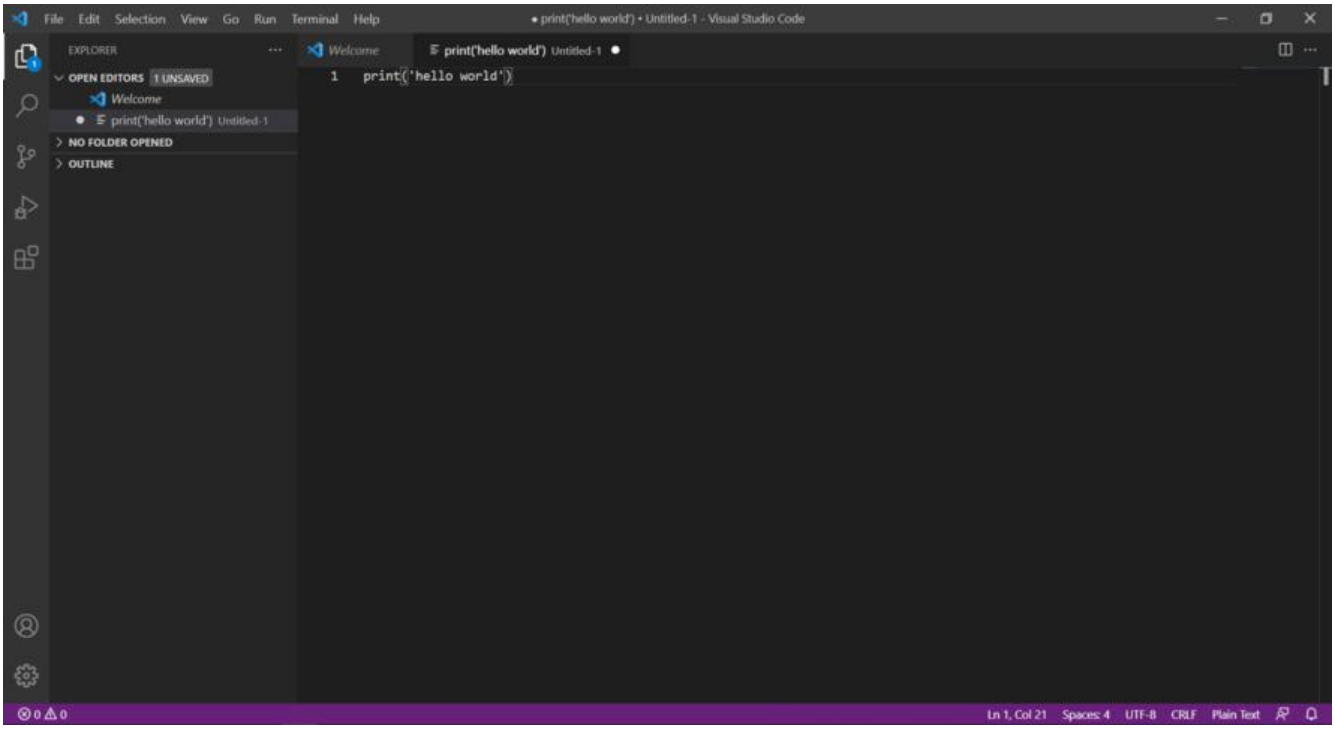
点击左上角的 File -> New File（或者按 Ctrl + N）即可创建新文件。



