Python编程语言

基础篇

零基础入门系列教程

作者: 魏明择

2025 年版

https://weimingze.com

目录

第一章、python开发环境搭建

前言

为什么学习python

1. 简单，容易入门:
   - 最适合作为人生的第一编程语言；学会Python后再学习其他难懂的编程语言（比如C/C++）更容易上手，驾轻就熟，实现弯道超车。
2. 用途广泛:
   - Python常用于数据分析和数据科学、人工智能、机器学习、算法开发、网络爬虫、web网站后端开发、金融以及教育教学等领域，上手快，开发效率高，出产品快。
3. 容易扩展
   - 支持面向对象编程，更容易修改和更新，以最小的代价任意扩充功能。
4. 模块多
   - 支持百万级的模块，做任何事都不用从头开始（直接从1开始做起，跳过0～1的过程）
5. 可以和C/C++混合编程
   - 可以更好的进行底层调用，保密性封装。兼顾执行效率和易用性。

本课程目标人群

- 学生
- 数据分析师
- 人工智能算法工程师
- 国家计算机二级考试考生
- 中小学python教师
- python后端开发人员
- 高校准毕业生
- 计算机培训机构讲师
- 少儿编程讲师

教程内容

基础篇	高级篇
基础数据类型-整数、小数等	异常
运算符和表达式	闭包
容器类型--列表，字典等	生成器和迭代器
各种语句	递归
函数和作用域	内建函数重写
模块和包	运算符重载
类和面向对象	特性属性
文件	汉字编码
常用内建函数等	

本课程特点

- 简单易懂
  - 专业知识白话文解读

- 权威
  - 以python官方文档作为教材，避免错误解读。

- 内容全面精细 -- 解读python语言部分全部知识点
  1. python 语言基础篇
  2. python 语言高级篇

- 效率高
  - 不废话，干货满满！

1. Python简介

- 创始人- Guido van Rossum（荷兰人）

- 时间: 1989年圣诞节期间

- 地 点: 阿姆斯特丹
- 其他流行编程语言

编程语言	时间	作者	应用场景
C语言	1972	Dennis Ritchie	系统、驱动、嵌入式开发
C++	1979	Bjarne Stroustrup	大型软件开发（游戏后台、GUI等）
java	1995	James Gosling	Web后端、安卓开发
Go(Golang)	2009	Google公司	Web后端服务器

python 的优缺点

1. 优点
    - 简单易学易用
    - 应用领域广泛
    - 免费
    - 可移植（支持 Windows, Mac OS X, Linux操作系统)
    - 开源
    - 面向对象(Java, C++, Python)
    - 可混合编程(C/C++)

2. 缺点
   - 与C/C++相比，执行速度不够快
   - 不能封闭源代码

python 的资源

- python 的官方网站:
  - 网址: https://www.python.org

> 我们可以通过官网得到更多更新的信息及文档。

- python的官方文档：

- 网址: https://docs.python.org/zh-cn/3/

- python 的版本
  - Python V3.13(当前教学)
  - Python V3.15(最新)

- python 下载地址:
  - 网址: https://www.python.org/ftp/python/

建议

因 Python 编程语言学习有先后顺序关系，因此建议 Python 的爱好者们从前向后阅读此站点文章，由浅入深，直达**资深**！

> 成功属于通过勤奋来掌握自己命运的人！加油！！！

声明：

1. 本站点讲解的内容均为 Python3 的内容，忽略其他版本。
2. 本站点内容和视频为原创，转发请不要随意篡改，更不要改错！
3. 本站点为原创手写，有错误文字请联系作者：魏明择，谢谢。

2. Mac安装python环境

如果你已经安装了 Python 解释执行器请略过此文章进入下一节。

本教程安装环境

```
操作系统: Mac OS 10.15(或以上)
CPU: Intel i5 处理器
内存: 8G(或以上)
```

> 其他系统下载对应版本即可

本教程分为以下几个步骤讲解

1. 下载Python安装包
2. 安装Pyhton
3. 验证运行Python

第一步：下载Python安装包

去官方网站下载Python 3.13.1 版本的安装包

此电脑为Mac OS 10.15 因此使用安装包文件: **python-3.13.1-macos11.pkg**

- Mac OS 10.15 操作系统的下载地址: https://www.python.org/ftp/python/3.13.1/python-3.13.1-macos11.pkg