（2）编写代码

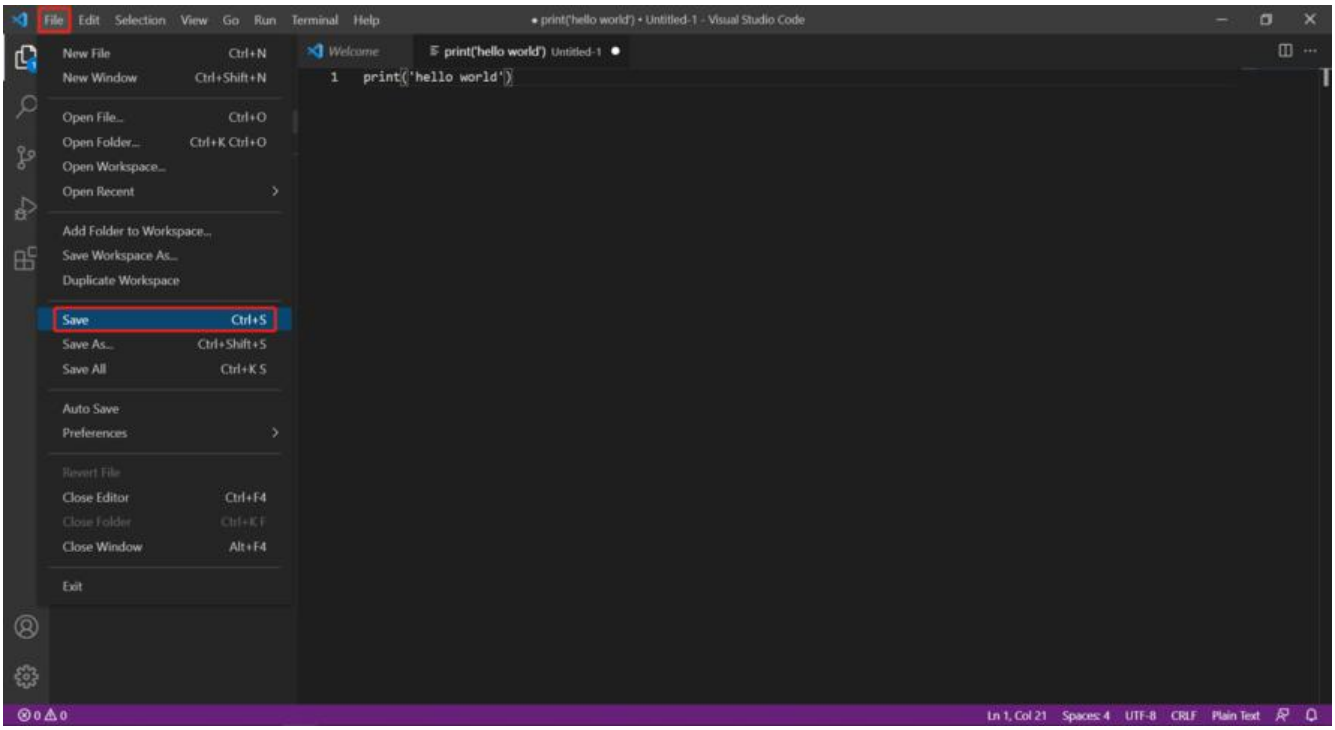
在编辑栏内输入 print('hello world')。

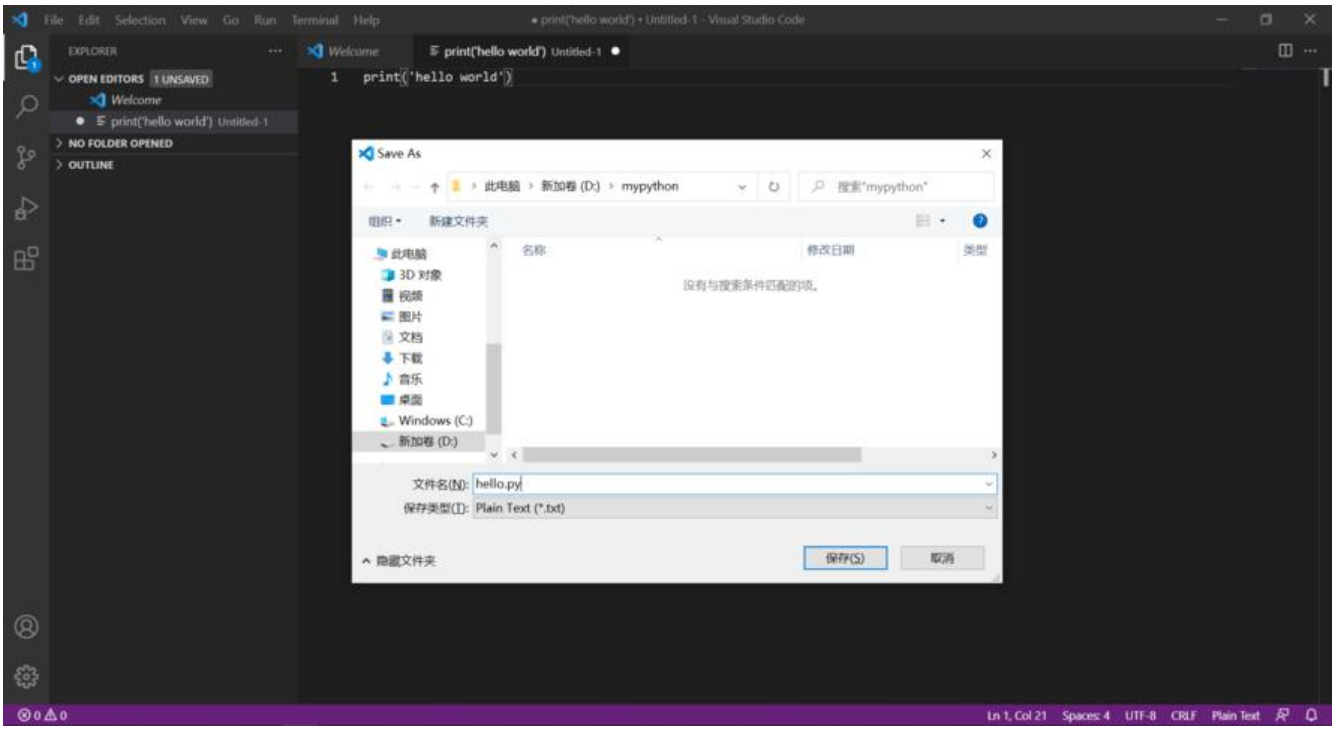
print() 函数用于在控制台打印内容。



（3）保存文件

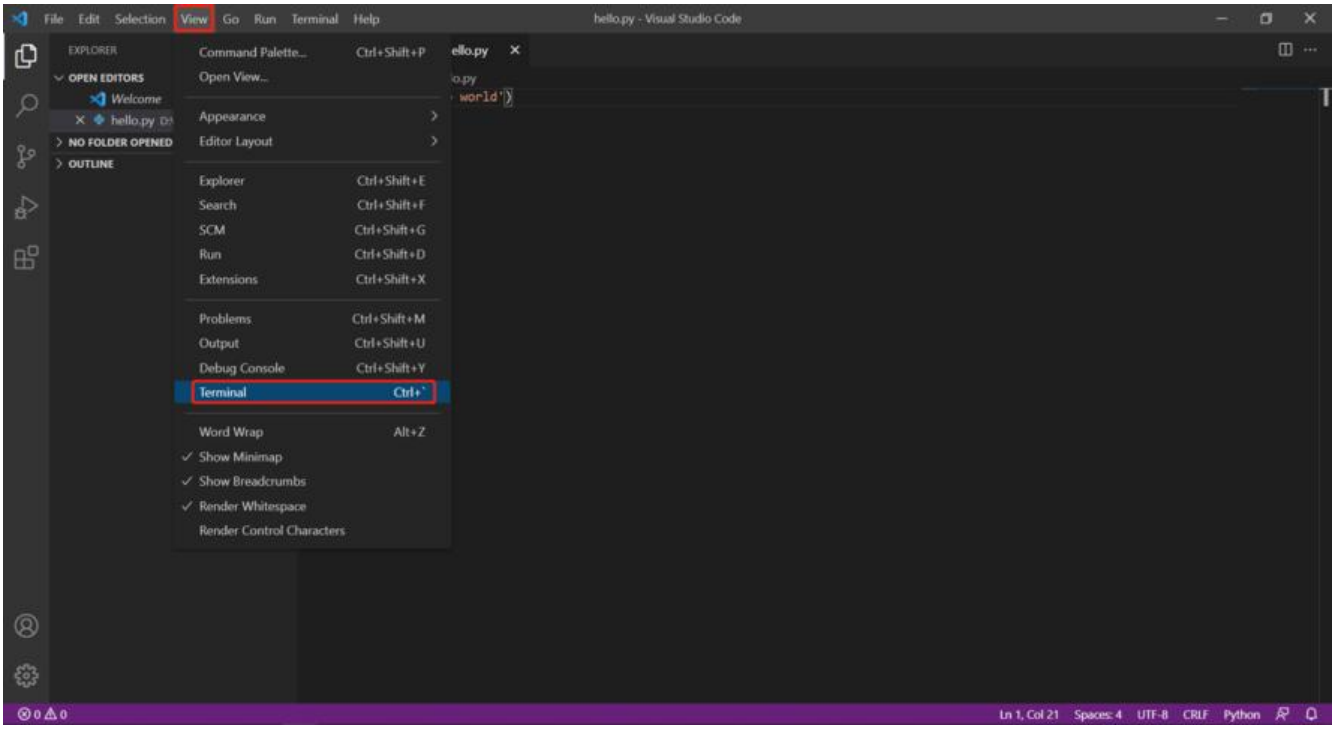
点击左上角的 File -> Save（或者按 Ctrl + S），在弹出框内选择保存路径（图例中保存在 D:\mypyton），并命名为 hello.py 后点击保存。

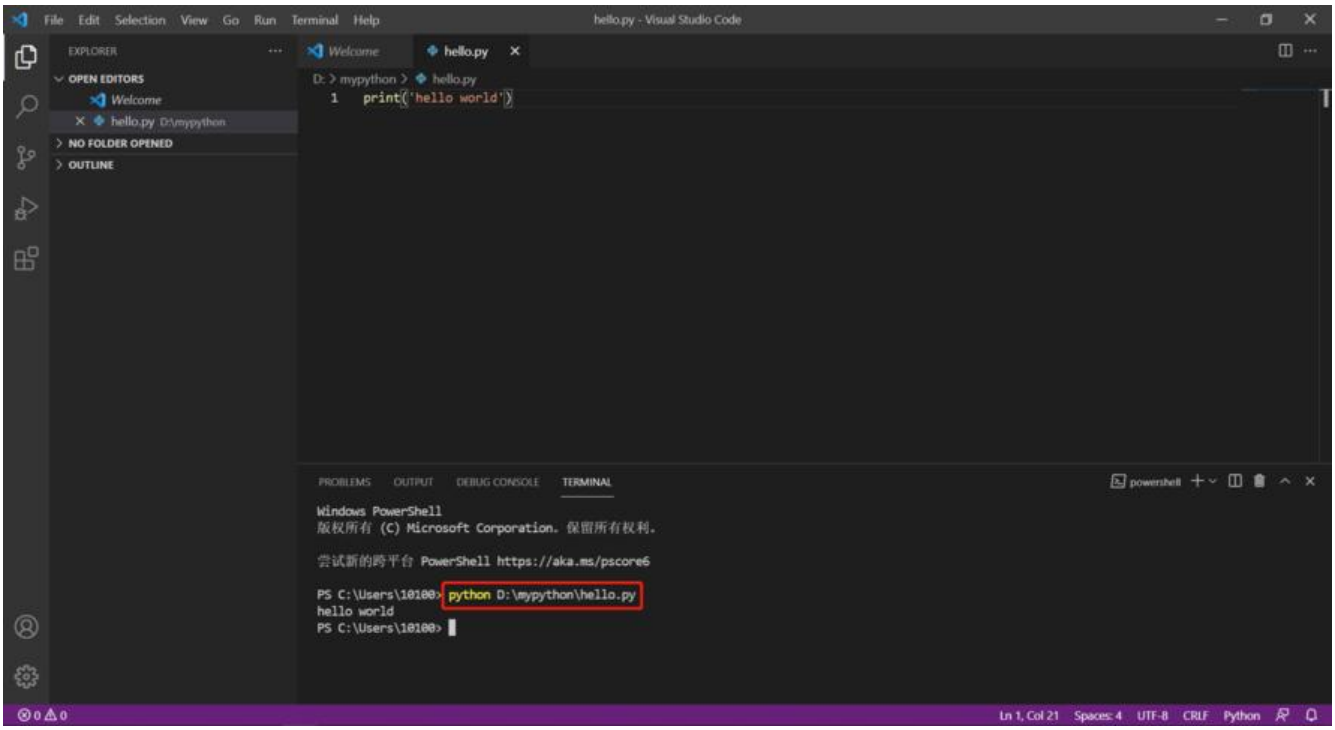




（4）执行程序文件

点击 View -> Terminal（或者按 Ctrl + `（数字 1 左边的键）打开命令行终端，在命令行终端内输入 python 文件完整路径（文件完整路径要与上一步的保存路对应，图例中为 python D:\mypython\hello.py）回车，即可执行程序。





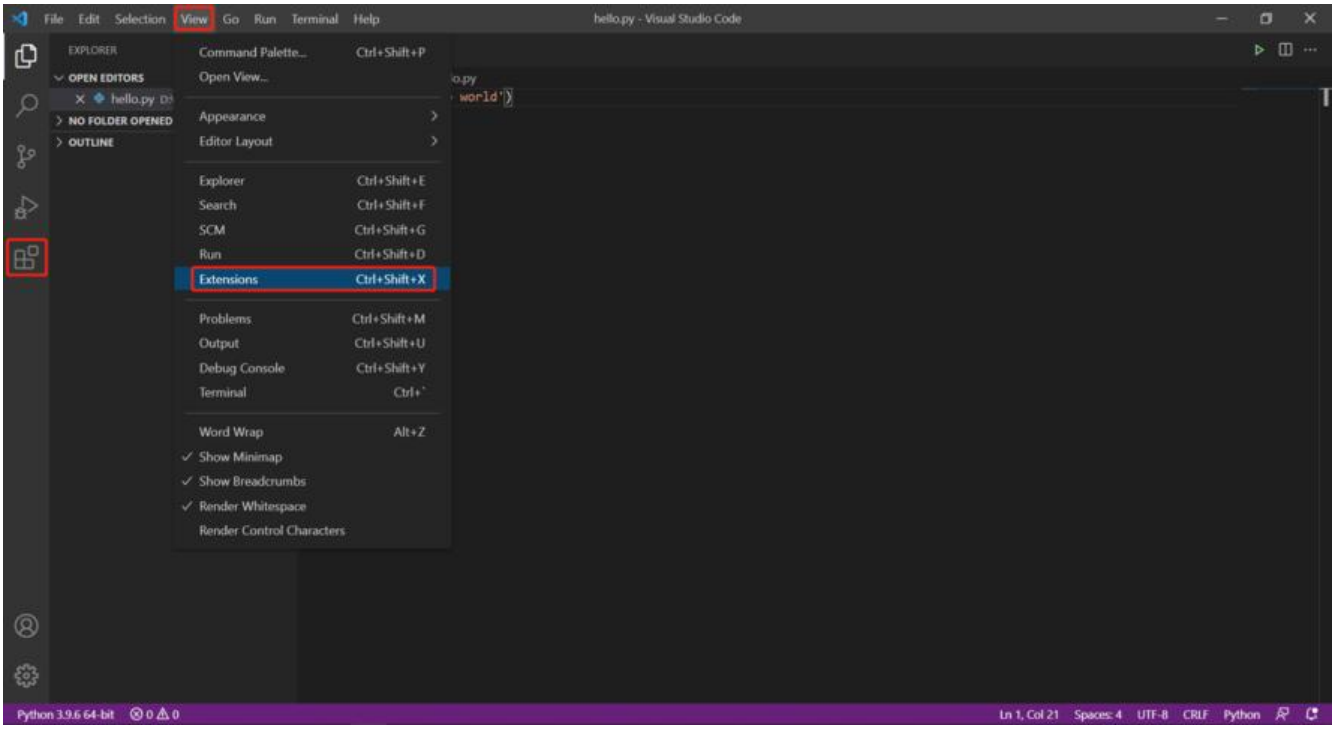
3、给 VS Code 安装 Python 插件

如果每次执行 Python 程序，都要在命令行终端手动输入命令就太麻烦了。因此，我们可以给 VS Cod 安装一个 Python 插件，这个插件不仅可以方便我们执行程序，也可以帮助我们进行代码提示、代码查等。