- Mac OS 11或 其他版本下载地址: https://www.python.org/ftp/python/

第二步：安装Pyhton

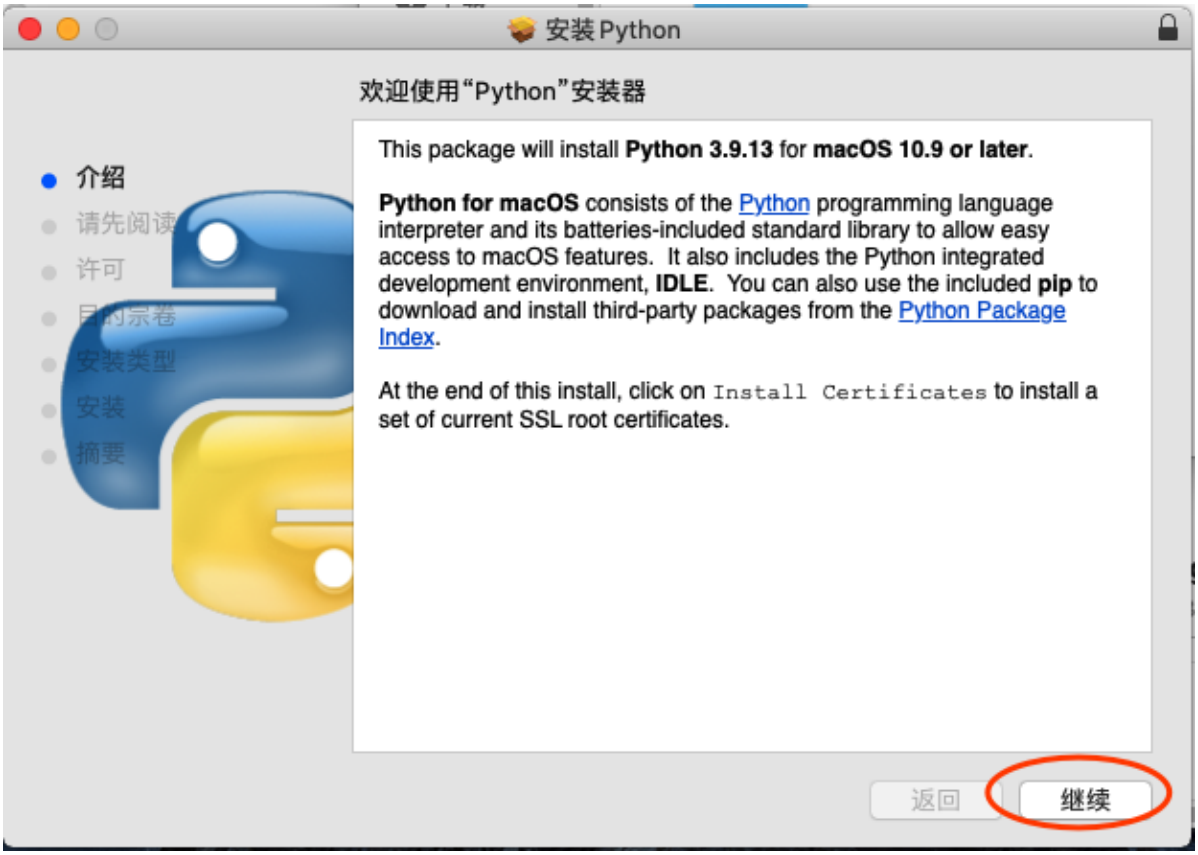
> 以下以**python-3.9.13版本为例进行截图示范**

1. 点击安装包 **python-3.13.1-macos11.pkg**

python-3.9.13-
macosx10.9.pkg

进入安装程序

2. 点击 **继续** 进入安装流程。



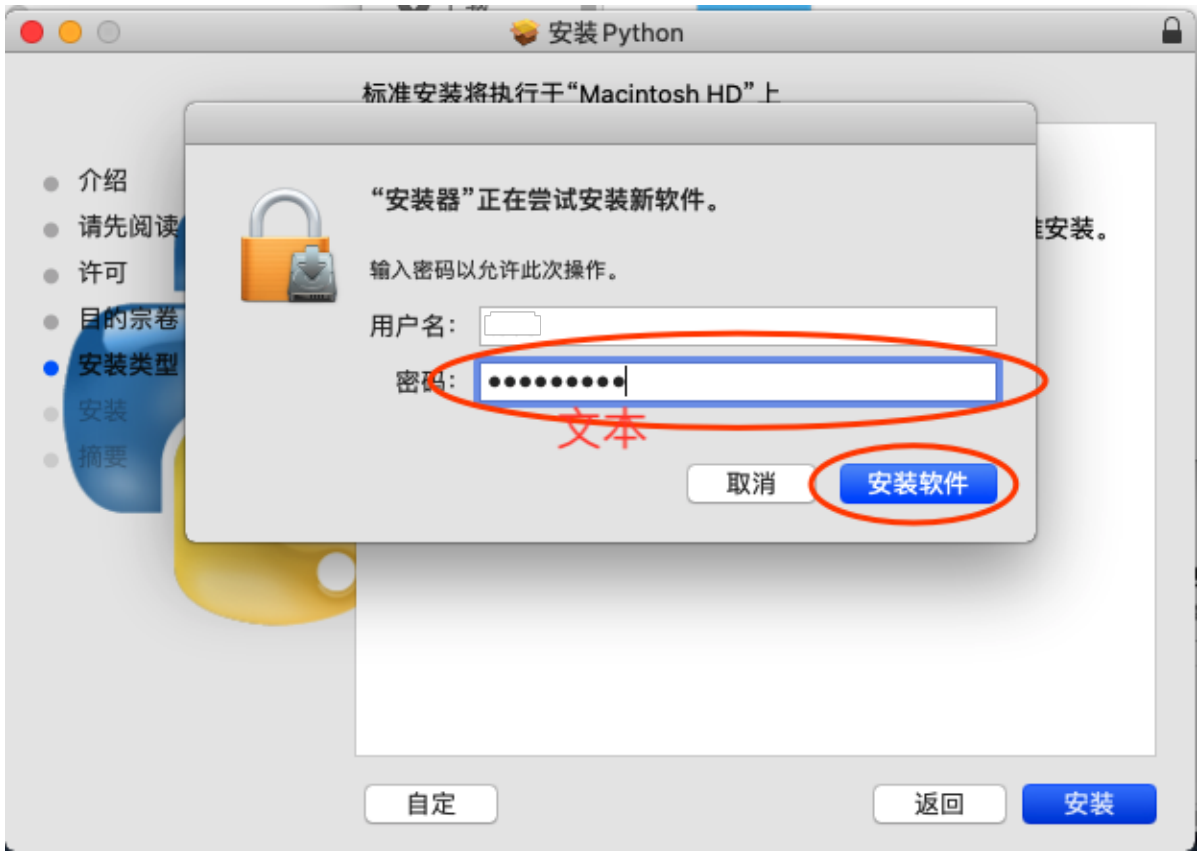
3. 之后一路使用默认设置，点击 **继续**。

4. 查看**软件许可协议**，点击 **继续** 进入

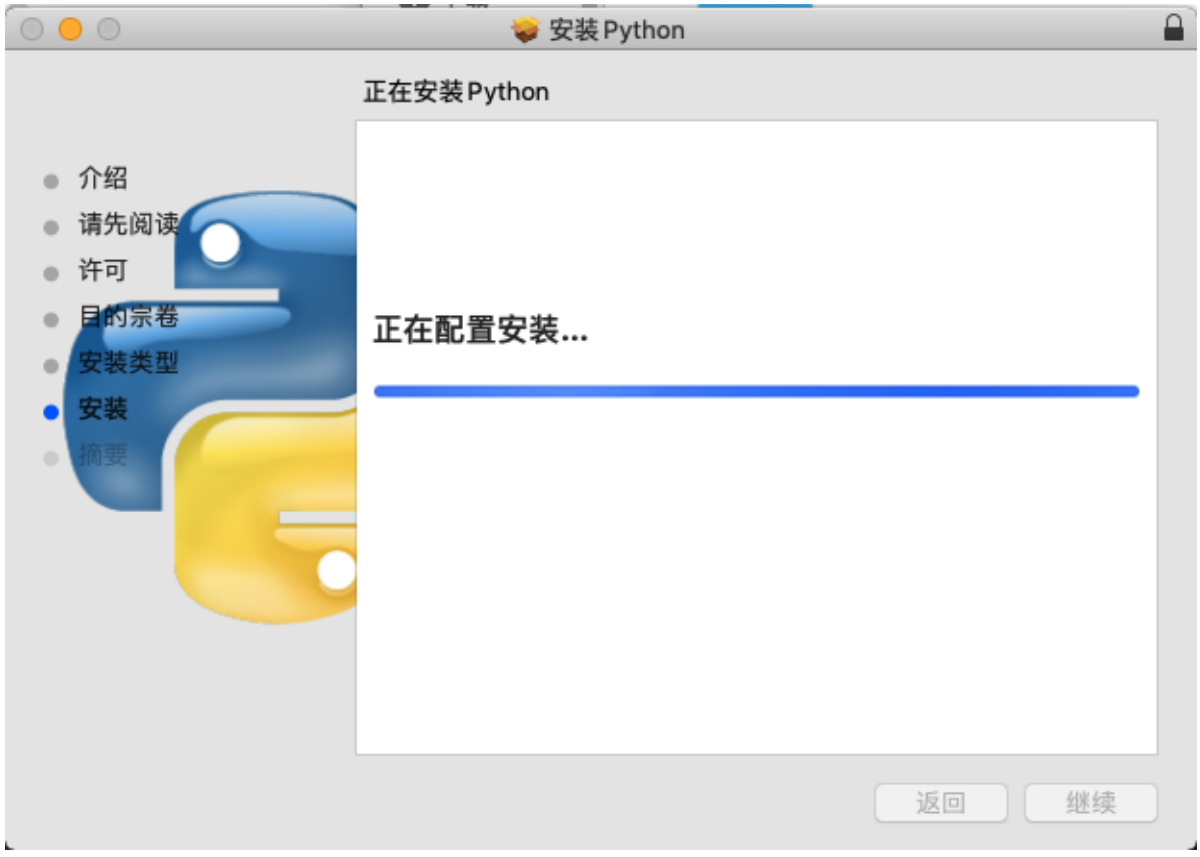
5. 点击 **同意** 进入安装

6. 点击 **安装**，进入安装过程....

7. 输入密码，点击 **安装软件** 确认安装



8. 正在进行安装



9. 安装完成

至此，python 3.13.1 安装成功

第三步：验证运行Python

- 点击 **启动器**  在 **其他** 中找到 **终端**，如下图所示

- 在 **终端** 内输入 **python3** 进入 python的交互模式，说明python已经安装成功了。如下图所示！

- 在三个大于号提示符下输入 **exit()** 即可退出Python程序，回到 **命令提示符** 程序中

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

至此，python的解释执行器安装完毕

3. Win10安装python环境

如果你已经安装了 Python 解释执行器请略过此文章进入下一节。

本教程安装环境

```
操作系统: Windows 10(或以上)
CPU: Intel i5 处理器
内存: 8G(或以上)
```

本教程分为以下几个步骤讲解

1. 下载Python安装包
2. 安装Pyhton
3. 验证运行Python
4. 卸载Python

第一步：下载Python安装包

去官方网站下载Python 3.13.1 版本的安装包

安装包文件:`python-3.13.1-amd64.exe`

- 64位操作系统的下载地址:
  https://www.python.org/ftp/python/3.13.1/python-3.13.1-amd64.exe

- 32位操作系统的下载地址(适用于Win7 32位系统):
  https://www.python.org/ftp/python/3.13.1/python-3.13.1.exe

> 其他版本的Python解释执行器的下载地址:
>
> > https://www.python.org/ftp/python/

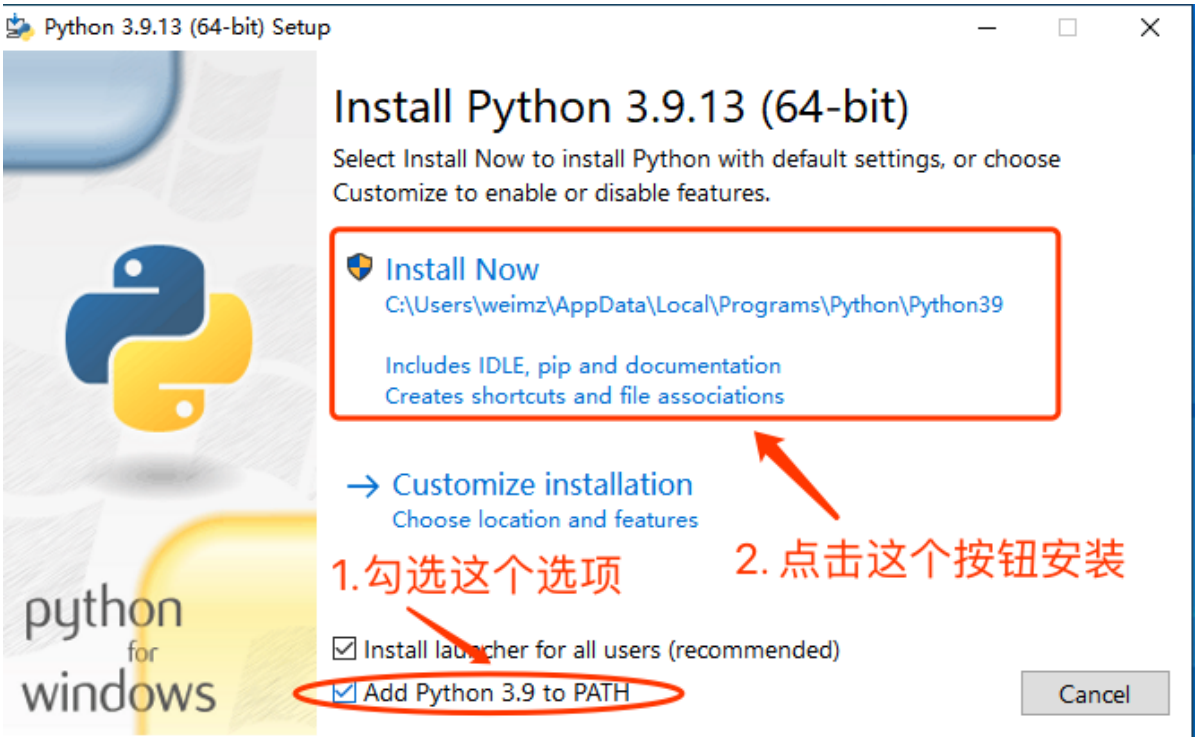
第二步： 安装Python

> 以下以**python-3.9.13版本为例进行截图示范**

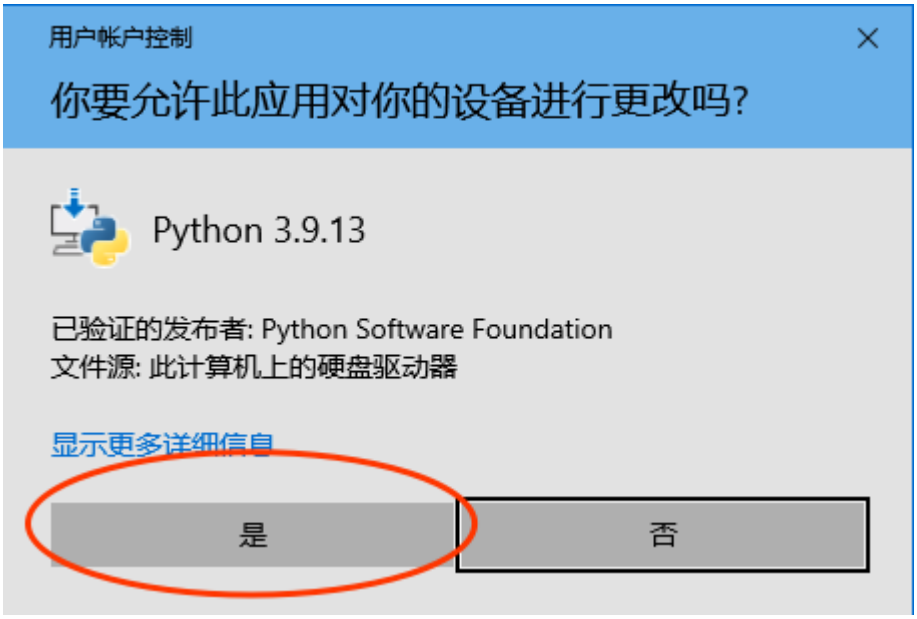
1. 双击文件 `python-3.13.1-amd64.exe` ，运行安装程序。

   如图:

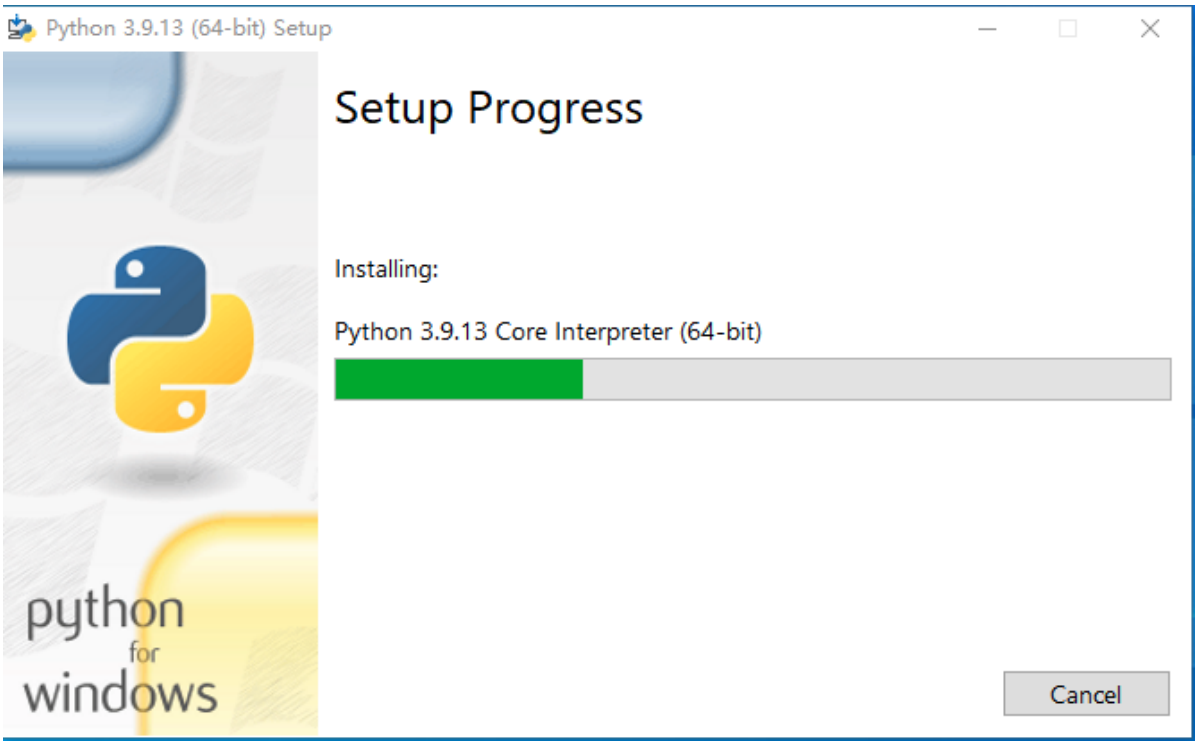
2. 勾选 **"Add Python 3.13.1 to PATH"** 选项，然后点击 **"Install Now"** 进行安装。



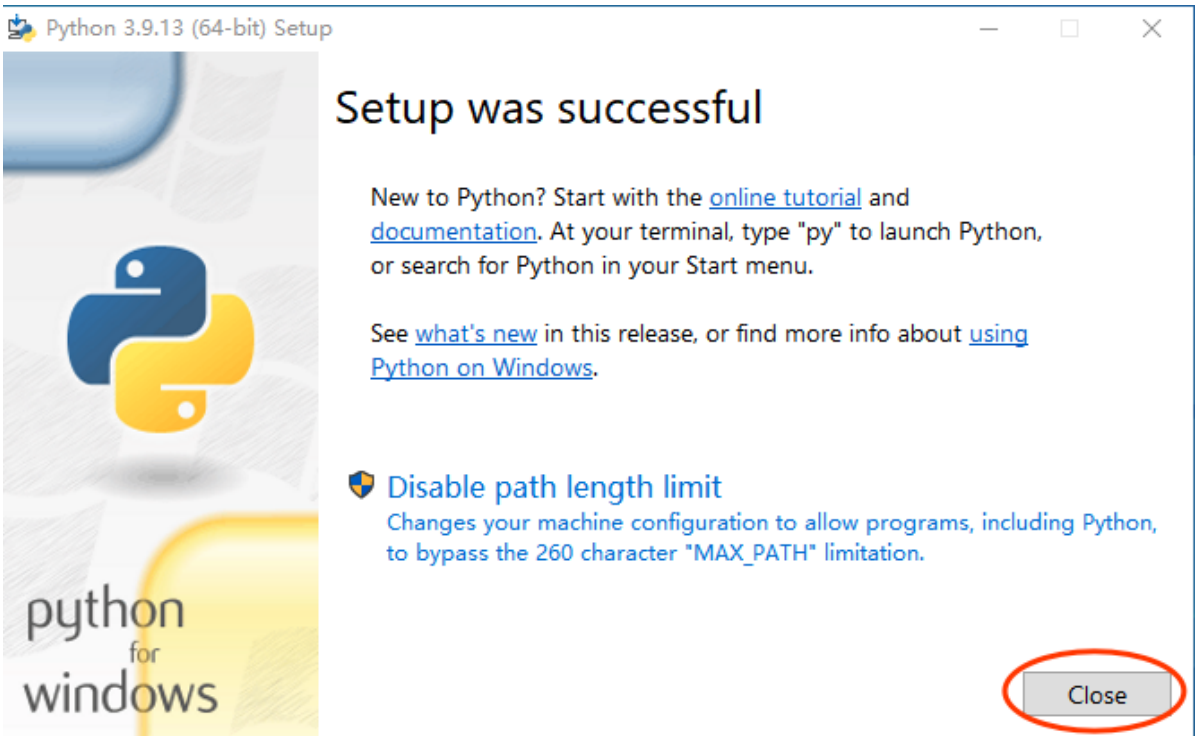
3. 点击 **是** 允许次应用对你的设备进行更改。



4. 进入安装过程中...