安装步骤：

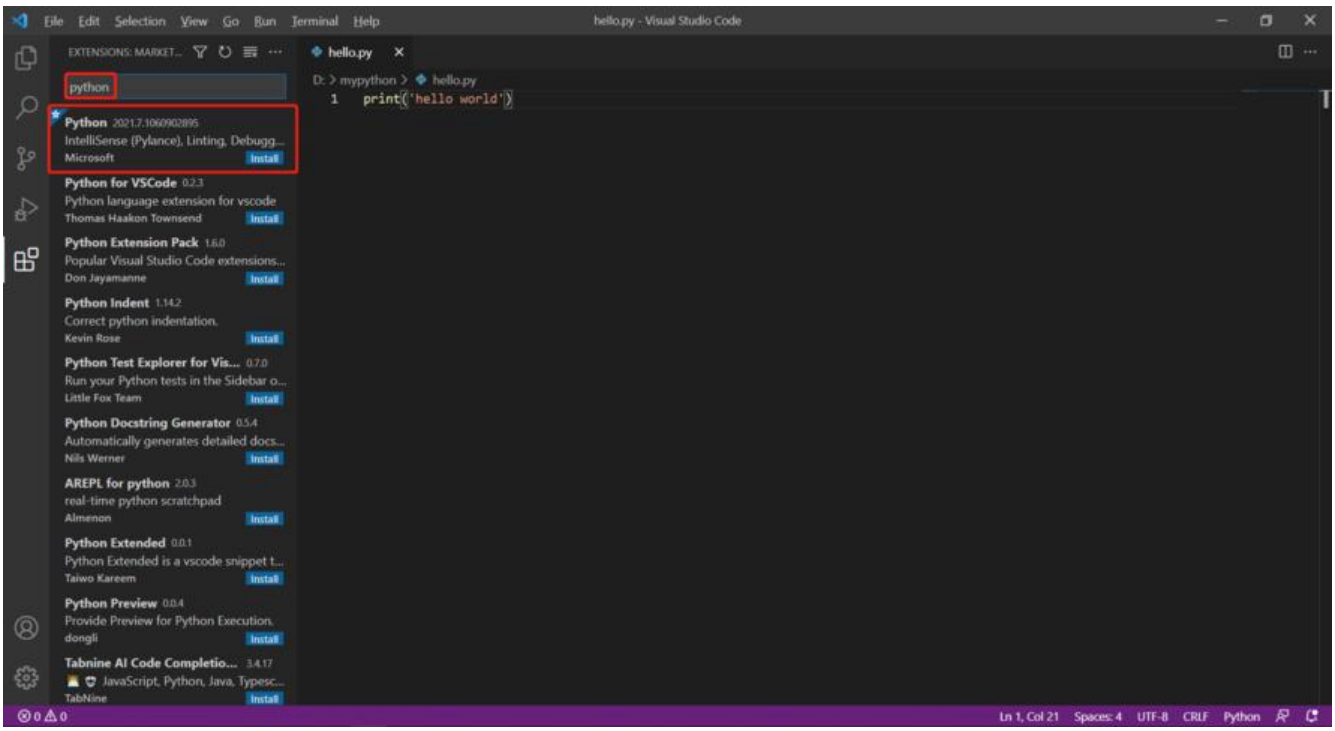
（1）打开插件市场

点击 View -> Extensions（或者按 Ctrl + Shift + X，又或者点击左侧活动栏的第五个图标）打开插市场。



（2）搜索插件并安装

在插件市场的搜索框内输入“python”搜索插件，找到名为“Python”的插件（一般在第一个），击“Install”即可进行安装。



安装完 Python 插件后，在编辑栏内点击右键 -> Run Python File in Terminal，即可执行 Python 件。

三、Python 基本代码结构

现在我们来看一下 Python 文件的基本代码结构，先有个整体性的认识。看得懂最好，看不懂也没关，可以等学完后再回过头来看。

```
#!/usr/bin/env python3                  # 指定解释器，windows 系统下执行时可省略，但为了可移植最好加上
# -*- coding: utf-8 -*-                 # 指定编码格式，python 3.x 版本可省略，但为了可移植性最好上

'''
该模块用于展示 Python 基本代码结构
'''
                                        # 模块说明

import sys, os                          # 导入模块

debug = True                            # 定义全局变量

class ClassName(object):                # 定义类，(object) 表示继承 object 类，可以不写 (object)
    pass

def func():                             # 定义函数
    pass
```

```
if __name__ == '__main__':          # 是否主函数，只有该文件作为执行入口时，下面代码才会执
，一般用于测试本模块
    c = ClassName()
    func()
```

- pass 是空语句，可以起占位作用。当你还没写业务代码时，可以使用 pass 来保持语法结构的完整。
- if __name__ == '__main__' 用于判断当前文件是不是程序的执行入口。当通过当前文件执行程序时下面的代码就会被执行，一般用于测试本模块中的代码。

四、基础语法

1、变量与赋值

变量用于存储数据，而赋值就是将数据存到变量的过程（给数据取别名）。

Python 中，可以使用任意数据给变量重新赋值，此时新的数据会替换旧的数据。

赋值写法：变量名 = 变量值。

- 变量名可以随便取，但是必须符合标识符规则（下面会讲到）。
- 变量值可以是一个数据值、表达式（由运算符和操作数构成）或函数。
- 这里的 =，不是我们在数学中理解的“等于”，它的含义是“赋值”。

示例：

```
# 将数字数据赋值给变量 a
a = 1
# 将列表数据赋值给变量 a
a = [1, 2, 3]

# 将表达式 1 + 2 的结果值赋值给变量 b
b = 1 + 2
# 将表达式 b + 1 的结果值赋值给变量 b
b = b + 1

# 将函数的返回值赋值给变量 c
c = abs(-1)
```

全局变量与局部变量：

- 全局变量：定义在函数外的变量。全局变量在任何地方都能使用。
- 局部变量：定义在函数内的变量。局部变量只能在函数内使用。

2、标识符

标识符用于给变量、函数等命名。换句话说，我们给变量、函数等取的名称，就是标识符。

标识符组成规则：

- 由字母、数字、下划线组成。

- 第一个字符必须是字母或下划线。
- 区分大小写。

3、关键字

关键字又称保留字，是被 Python 语言赋予了特殊含义的单词。

不能使用关键字作为标识符。

我们使用的 Python 3.9.6 版本有 36 个关键字：False，None，True，__peg_parser__，and，as，assert，async，await，break，class，continue，def，del，elif，else，except，finally，for，from，lobal，if，import，in，is，lambda，nonlocal，not，or，pass，raise，return，try，while，with，ield。

4、注释

注释是用来解释代码的。

为了防止自己以后或者别人看不懂代码，我们可以使用注释来解释你的代码是用来干嘛的或者为什么这么写。计算机在执行 Python 代码时，会自动忽略注释内容。

其实负责执行 Python 代码的是 Python 解释器，它会将 Python 语言翻译成计算机能看得懂的机器语。但我们可以把 Python 解释器简单地理解为计算机的一部分。

（1）单行注释

单行注释以 # 开头，# 后为注释内容。

示例：

```
# 向世界问好
print('hello world')  # 向世界问好
```

（2）多行注释

多行注释以成对的 '''（3 个英文单引号）或 """（3 个英文双引号）包裹。

示例：

```
'''
向世界问好
向世界问好
'''

"""
向世界问好
向世界问好
"""
print('hello world')
```

5、缩进

Python 中使用缩进来表示代码块。同一层级（包含子层级）的代码堆叠在一起，就是代码块。缩进以使用 Tab 符号或空格，但同一代码块内不能混用。一般使用 4 个空格作为 1 个缩进。

关于如何缩进，我们只需要记住以下两点：

1. 同一代码块的语句必须包含相同的缩进，简单的说，就是要对齐。
2. 遇到某行代码结尾有 `:`（英文冒号）时，紧跟着的代码块必须再次缩进。

示例：

```
a = -1
if a < 0:
    print('a 是负数')
    a = -a
else:
    print('a 不是负数')
print('a 的绝对值为:', a)
```

以上代码可以提取出 3 个代码块：

- 代码块 1：由所有代码构成，包含 0 个缩进。
- 代码块 2：由 `print('a 是负数')` 和 `a = -a` 构成，包含 1 个缩进。
- 代码块 3：由 `print('a 不是负数')` 单独构成，包含 1 个缩进。