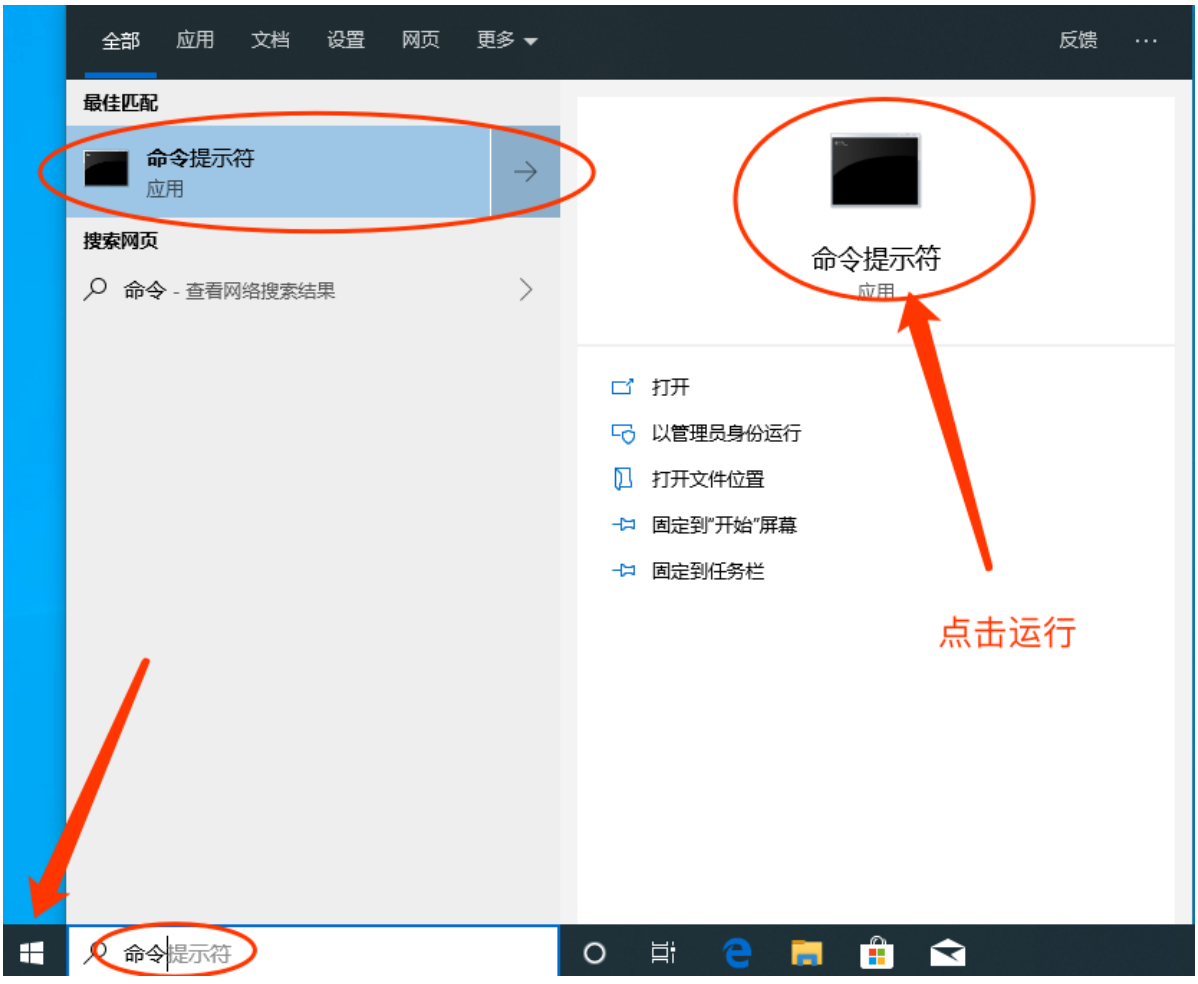


5. 安装成功显示如下界面。点击 **Close** 关闭即可!

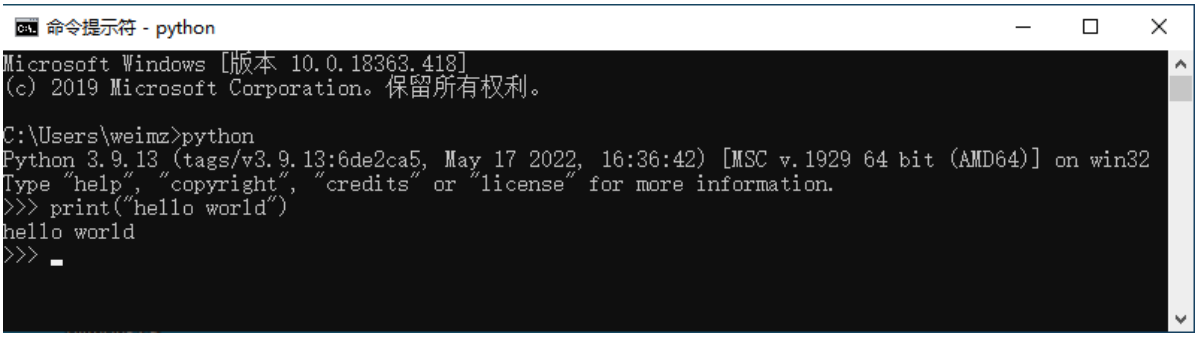


第三步：验证运行Python

- 运行 **命令行提示符** 程序



- 输入命令 **python** 后 按下回车键，即可进入Python的交互模式，如下图所示：



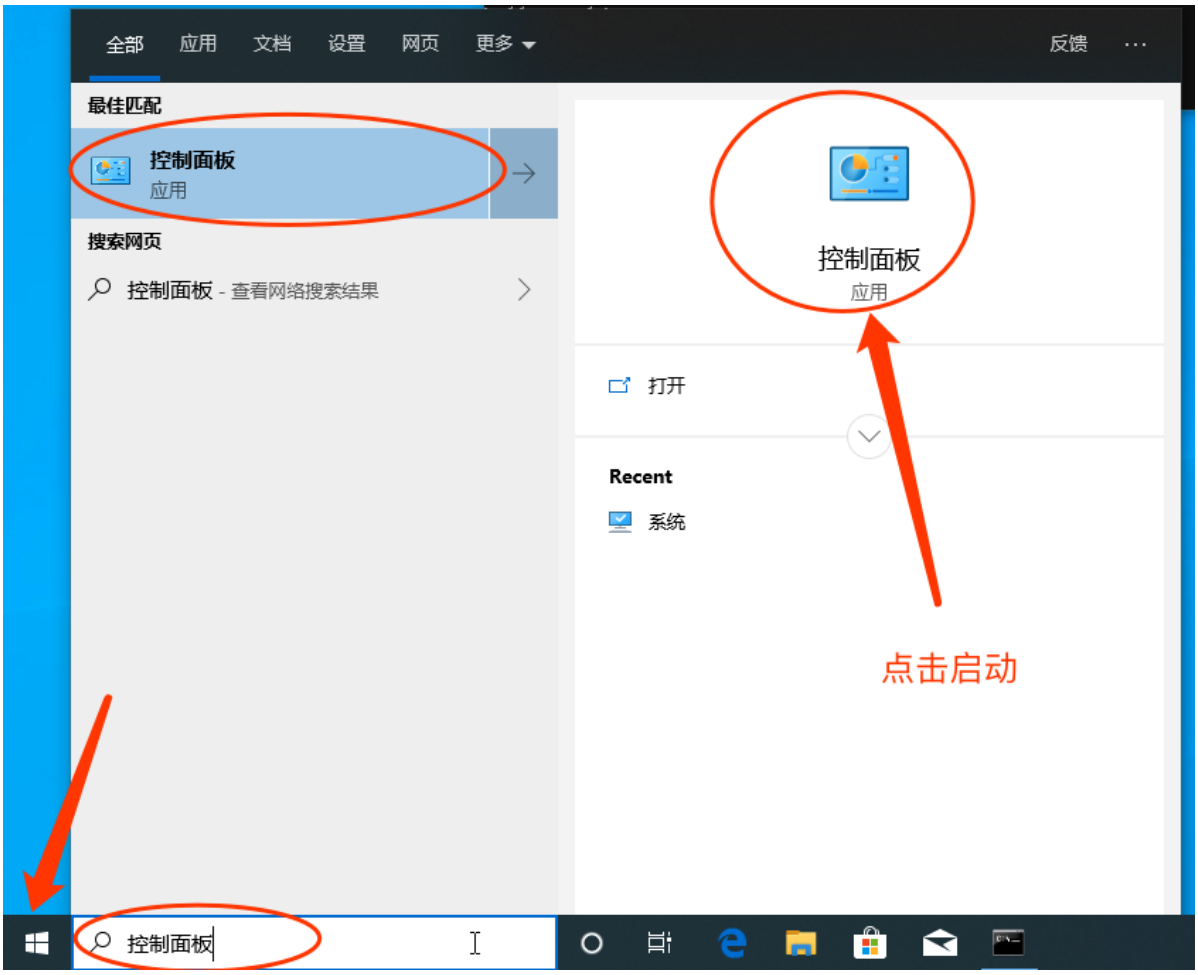
- 在三个大于号提示符下输入 **exit()** 即可退出Python程序，回到 **命令提示符** 程序中