由于代码块 2 和代码块 3 紧跟在 `:` 后面，需要再次缩进，因此包含 1 个缩进。

五、基本数据类型

Python 提供了六种基本数据类型，用于给我们创建基本类型的数据，包括数字、字符串、列表、元、字典、集合。

根据数据创建完后可不可以被修改，可分为：

- 不可变数据：数字、字符串、元组。
- 可变数据：列表、字典、集合。

可以使用 `type()` 函数来查看数据的类型：

```
>>> type(1)
<class 'int'>
```

1、数字

Python 中的数字类型包含：整数、浮点数、布尔类型、复数。

- 整数：这个没什么好说的，就是 1、2、-1 这种。
- 浮点数：其实就是小数，像 1.2、2.3 这种。
- 布尔类型（布尔值）：布尔类型只有 `True`（真值，可理解为“是”）、`False`（假值，可理解为否”） 两种值，要么 `True`，要么 `False`。用于做非此即彼的判断。
- 复数：形如 `a + bj`，`a` 为实部，`b` 为虚部，`j` 为虚数单位。这个了解即可，一般不会用到。

（1）创建数字

语法格式：

```
变量 = 数字值
```

示例：

```
# 创建整数，并赋值给变量 a
a = 1

# 创建浮点数，并赋值给变量 b
b = 1.2

# 创建布尔类型数据，并赋值给变量 c
c = True

# 创建复数，并赋值给变量 d
d = 1 + 2j
```

（2）常见数字操作

表格中，x、x1、x2 为要操作的数字，r 为接收操作结果的变量。

操作	如何操作	操作说明	作示例
加、减、乘、除	使用 +、-、*、/ 运算符	-	= 1 + 2
将数字转换为整数	使用 int(x) 函数	int() 也可以将	字字符串转换为整数 r = int(1.2)
将数字转换为浮点数	使用 float(x) 函数	float() 也	以将数字字符串转换为浮点数 r = float(1)
获取绝对值	使用 abs(x) 函数	-	= abs(-1)
向上取整	使用 math.ceil(x) 函数	需要导入 math	块，操作结果为大于等于 x 的最小整数 r = math.ceil(4.5)
向下取整	使用 math.floor(x) 函数	需要导入 math	模块，操作结果为小于等于 x 的最大整数 r = math.floor(4.5)
获取多个数字的最大值	使用 max(x1, x2, ...) 函数	-	= max(1, 2, 3)
获取多个数字的最小值	使用 min(x1, x2, ...) 函数	-	= min(1, 2, 3)
四舍五入保留 n 位小数	使用 round(x, n) 函数	n	保留小数位 r = round(4.543, 2)

示例：

```
import math

# 加、减、乘、除、求余
```

```
r = 1 + 2
print('1 加 2 结果为', r)  # 输出：1 加 2 结果为 3
r = 2 - 1
print('2 减 1 结果为', r)  # 输出：2 减 1 结果为 1
r = 2 * 3
print('2 乘 3 结果为', r)  # 输出：2 乘 3 结果为 6
r = 3 / 2
print('3 除 2 结果为', r)  # 输出：3 除 2 结果为 1.5

# 将数字转换为整数
r = int(1.2)
print('1.2 转换为整数的结果为', r)  # 输出：1.2 转换为整数的结果为 1

# 将数字转换为浮点数
r = float(1)
print('1 转换为浮点数的结果为', r)  # 输出：1 转换为浮点数的结果为 1.0

# 获取绝对值
r = abs(-1)
print('-1 的绝对值为', r)  # 输出：-1 的绝对值为 1

# 向上取整
r = math.ceil(4.5)
print('4.5 向上取整的结果为', r)  # 输出：4.5 向上取整的结果为 5

# 向下取整
r = math.floor(4.5)
print('4.5 向下取整的结果为', r)  # 输出：4.5 向下取整的结果为 4

# 获取多个数字的最大值
r = max(1, 2, 3)
print('数字 1、2、3 的最大值为', r)  # 输出：数字 1、2、3 的最大值为 3

# 获取多个数字的最小值
r = min(1, 2, 3)
print('数字 1、2、3 的最小值为', r)  # 输出：数字 1、2、3 的最小值为 1

# 四舍五入保留 n 位小数
r = round(4.543, 2)
print('4.543 四舍五入保留 2 位小数的结果为', r)  # 输出：4.543 四舍五入保留 2 位小数的结果为 4.5
```

2、字符串

字符串可理解为文本。

（1）创建字符串

Python 中使用成对的 '（英文单引号）或 "（英文双引号）来创建字符串。

● 由于 '、" 用于创建字符串，因此不能随意出现在字符串内容中，此时，我们可以用 \'、\" 来代替。\ 称为转义符，可以将某个符号的含义转成另一种含义。（也可以交替使用 '、" 或者使用三引号（'''、"""，这个我们不细讲。）

● 注意字符串中的内容哪怕跟代码再像，它也不是代码。因此，如果看到字符串中出现了跟变量、表式等一样的内容，那它们也不是变量或表达式。

● 注意不要混淆数字字符串与数字，数字字符串形如 '123'，数字形如 123。

语法格式：

```
变量 = '字符串内容'
变量 = "字符串内容"
```

示例：

```
# 创建字符串，并赋值给变量 s
s = '在学 Python，很忙！'
s = "在学 Python，很忙！"

# 创建包含单引号的字符串，并赋值给变量 s
s = '在学 \'Python\'，很忙！'
s = "在学 'Python'，很忙！"

# 创建包含双引号的字符串，并赋值给变量 s
s = "在学 \"Python\"，很忙！"
s = '在学 "Python"，很忙！'

# 创建内容与代码相似的字符串，并赋值给变量 s
s = 'a + 1'  # 这里的 a + 1 不是表达式代码，只是长得像而已

# 创建数字字符串，并赋值给变量 s
s = '123'

# 创建空字符串，并赋值给变量 s
s = ''
s = ""
```

（2）常见字符串操作

表格中，x、x1、x2 为要操作的字符串，r 为接收操作结果的变量。

操作	如何操作	操作说明	操作示例
字符串拼接	使用 + 运算符	将多个字符串拼接成个新的字符串	r = x1 + x2
字符串截取	使用 [:] 运算符	截取字符串一部分，号两边为开始、结束索引，遵循左闭右开（包含开始索引的字符，不包含结束索引的字符）	= x[0:4]
字符串格式化	使用 % 运算符	用右边数据依次替左边字符串内的占位符（常用 %s），右边有多个数据时要用 () 包起来，并用逗号分隔	= 'hello,%s' % 'world'
获取字符串长度	使用 len(x) 函数	字符串内的字符数	r = len(x)
去除字符串两端空格	使用 x.strip() 函数	-	= x.strip()

根据分隔符分割字符串 使用 x.split(s) 函数 s 为分符 r = x.split(',')

在字符串中查找某一字符串 使用 x.find(s) 函数 s 要查找的字符串，结果为索引值，找不到则为 -1 r = x.find('hello')

判断字符串是否以某一字符串开头 使用 x.startswith(s) 函数 为开头字符串 r = x.startswith('http')

判断字符串是否以某一字符串结尾 使用 x.endswith(s) 函数 为结尾字符串 r = x.endswith('.jpg')

大写字母转小写 使用 x.lower() 函数 - = x.lower()

小写字母转大写 使用 x.upper() 函数 - = x.upper()

字符串替换 使用 x.replace(s1, s2) 函数 将 x 字符中的 s1 字符串替换为 s2 字符串 r = x.replace('hello', 'hi')

示例：

```
# 字符串拼接
r = 'hello' + 'world'
print('字符串拼接结果为', r)  # 输出：字符串拼接结果为 helloworld

# 字符串截取
x = 'hello,world'
r = x[0:4]
print('从索引 0 到 4 截取字符串，结果为', r)  # 输出：从索引 0 到 4 截取字符串，结果为 hell

# 字符串格式化
r = 'hello,%s' % 'world'  # 替换一个数据
print('字符串格式化结果为', r)  # 输出：字符串格式化结果为 hello,world
r = 'hello,%s! hello,%s!' % ('world', 'python')  # 替换多个数据
print('字符串格式化结果为', r)  # 输出：字符串格式化结果为 hello,world! hello,python!

# 获取字符串长度
x = 'hello,world'
r = len(x)
print('字符串长度为', r)  # 输出：字符串长度为 11

# 去除字符串两端空格
r = '  hello,world  '.strip()
print('去除两端空格的结果为', r)  # 输出：去除两端空格的结果为 hello,world

# 根据分隔符分割字符串
x = 'hello,world'
r = x.split(',')
print('根据逗号分割字符串，结果为', r)  # 输出：根据逗号分割字符串，结果为 ['hello', 'world']

# 在字符串中查找某一字符串
x = 'hello,world'
r = x.find('ello')
print('字符串中查找 ello，结果为', r)  # 输出：字符串中查找 ello，结果为 1
```

```
# 判断字符串是否以某一字符串开头
x = 'hello,world'
r = x.startswith('hello')
print('判断字符串是否以 hello 开头，结果为', r)  # 输出：判断字符串是否以 hello 开头，结果为 Tru


# 判断字符串是否以某一字符串结尾
x = 'hello,world'
r = x.endswith('world')
print('判断字符串是否以 world 结尾，结果为', r)  # 输出：判断字符串是否以 world 结尾，结果为 Tr
e

# 大写字母转小写
x = 'HELLO,world'
r = x.lower()
print('大写字母转小写，结果为', r)  # 输出：大写字母转小写，结果为 hello,world

# 小写字母转大写
x = 'HELLO,world'
r = x.upper()
print('小写字母转大写，结果为', r)  # 输出：小写字母转大写，结果为 HELLO,WORLD

# 字符串替换
x = 'hello,world'
r = x.replace('hello', 'hi')
print('字符串替换后，结果为', r)  # 输出：字符串替换后，结果为 hi,world
```