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

第四步：卸载Python

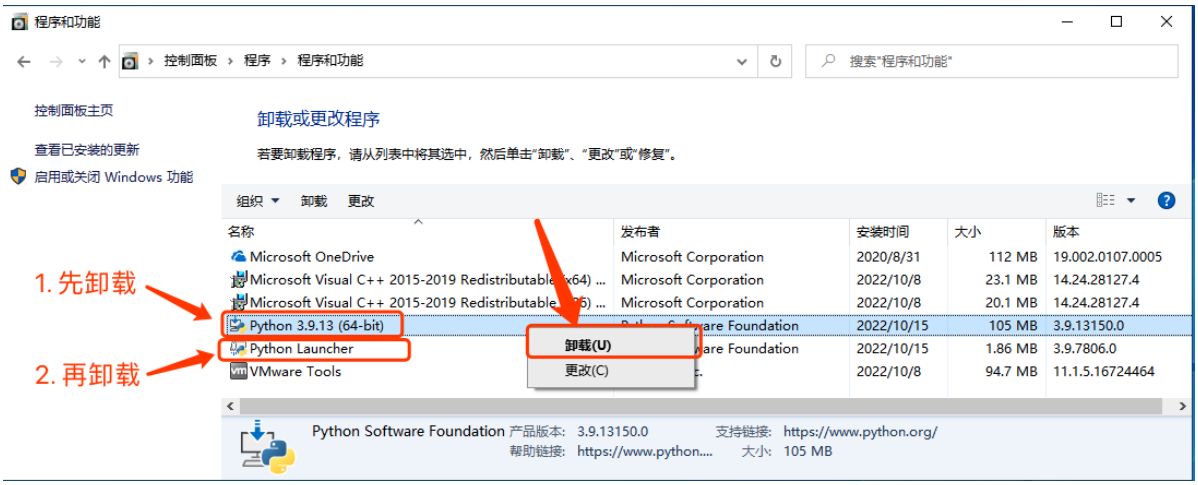
> 如果你安装有多个版本的 python, 在运行时可能会出现冲突，建议你卸载多余版本的python，只保留一个版本的python。

- 启动Windows **控制面板** 如下图所示

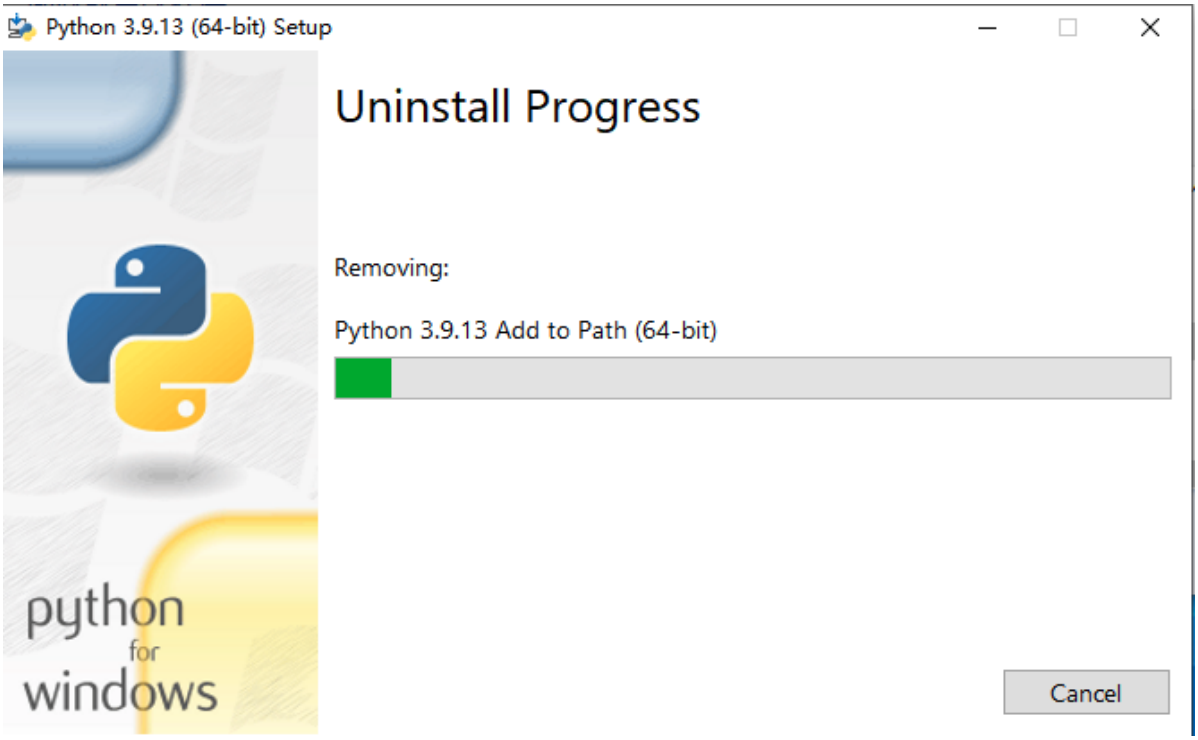


- 在控制面板中找到 **卸载程序** 并点击运行

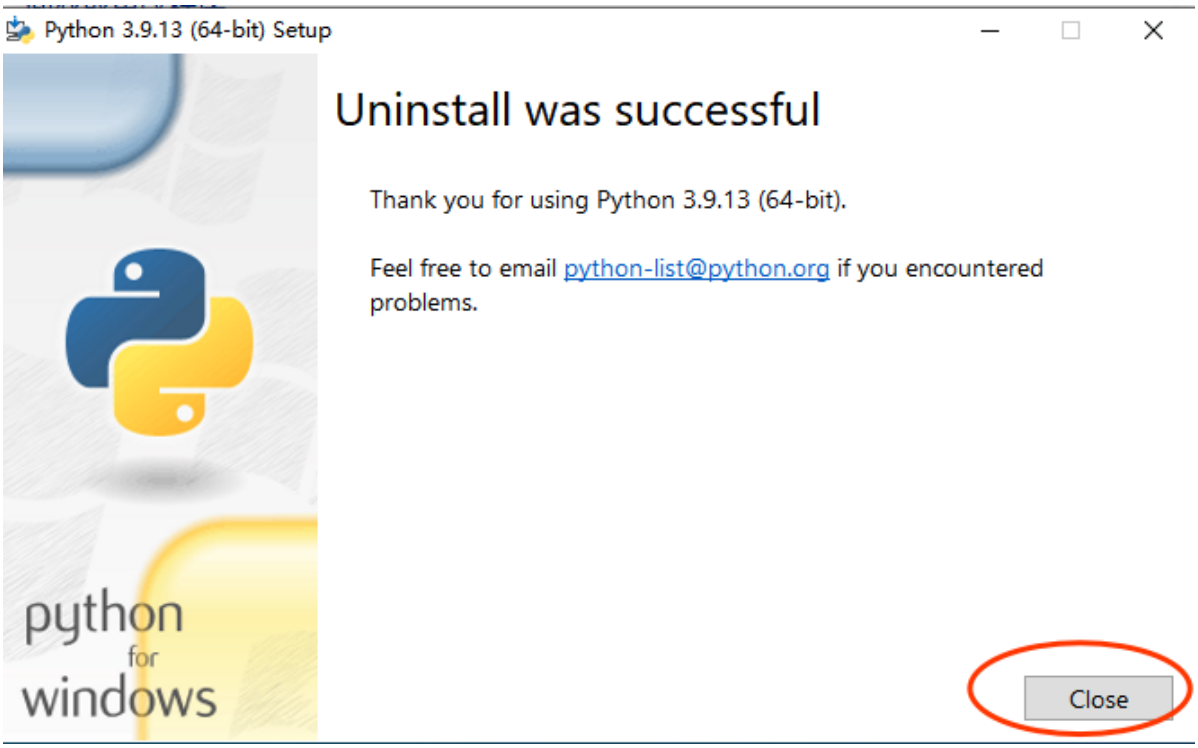
- 进入 **程序和功能** 界面，先要卸载 **Python 3.13.1(64-bit)** 然后再卸载 **Python Launcher** 两个都需要卸载，如下图所示:



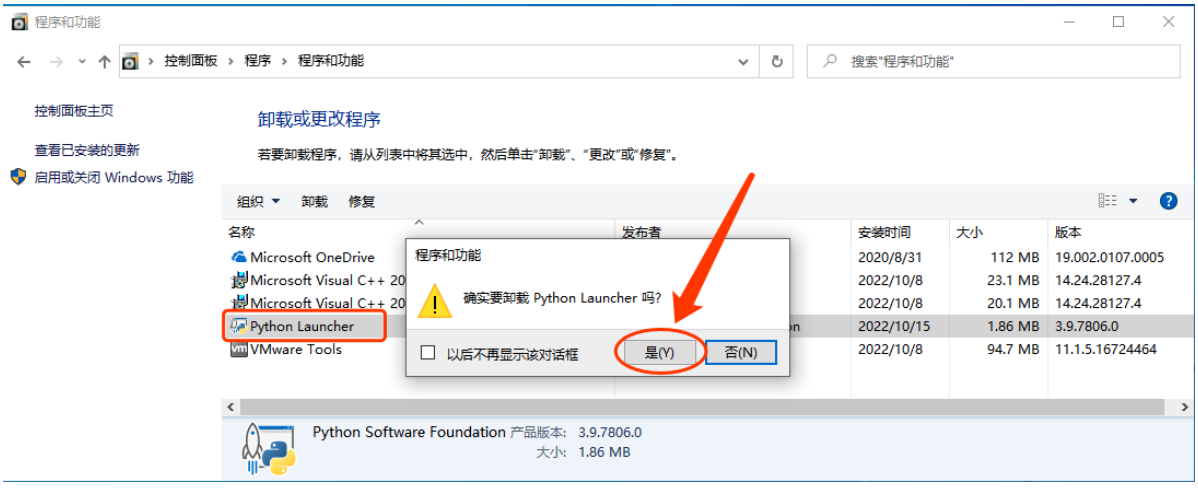
- 进入卸载程序...



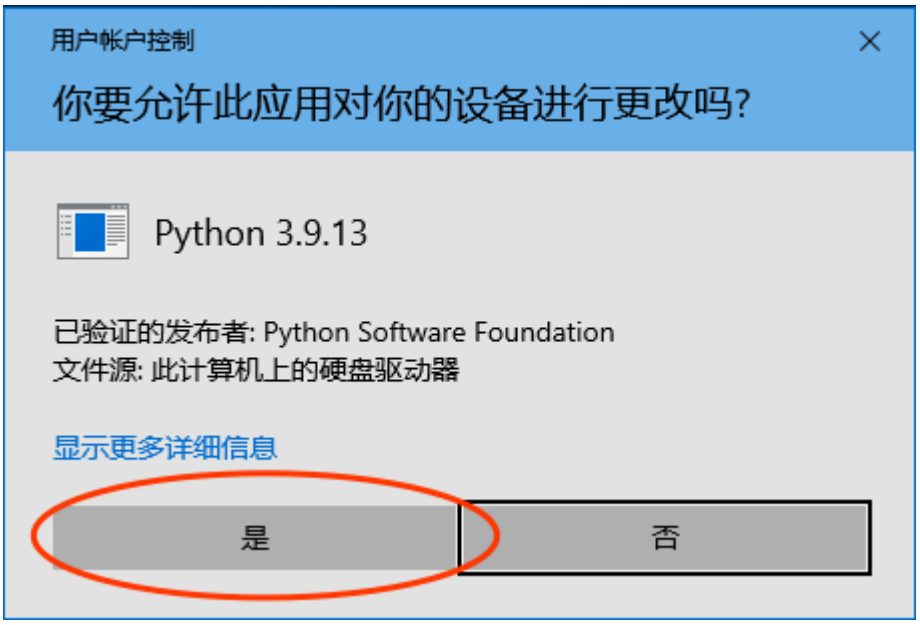
- 卸载 **Python 3.13.1(64-bit)** 成功。



- 接下来卸载 **Python Launcher** 如下图所示



- 点击是，开始卸载



恭喜你卸载成功!

至此，python的解释执行器安装完毕

4. PyCharm安装

python 常用编写代码的工具有 PyCharm、Visual Studio Code、XCode(MacOS)、记事本(Windows)、Notepad++、UltraEdit(Windows)、Vim(Linux/MacOS)、Sublime Text等。

PyCharm 是编写Python代码最好用的工具。PyCharm 提供如下功能：

- 语法高亮
- 语法错误提示
- 输入提示
- 编辑器内执行程序
- 调试功能等

PyCharm 的两个版本

- 社区版（Community）- 免费（推荐）
- 专业版（Professional）- 收费

PyCharm 下载地址：

- 网址:https://www.jetbrains.com/pycharm/download/

4.1 Mac 安装 PyCharm

如果你已经安装了 PyCharm 请略过此文章进入下一节。

在安装PyCharm之前请先安装Python解释执行器

本教程安装环境

```
操作系统: Mac OS 10.15(或以上)
CPU: Intel i5 处理器
内存: 8G(或以上)
```

本教程分为以下几个步骤讲解

1. 下载 PyCharm 社区版 软件
2. 安装 PyCharm 社区版
3. 启动运行 PyCharm 社区版
4. 新建PyCharm 工程(Project)

第一步：下载 PyCharm 社区版 软件

1. PyCharm Mac OS X (苹果mac 系统) 版本 安装包如下:
   - 下载地址：https://download.jetbrains.com.cn/python/pycharm-community-2024.3.dmg

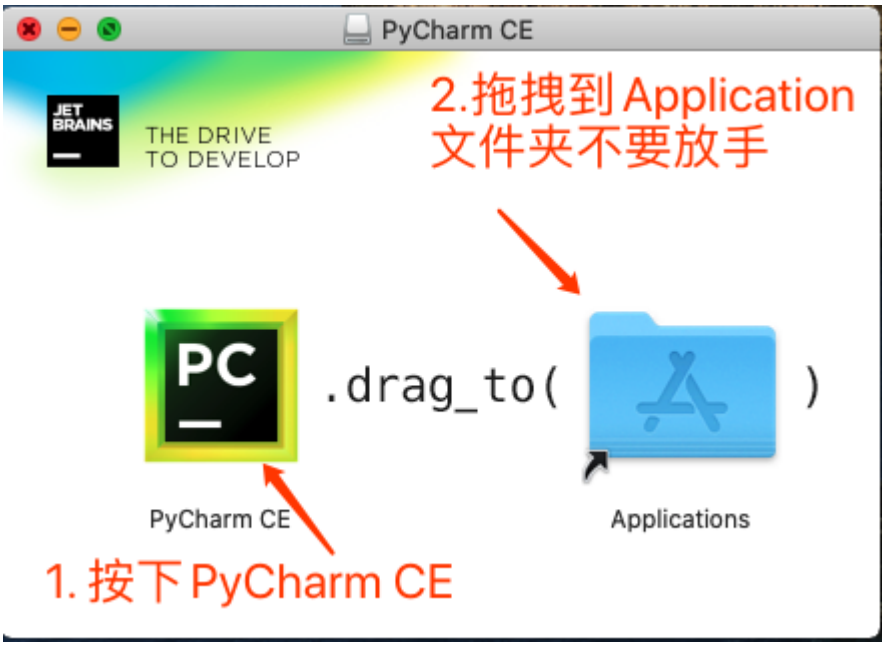
   > 如果你的电脑是 M1芯片的Mac电脑，你可以点击如下链接下载最新的版本。
   > https://www.jetbrains.com/pycharm/download/

第二步：安装PyCharm 社区版

1. 双击图标

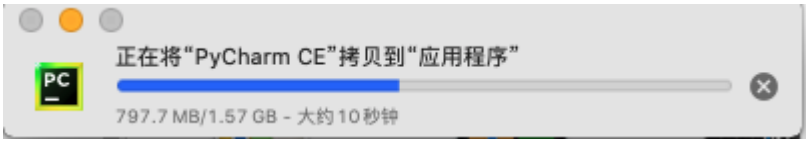
   运行Pycharm安装包文件 `pycharm-community-2024.3.dmg`

2. 将 应用程序 "PyCharm CE" 拖拽到 "Applications" 文件夹



3. 在"应用程序"后放手。

4. 安装程序开始复制中...