3、列表

Python 中的列表可理解为 Excel 表格中的某一列，列中每个单元格带有序号，我们可以根据序号找某个单元格，从而操作单元格的数据。

列表的每个位置也带有序号，序号从 0 开始，依次递增，这个序号被称为索引，每个位置存储的数据称为元素。列表中可存放不同类型的数据，并且可以随时添加、删除里面的数据。

（1）创建列表

Python 中使用 [] 来创建列表，并使用 ,（英文逗号）分隔各个数据。

语法格式：

```
变量 = [数据1, 数据2, ..., 数据n]
```

示例：

```
# 创建名字列表，并赋值给变量 names
names = ['小白', '小黑', 'Tom', 'Jerry']

# 创建分数列表，并赋值给变量 scores
scores = [90, 80, 85, 85]

# 创建包含名字和分数的列表，并赋值给变量 data
data = ['小白', '小黑', 'Tom', 'Jerry', 90, 80, 85, 85]
```

```
# 创建空列表，并赋值给变量 data
data = []
```

（2）常见列表操作

表格中，x、x1、x2 为要操作的列表，r 为接收操作结果的变量。

操作	如何操作	操作说明	操作示例
根据索引获取列表中的数据	使用 [] 运算符	方括号为索引	r = x[1]
根据索引修改列表中的数据	使用 []、= 运算符	[] 运算符用于定位数据，= 运算符用于修改数据	x[1] = 2
根据索引删除列表中的数据	使用 []、del 运算符	[] 运算符用于定位数据，del 运算符用于删除数据	del x[1]
列表拼接	使用 + 运算符	将两个列表拼接成一个的列表	r = x1 + x2
列表截取	使用 [:] 运算符	截取列表的一部分，冒两边为开始、结束索引，遵循左闭右开	r = x[0:4]
判断数据是否在列表中	使用 in 运算符	操作结果布尔值，True 表示在，False 表示不在	r = 2 in x
获取列表长度	使用 len(x) 函数	列表中的数据个数	= len(x)
获取列表中数据最大值	使用 max(x) 函数	列表中数据必须是相同类型	r = max(x)
获取列表中数据最小值	使用 min(x) 函数	列表中数据必须是相同类型	r = min(x)
在列表末尾添加新数据	使用 x.append(d) 函数	d 要添加的数据	x.append(3)
删除列表中第一个匹配数据	使用 x.remove(d) 函数	为要删除的数据	x.remove(3)
列表排序	使用 x.sort() 函数	默认升序，降序则使用 x.sort(reverse=True)	x.sort()
清空列表	使用 x.clear() 函数	清空列表中的所有数据	.clear()

示例：

```
# 根据索引获取列表中的数据
x = [0, 1, 2, 3, 4, 5]
r = x[1]
print('索引 1 的数据为', r)  # 输出：索引 1 的数据为 1

# 根据索引修改列表中的数据
x = [0, 1, 2, 3, 4, 5]
x[1] = 3
print('索引 1 的数据修改后为', x[1])  # 输出：索引 1 的数据修改后为 3
```

```
# 根据索引删除列表中的数据
x = [0, 1, 2, 3, 4, 5]
del x[1]
print('删除索引 1 的数据后，列表为', x)  # 输出：删除索引 1 的数据后，列表为 [0, 2, 3, 4, 5]

# 列表拼接
r = [0, 1, 2] + [3, 4, 5]
print('列表拼接结果为', r)  # 输出：列表拼接结果为 [0, 1, 2, 3, 4, 5]

# 列表截取
x = [0, 1, 2, 3, 4, 5]
r = x[0:4]
print('从索引 0 到 4 截取列表，结果为', r)  # 输出：从索引 0 到 4 截取列表，结果为 [0, 1, 2, 3]

# 判断数据是否在列表中
x = [0, 1, 2, 3, 4, 5]
r = 2 in x
print('判断 2 是否在列表中，结果为', r)  # 输出：判断 2 是否在列表中，结果为 True

# 获取列表长度
x = [0, 1, 2, 3, 4, 5]
r = len(x)
print('列表长度为', r)  # 输出：列表长度为 6

# 获取列表中数据最大值
x = [0, 1, 2, 3, 4, 5]
r = max(x)
print('列表中数据最大值为', r)  # 输出：列表中数据最大值为 5

# 获取列表中数据最小值
x = [0, 1, 2, 3, 4, 5]
r = min(x)
print('列表中数据最小值为', r)  # 输出：列表中数据最小值为 0

# 在列表末尾添加新数据
x = [0, 1, 2, 3, 4, 5]
x.append(6)
print('在列表末尾添加新数据后，列表为', x)  # 输出：在列表末尾添加新数据后，列表为 [0, 1, 2, 3, 4,
5, 6]

# 删除列表中第一个匹配数据
x = [0, 1, 2, 3, 4, 5]
x.remove(2)
print('删除列表中第一个匹配数据后，列表为', x)  # 输出：删除列表中第一个匹配数据后，列表为 [0,
, 3, 4, 5]

# 列表排序
x = [3, 1, 4, 2, 5, 0]
x.sort()
print('排序后的列表为', x)  # 输出：排序后的列表为 [0, 1, 2, 3, 4, 5]

# 清空列表
x = [0, 1, 2, 3, 4, 5]
x.clear()
```

```
print('清空后的列表为', x)  # 输出：清空后的列表为 []
```

4、元组

元组与列表基本类似，不同之处在于元组内的数据不能被修改。

（1）创建元组

Python 中使用 () 来创建元组，并使用 ,（英文逗号）分隔各个数据。

语法格式：

```
变量 = (数据1, 数据2, ..., 数据n)
```

创建只有一个数据的元组时，必须在数据后面加 ,，否则 () 会被当做运算符，此时创建的将不是一个组。

示例：

```
# 创建名字元组，并赋值给变量 names
names = ('小白', '小黑', 'Tom', 'Jerry')

# 创建分数元组，并赋值给变量 scores
scores = (90, 80, 85, 85)

# 创建包含名字和分数的元组，并赋值给变量 data
data = ('小白', '小黑', 'Tom', 'Jerry', 90, 80, 85, 85)

# 创建只有一个数据的元组，并赋值给变量 data
data = (90,)

# 创建空元组，并赋值给变量 data
data = ()
```

（2）常见元组操作

表格中，x、x1、x2 为要操作的元组，r 为接收操作结果的变量。

操作	如何操作	操作说明	操作示例
根据索引获取元组中的数据	使用 [] 运算符	方括号为索引	r = x[1]
元组拼接	使用 + 运算符	将两个元组拼接成一个的元组	r = x1 + x2
元组截取	使用 [:] 运算符	截取元组的一部分，冒两边为开始、结束索引，遵循左闭右开	r = x[0:4]
判断数据是否在元组中	使用 in 运算符	操作结果布尔值，True 表示在，False 表示不在	r = 2 in x
获取元组长度（数据个数）	使用 len(x) 函数	元组的数据个数	r = len(x)
获取元组中数据最大值	使用 max(x) 函数	元组中	

数据必须是相同类型 r = max(x)

获取元组中数据最小值 使用 min(x) 函数 元组中数据必须是相同类型 r = min(x)

示例：

```
# 根据索引获取元组中的数据
x = (0, 1, 2, 3, 4, 5)
r = x[1]
print('索引 1 的数据为', r)  # 输出：索引 1 的数据为 1

# 元组拼接
r = (0, 1, 2) + (3, 4, 5)
print('元组拼接结果为', r)  # 输出：元组拼接结果为 (0, 1, 2, 3, 4, 5)

# 元组截取
x = (0, 1, 2, 3, 4, 5)
r = x[0:4]
print('从索引 0 到 4 截取元组，结果为', r)  # 输出：从索引 0 到 4 截取元组，结果为 (0, 1, 2, 3)

# 判断数据是否在元组中
x = (0, 1, 2, 3, 4, 5)
r = 2 in x
print('判断 2 是否在元组中，结果为', r)  # 输出：判断 2 是否在元组中，结果为 True

# 获取元组长度
x = (0, 1, 2, 3, 4, 5)
r = len(x)
print('元组长度为', r)  # 输出：元组长度为 6

# 获取元组中数据最大值
x = (0, 1, 2, 3, 4, 5)
r = max(x)
print('元组中数据最大值为', r)  # 输出：元组中数据最大值为 5

# 获取元组中数据最小值
x = (0, 1, 2, 3, 4, 5)
r = min(x)
print('元组中数据最小值为', r)  # 输出：元组中数据最小值为 0
```

5、字典

Python 中的字典就像我们上学时经常用的《新华字典》，《新华字典》中存储了汉字以及对应的汉解释，我们可以通过某个汉字找到对应的汉字解释。

字典中存储了键值对（键和值），键对应《新华字典》中的汉字，值对应《新华字典》中的汉字解释键可理解为别名，值为对应的数据，我们可以根据键从字典中找到对应的值。

字典中的键必须唯一，而值则不用。另外，键必须是不可变的（比如数字、字符串），而值可以是任数据类型。

（1）创建字典

Python 中使用 {:} 来创建字典，:（英文冒号）用于创建键值对，并使用 ,（英文逗号）分隔各个键值。

语法格式：

```
变量 = {键1: 值1, 键2: 值2, ..., 键n: 值n}
```

示例：

```
# 创建保存学生信息的字典，并赋值给变量 student
student = {'名字': '小白', '分数': 90}

# 创建保存座位号及对应学生名字的字典，并赋值给变量 seats
seats = {1: '小白', 2: '小黑', 3: 'Tom', 4: 'Jerry'}

# 创建保存名字及对应分数的字典，并赋值给变量 data
data = {'小白': 90, '小黑': 80, 'Tom': 85, 'Jerry': 85}

# 创建空字典，并赋值给变量 data
data = {}
```