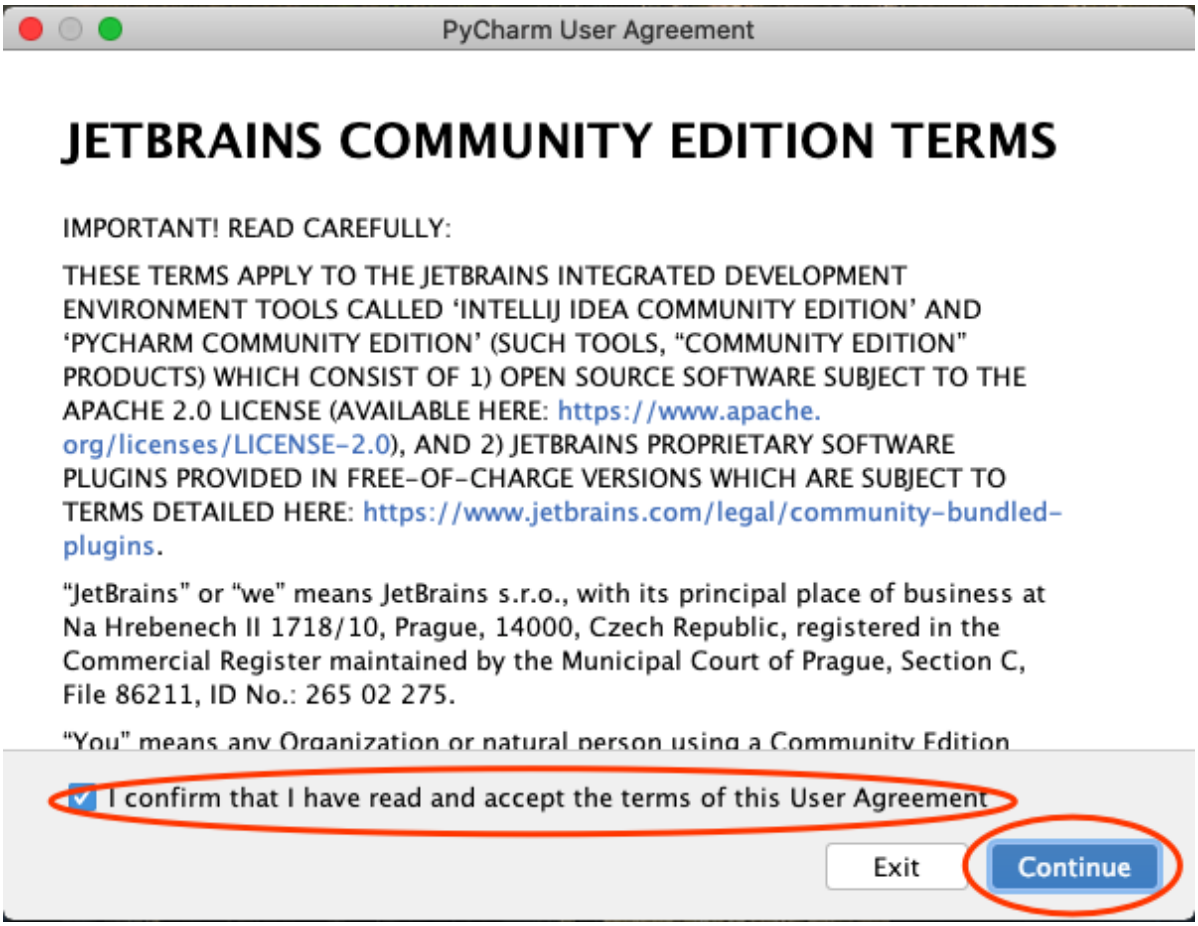


5. 安装成功，点击 **应用程序** 中 **PyCharm CE** 可以启动PyCharm

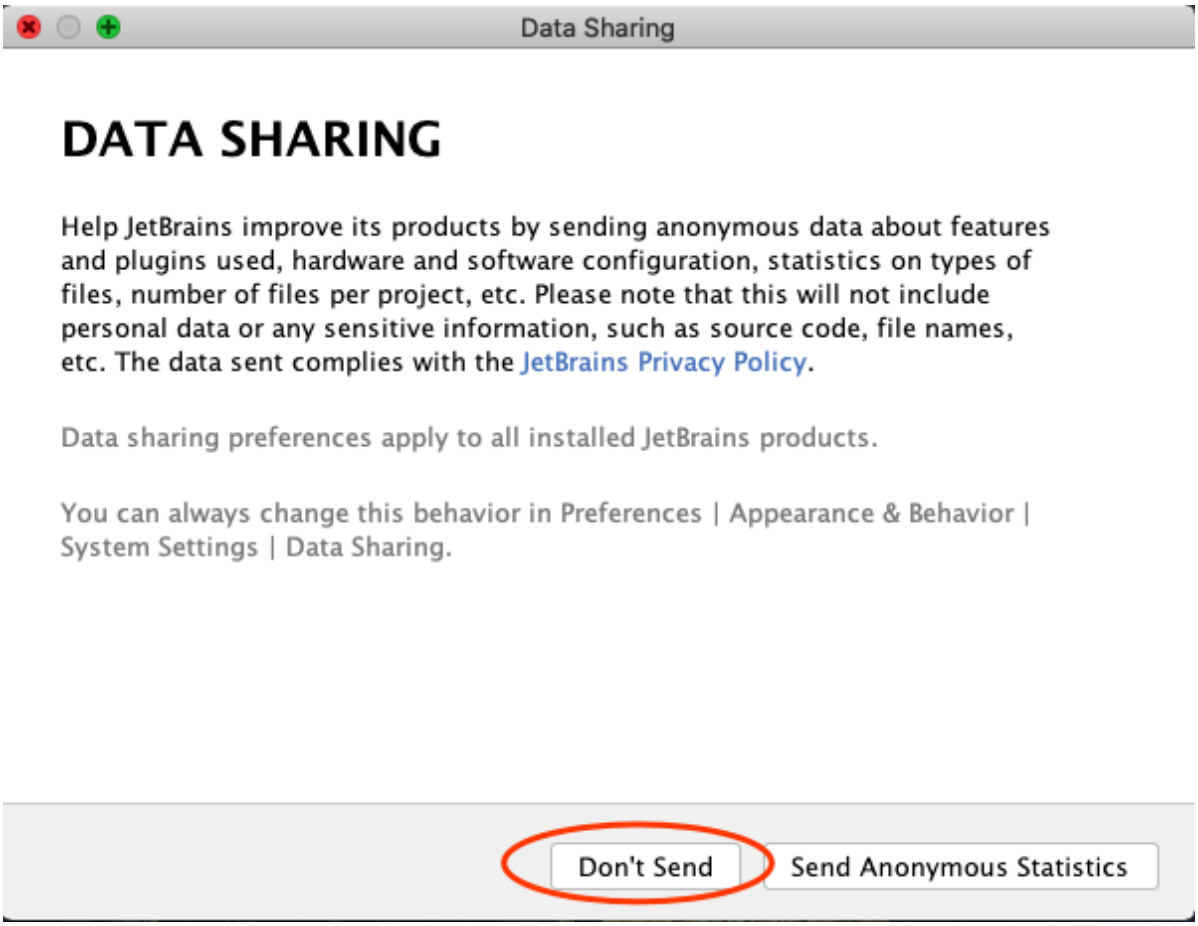
第三步：启动运行 PyCharm 社区版

1. 从 **应用程序** 中找到 **PyCharm CE** 点击启动 PyCharm

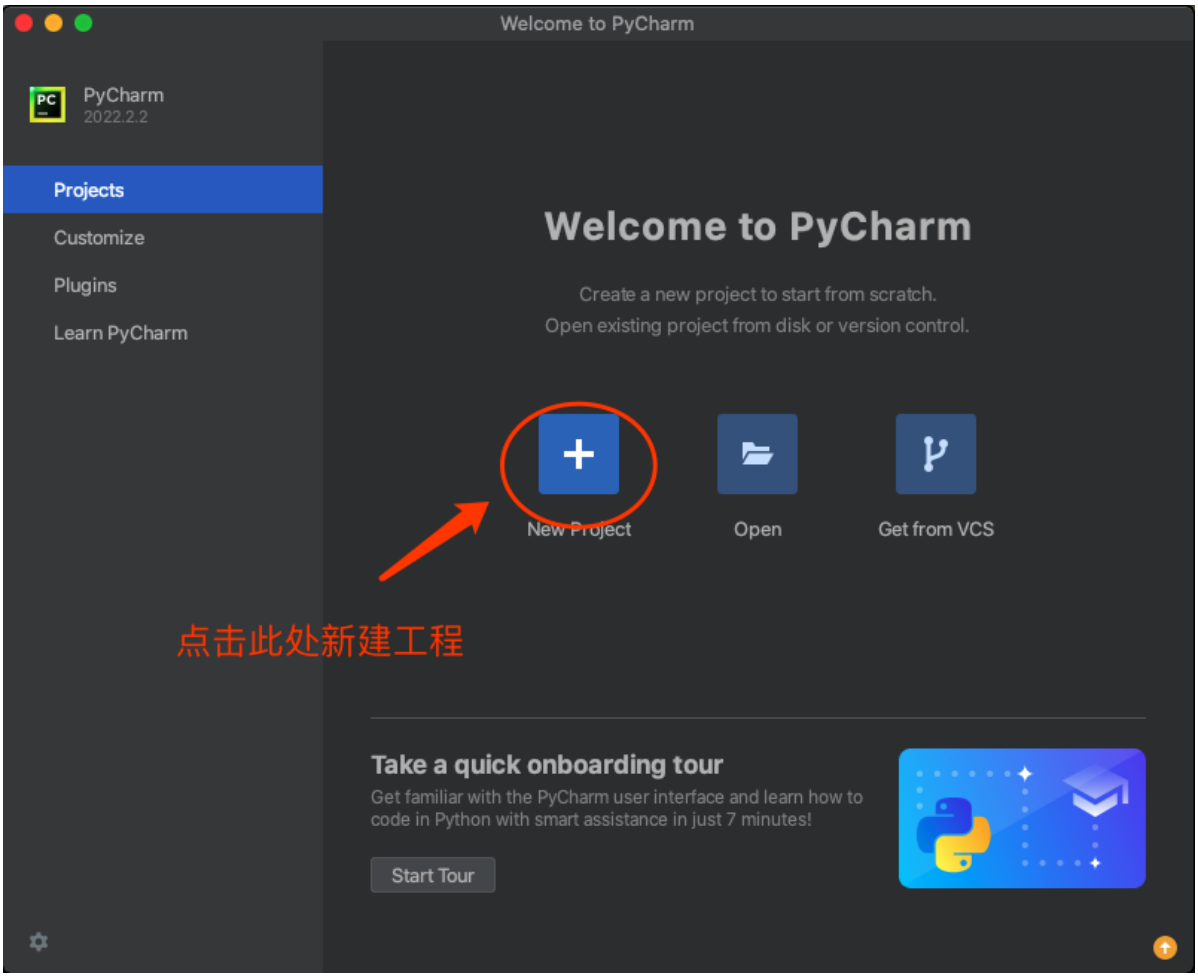
2. 第一次启动需要确认许可协议，勾选同意后点击"Continue" 继续下一步。