（2）常见字典操作

表格中，x 为要操作的元组，r 为接收操作结果的变量。

操作	如何操作	操作说明	操作示例
根据键获取数据	使用 [] 运算符	方括号内为键	= x['name']
新增或修改数据	使用 []、= 运算符	[] 运算符用于定位数据，= 运算符用于修改数据	x['name'] = 'tom'
根据键删除数据	使用 []、del 运算符	[] 运算符用于定位数据，del 运算符用于删除数据	del x['name']
判断某个键是否在字典中	使用 in 运算符	操作结为布尔值，True 表示在，False 表示不在	r = 'name' in x
获取字典长度	使用 len(x) 函数	字典中的键值对个数	= len(x)
获取字典的所有键	使用 x.keys() 函数	获取到的不一个列表，可以使用 list() 函数转换为列表	r = x.keys()
获取字典的所有值	使用 x.values() 函数	获取到的是一个列表，可以使用 list() 函数转换为列表	r = x.values()
获取字典的所有键值对	使用 x.items() 函数	获取的不是一个列表，可以使用 list() 函数转换为列表	r = x.items()
清空字典	使用 x.clear() 函数	清空字典中的所有键对	x.clear()

示例：

```
# 根据键获取数据
x = {'name': '小白', 'age': 17}
```

```
r = x['name']
print('键为 name 的数据为', r)  # 输出：键为 name 的数据为 小白

# 新增数据
x = {'name': '小白', 'age': 17}
x['country'] = '中国'
print('新增数据后字典为', x)  # 输出：增数据后字典为 {'name': '小白', 'age': 17, 'country': '中国'}

# 修改数据
x = {'name': '小白', 'age': 17}
x['age'] = 27
print('键为 age 的数据修改后为', x['age'])  # 输出：键为 age 的数据修改后为 27

# 根据键删除数据
x = {'name': '小白', 'age': 17}
del x['age']
print('删除键为 age 的数据后，字典为', x)  # 输出：删除键为 age 的数据后，字典为 {'name': '小白'}

# 判断某个键是否在字典中
x = {'name': '小白', 'age': 17}
r = 'age' in x
print('判断键 age 是否在字典中，结果为', r)  # 输出：判断键 age 是否在字典中，结果为 True

# 获取字典长度
x = {'name': '小白', 'age': 17}
r = len(x)
print('字典长度为', r)  # 输出：字典长度为 2

# 获取字典的所有键
x = {'name': '小白', 'age': 17}
r = x.keys()
print('获取字典的所有键，结果为', r)  # 输出：获取字典的所有键，结果为 dict_keys(['name', 'age'])

# 获取字典的所有值
x = {'name': '小白', 'age': 17}
r = x.values()
print('获取字典的所有值，结果为', r)  # 输出：获取字典的所有值，结果为 dict_values(['小白', 17])

# 获取字典的所有键值对
x = {'name': '小白', 'age': 17}
r = x.items()
print('获取字典的所有键值对，结果为', r)  # 输出：获取字典的所有键值对，结果为 dict_items([('n
me', '小白'), ('age', 17)])

# 清空字典
x = {'name': '小白', 'age': 17}
x.clear()
print('清空后的字典为', x)  # 输出：清空后的字典为 {}
```

6、集合

Python 中的集合可理解为无序的、不重复的列表，集合中存储的数据是无序的、不重复的。集合中存放不同类型的数据。

- 由于集合是无序的，因此没有索引，我们不能根据索引获取数据。
- 由于集合是不重复的，因此即使我们保存了重复的数据，最终也只会保留一个。

（1）创建集合

Python 中使用 {} 创建集合，并使用 ,（英文逗号）分隔各个数据。

语法格式：

```
变量 = {数据1, 数据2, ..., 数据n}
```

示例：

```
# 创建名字列表，并赋值给变量 names
names = {'小白', '小黑', 'Tom', 'Jerry'}

# 创建分数列表，并赋值给变量 scores
scores = {90, 80, 85, 85}  # 两个 85，最终只会保留一个

# 创建包含名字和分数的列表，并赋值给变量 data
data = {'小白', '小黑', 'Tom', 'Jerry', 90, 80, 85, 85}  # 两个 85，最终只会保留一个

# 创建空集合，并赋值给变量 data
data = set()  # 注意：{} 用于创建空字典，因此创建空集合不能用 {}
```

（2）常见集合操作

表格中，x 为要操作的元组，r 为接收操作结果的变量。

操作	如何操作	操作说明	作示例
添加数据	使用 x.add(d)	d 为要添加的数据	.add(2)
删除数据	使用 x.remove(d)	d 为要删除的数据	.remove(2)
判断数据是否在集合中	使用 in 运算符	操作结果布尔值，True 表示在，False 表示不在	r = 2 in x
获取集合长度	使用 len(x) 函数	集合中的数据个数	= len(x)
获取集合中数据最大值	使用 max(x) 函数	集合中数据必须是相同类型	r = max(x)
获取集合中数据最小值	使用 min(x) 函数	集合中数据必须是相同类型	r = min(x)
清空集合	使用 x.clear() 函数	清空集合中的所有数据	.clear()

示例：

```
# 添加数据
x = {0, 1, 2, 3, 4, 5}
```

```
x.add(6)
print('添加数据后集合为', x)  # 输出：添加数据后集合为 {0, 1, 2, 3, 4, 5, 6}

# 删除数据
x = {0, 1, 2, 3, 4, 5}
x.remove(3)
print('删除数据集合为', x)  # 输出：删除数据集合为 {0, 1, 2, 4, 5}

# 判断数据是否在集合中
x = {0, 1, 2, 3, 4, 5}
r = 2 in x
print('判断 2 是否在集合中，结果为', r)  # 输出：判断 2 是否在集合中，结果为 True

# 获取集合长度
x = {0, 1, 2, 3, 4, 5}
r = len(x)
print('集合长度为', r)  # 输出：集合长度为 6

# 获取集合中数据最大值
x = {0, 1, 2, 3, 4, 5}
r = max(x)
print('集合中数据最大值为', r)  # 输出：集合中数据最大值为 5

# 获取集合中数据最小值
x = {0, 1, 2, 3, 4, 5}
r = min(x)
print('集合中数据最小值为', r)  # 输出：集合中数据最小值为 0

# 清空集合
x = {0, 1, 2, 3, 4, 5}
x.clear()
print('清空后的集合为', x)  # 输出：清空后的集合为 set()
```

六、运算符

Python 提供了七种类型的运算符，用于给我们操作数据。

1、算术运算符

算术运算符用于对数据进行算术运算。

运算符	描述	示例
+	加	a + b
-	减	a - b
*	乘	a * b
/	除	a / b
%	取模（求余）	a % b
**	幂（乘方）	a ** b
//	向下整除（小于等于相除结果的最大整数）	

// b

示例：

```
a = 1 + 2
print('a 的结果为', a)  # 输出：a 的结果为 3

b = 2 - 1
print('b 的结果为', b)  # 输出：b 的结果为 1

c = 2 * 3
print('c 的结果为', c)  # 输出：c 的结果为 6

d = 3 / 2
print('d 的结果为', d)  # 输出：d 的结果为 1.5

e = 3 % 2
print('e 的结果为', e)  # 输出：e 的结果为 1

f = 2 ** 2
print('f 的结果为', f)  # 输出：f 的结果为 4

g = 3 // 2
print('g 的结果为', g)  # 输出：g 的结果为 1
```

2、比较（关系）运算符

比较运算符用于比较两个数据的关系，操作结果是一个布尔值。

运算符	描述	示例
==	等于	a == b
!=	不等于	a != b
>	大于	a > b
<	小于	a < b
>=	大于等于	a >= b
<=	小于等于	a <= b

示例：

```
a = 1 == 1
print('判断 1 是否等于 1，结果为', a)  # 输出：判断 1 是否等于 1，结果为 True
a = 1 == 2
print('判断 1 是否等于 2，结果为', a)  # 输出：判断 1 是否等于 2，结果为 False

b = 1 != 2
print('判断 1 是否不等于 2，结果为', b)  # 输出：判断 1 是否不等于 2，结果为 True
b = 1 != 1
print('判断 1 是否不等于 1，结果为', b)  # 输出：判断 1 是否不等于 1，结果为 False

c = 2 > 1
```

```
print('判断 2 是否大于 1，结果为', c)  # 输出：判断 2 是否大于 1，结果为 True
c = 2 > 2
print('判断 2 是否大于 2，结果为', c)  # 输出：判断 2 是否大于 2，结果为 False

d = 1 < 2
print('判断 1 是否小于 2，结果为', d)  # 输出：判断 1 是否小于 2，结果为 True
d = 1 < 1
print('判断 1 是否小于 1，结果为', d)  # 输出：判断 1 是否小于 1，结果为 False

e = 1 >= 1
print('判断 1 是否大于等于 1，结果为', e)  # 输出：判断 1 是否大于等于 1，结果为 True
e = 1 >= 2
print('判断 1 是否大于等于 2，结果为', e)  # 输出：判断 1 是否大于等于 2，结果为 False

f = 1 <= 1
print('判断 1 是否小于等于 1，结果为', f)  # 输出：判断 1 是否小于等于 1，结果为 True
f = 2 <= 1
print('判断 2 是否小于等于 1，结果为', f)  # 输出：判断 2 是否小于等于 1，结果为 False
```

3、赋值运算符

赋值运算符用于给变量赋值。最常用的赋值运算符为 =（简单赋值），对于刚入门的小伙伴，只需要用这个就足够了。

除此之外，还包括 +=（加法赋值）、-=（减法赋值）、*=（乘法赋值）、/=（除法赋值）、%=（模赋值）、**=（幂赋值）、//=（整除赋值） 等。

示例：

```
a = 2
print('a 的结果为', a)  # 输出：a 的结果为 2
```

4、逻辑运算符

逻辑运算符用于根据已有的一个或多个条件计算出最终结果，操作结果是一个布尔值。

运算符	描述	示例
and	与（如果 and 左边的值为 False，则结果为左边的值，否则为	a and b
边的值)		
or	或（如果 or 左边的值为 True，则结果为左边的值，否则为右	a or b
的值)		
not	非（如果值为 True，则结果为 False，如果值为 Flase，则结	not a
为 True)		

示例：

```
a = False and 1
print('a 的结果为', a)  # 输出：a 的结果为 False

b = True and 1
print('b 的结果为', b)  # 输出：b 的结果为 1
```

```
c = True or 1
print('c 的结果为', c)  # 输出：c 的结果为 True

d = False or 1
print('d 的结果为', d)  # 输出：d 的结果为 1

e = not True
print('e 的结果为', e)  # 输出：e 的结果为 False

f = not False
print('f 的结果为', f)  # 输出：f 的结果为 True
```

5、位运算符

位运算符把数字看作二进制来进行操作，包括 &（按位与）、|（按位或）、^（按位异或）、~（按取反）、<<（左移）、>>（右移）。对于刚入门的小伙伴，知道有这个东西就行了。

6、成员运算符

成员运算符用于判断某个数据是否属于序列中数据的一员（是否在序列中），操作结果是一个布尔值。

运算符	描述	示例
in	判断数据是否在序列中	2 in a
not in	判断数据是否不在序列中	2 not in a

序列包括字符串、列表、元组、字典、集合。

示例：

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

a = 2 in data
print('判断 2 是否在列表中，结果为', a)  # 输出：判断 2 是否在列表中，结果为 True

b = 5 in data
print('判断 5 是否在列表中，结果为', b)  # 输出：判断 5 是否在列表中，结果为 False

c = 2 not in data
print('判断 2 是否不在列表中，结果为', c)  # 输出：判断 2 是否不在列表中，结果为 False

d = 5 not in data
print('判断 5 是否不在列表中，结果为', d)  # 输出：判断 5 是否不在列表中，结果为 True
```

7、身份运算符

身份运算符用于判断两个数据的内存地址是否一样，操作结果是一个布尔值。

运算符	描述	示例

is a is b	判断两个数据的内存地址是否一样
is not a is not b	判断两个数据的内存地址是否不一样

身份运算符也用于比较数据，但比较数据时一般用 ==、!= 比较多，两者的区别在于比较的内容不同
对于刚入门的小伙伴，只要会用 ==、!= 比较数据就足够了，这个了解即可。

七、条件语句

条件语句用于控制根据不同的条件，执行不同操作的情况。

1、if 语句

语法格式：

```
if 条件1:
    执行操作1
elif 条件2:
    执行操作2
else:
    执行操作3
```

语句含义：如果满足条件1，则执行操作1；否则如果满足条件2，则执行操作2；否则，执行操作3。

- if 语句中的条件，必须是一个布尔值，或者执行结果为布尔值的表达式或函数。如果为 True，则执行相应操作，为 False 则不执行。
- if 语句中，if 子句必须要有，elif 子句可以 0 到多个，else 子句则可以没有。
- elif 子句只有在前面的条件都不满足，但是当前条件满足时，才会执行相应的操作。
- else 子句是在所有条件都不满足时，才会执行操作。

示例：根据分数判断对应等级。

```
# 存储分数的变量
score = 85
if score >= 85:
    print('优秀')
elif score >= 75:
    print('良好')
elif score >= 60:
    print('及格')
else:
    print('不及格')
```

2、if 语句变种

（1）if...else

语法格式：

```
if 条件:
```

```
    执行操作1
else:
    执行操作2
```

示例：

```
score = 85
if score >= 85:
    print('优秀')
else:
    print('不优秀')
```

（2）if...elif

语法格式：

```
if 条件1:
    执行操作1
elif 条件2:
    执行操作2
```

示例：

```
score = 85
if score >= 85:
    print('优秀')
elif score >= 75:
    print('良好')
elif score >= 60:
    print('及格')
elif score < 60:
    print('不及格')
```

（3）if...

语法格式：

```
if 条件:
    执行操作
```

示例：

```
score = 85
if score >= 85:
    print('优秀')
```

八、循环语句

循环语句用于控制当满足条件时，重复执行某种操作的情况。

1、while 循环语句

语法格式：

```
while 条件:
    执行操作
```

语句含义：当满足条件时，重复执行操作。

示例：计算 1 到 10 所有整数之和。

```
# 存储总数的变量
sum = 0
# 存储当前整数的变量
n = 1
# 当当前整数小于等于 10 时
while n <= 10:
    # 将当前整数加到总数里面
    sum = sum + n
    # 将当前整数值加 1（即下一整数值）
    n = n + 1
print(sum)
```

2、for 循环语句

语法格式：

```
for 变量 in 可迭代对象:
    执行操作
```

语句含义：当可迭代对象中存在未迭代数据（未获取过的数据）时，不断地从里面获取数据，并把数赋值给变量，然后执行操作，直到所有数据都获取过一遍。

- 可迭代对象可以理解为一个数据包，里面包含了许多数据，我们可以不断地从里面获取数据。
- 基本数据类型中，字符串、列表、元组、集合、字典都是可迭代对象。

示例：计算 1 到 10 所有整数之和。

```
# 存储总数的变量
sum = 0
# 存储所有整数的列表
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# 逐个获取列表里的整数，并赋值给变量 n
for n in nums:
    # 将变量 n 的值（即当前整数）加到总数里面
    sum = sum + n
print(sum)
```

3、break 与 continue 语句

（1）break 语句

break 语句用于提前结束循环，此时不会执行剩余的循环。

示例：打印 1 到 10 所有整数，如果碰到 7，则提前结束打印。

```
n = 1
```

```
while n <= 10:
    print(n)
    # 当 n 等于 7 时，结束循环
    if n == 7:
        break
    n = n + 1
```

（2）continue 语句

continue 语句用于跳过当前该次循环，直接执行下次循环。

示例：打印 1 到 10 所有整数，如果碰到 7，则不打印。

```
n = 0
while n < 10:
    n = n + 1
    # 当 n 等于 7 时，跳过该次循环
    if n == 7:
        continue
    print(n)
```

九、函数

前面我们已经知道，函数是对一系列实现特定功能的代码片段的封装，相当于给一系列代码片段取名。

1、定义函数

语法格式：

```
def 函数名(参数1, 参数2, ..., 参数n):
    执行操作
    return 返回值
```

- 参数用于将数据传入函数中，给函数内要执行的操作使用。
- return 返回值 用于返回一个执行结果，它可以放在函数内任何可能需要返回执行结果的地方。
- 如果函数不需要返回一个执行结果，那么可以不用 return 返回值。

示例：

```
# 定义不带返回值的函数
def max1(a, b):
    if a >= b:
        print('最大值：%s' % a)
    else:
        print('最大值：%s' % b)

# 定义带返回值的函数
def max2(a, b):
    if a >= b:
        return a
    else:
        return b
```

2、调用函数

调用函数时，只需要使用函数名，并传入需要的参数就行了。传入的参数可以是值、变量、表达式、数等，只要结果是一个数据就行。对于有返回值的函数，我们可以使用一个变量去接收返回值（即将回值赋值给变量）。

语法格式：

```
函数名(参数1, 参数2, ..., 参数n)
```

示例：

```
# 调用不带返回值的函数
max1(1, 2)

# 调用带返回值的函数，并使用变量 r 接收返回值
r = max2(1, 2)

# 调用带返回值的函数，但不接收返回值
max2(1, 2)
```

十、模块

在 Python 中，一个 Python 文件（.py 文件）就称为一个模块。模块可以被其他程序引入，以使用块中的函数等功能。

Python 中自带了很多模块（称为内置模块），这些模块可以帮助我们实现特定的功能。我们在编写 Python 程序时，只需要导入相应的模块，即可使用模块中的功能。也就是说，有些代码我们完全可以用自己写，直接导入相应的模块白嫖就好了，省时又省力（白嫖党理直气壮）。

1、导入模块

导入模块的操作，一般放在文件顶部（不考虑注释）。

模块内的内容包含变量、函数、类。我们在编写 Python 程序时，一般有三种导入模块的方式（下面们以函数为例）——

（1）导入整个模块

可理解为把整个 Python 文件都拿过来。导入整个模块时，调用函数要使用 模块名.函数名() 的方式。

语法格式：

```
import 模块名
```

示例：

```
import math

a = math.ceil(3.5)
print(a)
```

（2）导入模块内的指定函数

可理解为只把我们需要的函数拿过来，包含在我们的程序内。调用函数时直接使用 函数名() 即可。

语法格式：

```
from 模块名 import 函数名1, 函数名2, ..., 函数名n
```

示例：

```
from math import ceil, floor

a = ceil(3.5)
print(a)
b = floor(3.5)
print(b)
```

（3）导入模块内的所有函数

相当于无论需不需要，先把所有函数都拿过来再说。调用函数时直接使用 函数名() 即可。

语法格式：

```
from 模块名 import *
```

一般不建议使用该方式。因为使用 import * 时，其实会将模块内的所有内容都导过来，除了会导入多不需要的内容外，还有可能会与自己的代码发生命名冲突。

示例：

```
from math import *

a = ceil(3.5)
print(a)
```

2、安装第三方模块

如果 Python 自带的模块满足不了我们的需求，也可以使用别人提供的模块（称为第三方模块）。使别人提供的模块时，需要我们另外安装——直接打开命令行模式，输入 pip install 模块名 回车，即开始下载并安装第三方模块。

示例：安装 Pillow，Pillow 是一个图像处理工具库。

```
Microsoft Windows [版本 10.0.19042.1110]
(c) Microsoft Corporation。保留所有权利。

C:\Users\10100>pip install Pillow
Collecting Pillow
  Downloading Pillow-8.3.1-1-cp39-cp39-win_amd64.whl (3.2 MB)
     |████████████████████████████████| 3.2 MB 6.8 MB/s
Installing collected packages: Pillow
Successfully installed Pillow-8.3.1
```

十一、异常处理

程序在执行时，有可能会出现各种异常，一旦出现了异常，程序将不会再往下执行。

但是有些异常其实无关紧要，即使发生了也不应该影响程序的正常执行。这时候我们就可以主动将异捕获，并进行异常处理，使其不影响程序的执行。除此之外，我们也可以根据具体情况，主动抛出异，来中断程序的执行。

1、捕获异常

语法格式：

```
try:
    执行业务操作
except 异常1 as 异常变量1:
    执行异常处理操作1
except 异常2 as 异常变量2:
    执行异常处理操作2
finally:
    执行必须操作
```

- try 子句用于执行我们自己的业务操作。
- except 子句用于在 try 子句发生指定异常时，执行对应的异常处理操作。可以有 0 至多个。
- finally 子句用于执行一些必须要执行的操作，无论 try 子句有没有发生异常都不会影响 finally 子的执行。
- except 子句与 finally 子句不是必须的，但是至少得有一个。
- 如果不知道 except 子句应该捕获什么异常，可以直接捕获 Exception 异常，即 except Exceptio as 异常变量。

示例：

```
try:
    print('try 子句开始执行')
    r = 10 / 0
except Exception as e:
    print('except 子句捕获到异常：', e)
finally:
    print('finally 子句开始执行')
```

2、抛出异常

语法格式：

```
raise 异常
```

最简单的写法是直接抛出一个 Exception 异常，即 raise Exception('异常提示信息') 。

示例：

```
a = -1
if a < 1:
    raise Exception('a 的值不能小于 1')
```

十二、文件读写

操作文件前，我们要先使用 open() 函数打开文件，然后才可以对文件进行读写操作。由于打开文件占用计算机资源，因此，读写操作完成后，要使用 close() 函数关闭文件。

1、操作文件写法

（1）操作文件一般写法

```
# 打开文件
f = open('文件路径', '打开模式', encoding='编码格式')
# 操作文件
...
# 操作文件完成后，关闭文件
f.close()
```

- 文件路径用于指定要操作的文件。
- 打开模式用于指定打开文件后，后续要以什么模式来操作文件。常用模式有 r、rb、w、wb 等。
  - r：只读模式，用于从文件中读取内容，文件必须事先存在。
  - rb：二进制只读模式，用于从二进制文件中读取内容，文件必须事先存在。一般用于图片、视等非文本文件。
  - w：只写模式，用于往文件中写入内容，文件不存在时会自动创建。
  - wb：二进制只写模式，用于往二进制文件中写入内容，文件不存在时会自动创建。一般用于图、视频等非文本文件。
- 编码格式用于指定以哪种编码字符集来打开文本文件。文本文件保存时会选择一种编码字符集来保内容，我们在打开文本文件时也要使用相同的编码字符集，否则可能会导致内容错乱。一般使用 utf-8 编码字符集。
- 操作二进制文件时，不用指定编码格式，只有文本文件需要。

（2）操作文件增强写法

如果操作文件过程中发生了异常，会导致 close() 函数不被执行。因此为了保证 close() 函数被执行可以使用 try ... finally 语句。

```
try:
    # 打开文件
    f = open('文件路径', '打开模式', encoding='编码格式')
    # 操作文件
    ...
finally:
    if f:
        # 操作文件完成后，关闭文件
        f.close()
```

（3）操作文件优雅写法（推荐）

使用 try ... finally 的写法每次都要调用 close() 函数太麻烦了，Python 提供了 with 语句来帮我们自调用 close() 函数。

```
with open('文件路径', '打开模式', encoding='编码格式') as f:
    # 操作文件
    ...
```

2、读文件

打开文件后，可以使用 read()函数来从文件中的读取内容。

示例：读取 D 盘 mypython 目录下的 test.txt 文件内容（事先要建立好文件）。

```
try:
    # 以只写模式打开文本文件
    f = open('D:\\mypython\\test.txt', 'r', encoding='utf-8')
    # 读取内容
    print(f.read())
finally:
    if f:
        # 操作文件完成后，关闭文件
        f.close()
```

3、写文件

打开文件后，可以使用 write()函数往文件中写入内容。

示例：

```
try:
    # 以只写模式打开文本文件，指定编码格式为 utf-8
    f = open('D:\\mypython\\test.txt', 'w', encoding='utf-8')
    # 写入内容
    f.write('今天你学 Python 了吗？')
finally:
    if f:
        # 操作文件完成后，关闭文件
        f.close()
```

十三、面向对象

面向对象是一种程序设计思想，它把程序中涉及到的所有东西都看作一种对象——万物皆对象。对象含义是指具体的某一事物，即我们现实生活中看得见摸得着的事物。

现实生活中，所有事物都具有属性和行为，比如猫具有品种、颜色、重量等属性，以及吃饭、睡觉、萌等行为。因此，每个事物都可以用属性和行为来描述。

Python 中的对象也具有属性和行为（一般称为方法），属性通过变量来体现，而行为则通过函数来现。也就是说，对象中包含了数据（存储在变量中）和操作数据的函数。也可以把对象理解为带有函的数据。实际上，Python 中的所有数据都是对象。

对于面向对象，我们这里只讲如何创建一个简单的对象，以及如何操作对象中的数据。如果有小伙伴算深入学习，那么建议一定要掌握面向对象的三大特性：封装、继承、多态。

1、如何创建对象

对象基于类创建，类是对象的模板。通过类这个模板，我们可以创建出各式各样属于同一个类，但是性和方法又有所不同的对象。

可以把类理解为事物的种类，同一种类下的事物虽然同属一个种类，但是可能在某些属性或行为上又所不同。比如所有的猫都属于猫这一种类，但是每只猫的品种、毛色、重量都会有所不同。

（1）定义类

语法格式：

```
class 类名:

    def __init__(self, 属性1, 属性2, ...):
        self.属性1 = 属性1
        self.属性2 = 属性2
        ...

    def 方法1(self, 参数1, 参数2, ...):
        执行操作

    def 方法2(self, 参数1, 参数2, ...):
        执行操作

    ...
```

- 类中方法的第一个参数必须是 self，它代表对象本身。
- 我们可以通过方法的 self 参数，来操作对象数据或者调用对象内部的其他方法。

示例：定义一个猫的类

```
class Cat:

    def __init__(self, breed, color, weight):
        # 品种
        self.breed = breed
        # 毛色
        self.color = color
        # 重量（斤）
        self.weight = weight

    # 吃饭
    def eat(self):
        print('这只%s开始吃鱼了，都已经%s斤了还吃...' % (self.breed, self.weight))
        # 每次吃完鱼后，重量增加1斤（太能吃了...）
        self.weight = self.weight + 1

    # 睡觉
    def sleep(self):
        print('这只%s开始睡觉了...' % self.breed)

    # 卖萌
    def act_cute(self, toy):
        print('你给了%s一个%s，它开始卖萌了...' % (self.breed, toy))
```

（2）通过类创建对象

语法格式：

```
对象变量 = 类名(属性1, 属性2, ...)
```

示例：创建一只猫的对象

```
cat = Cat('布偶猫', '白色', 10)
```

2、如何操作对象数据

我们可以通过对象的属性和方法来操作对象的数据。

（1）通过属性操作

创建完对象后，可以通过 对象.属性名 的方式来操作对象的数据。

示例：

```
cat = Cat('布偶猫', '白色', 10)
# 获取重量数据
w = cat.weight
# 更新重量数据
cat.weight = 15
# 新增昵称数据
cat.nickname = '小白'
```

（2）通过方法操作

创建完对象后，可以通过 对象.方法名() 的方式来操作对象的数据。其实本质上是在方法内部通过对的属性操作数据。

示例：

```
cat = Cat('布偶猫', '白色', 10)
# 通过吃饭方法，更新重量数据
cat.eat()  # eat() 方法的实现见“示例：定义一个猫的类”
```

结语

终于肝完了~

本来打算尽量用最简洁的语言来写清楚的，但是又怕自己表述不清楚，结果不知不觉越写越多，难搞。。果然，我的语文都还给体育老师了！之前买的《写作这回事》估计也等我很久了吧，是时候拿起看一看了~

交流区

```
<p align="center">
   <img src="https://b3logfile.com/file/2020/04/qrcode-8e461b12.jpg" height="250px" widt
="250px"><br/>
   微信公众号：惊却一目<br/>
   个人博客：<a href="https://www.jingqueyimu.com">惊却一目</a>
</p>
```