3. 数据分型，可以选择 "Don't Send" (不发送)

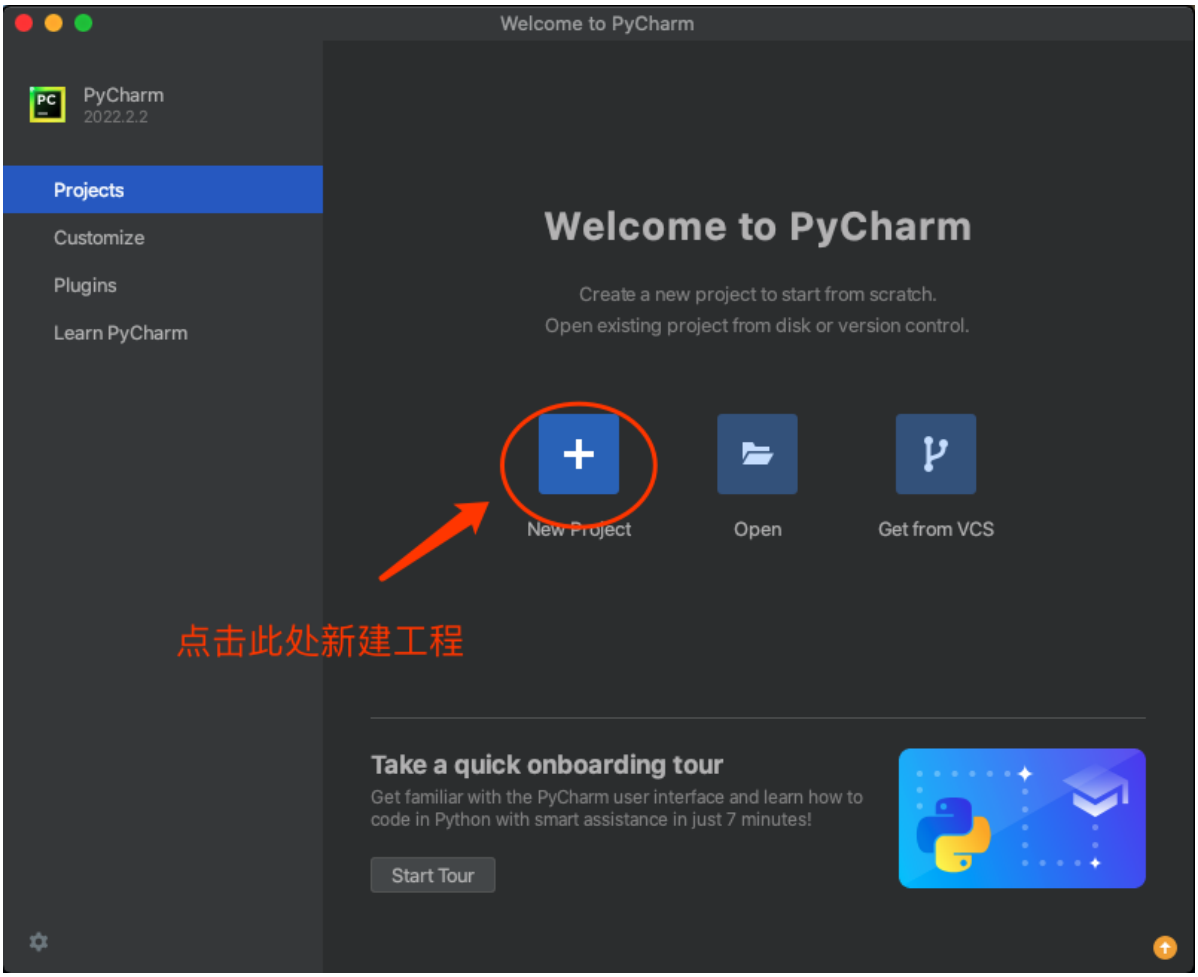


4. 进入新建工程界面

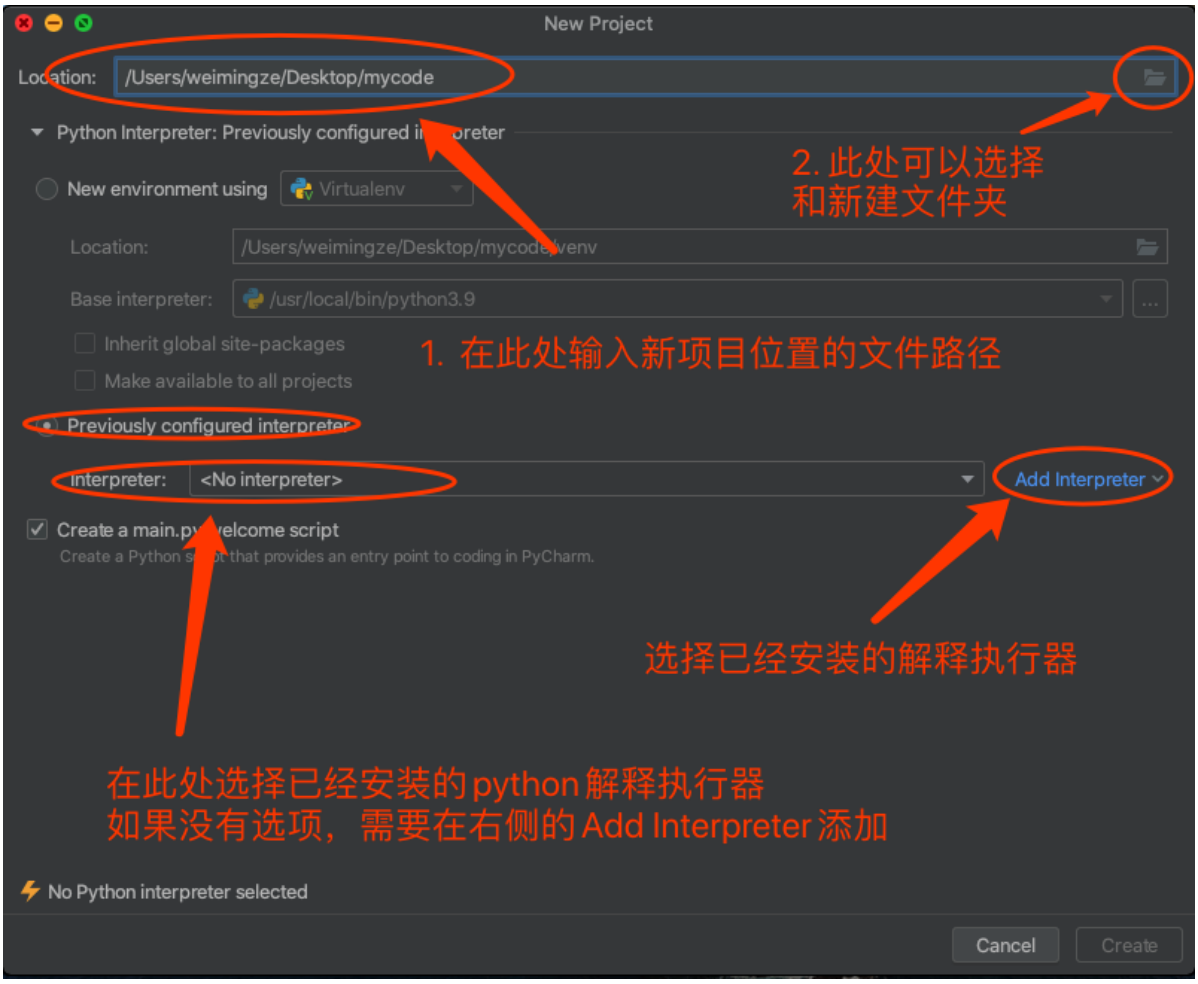


第四步：新建PyCharm 工程(Project)

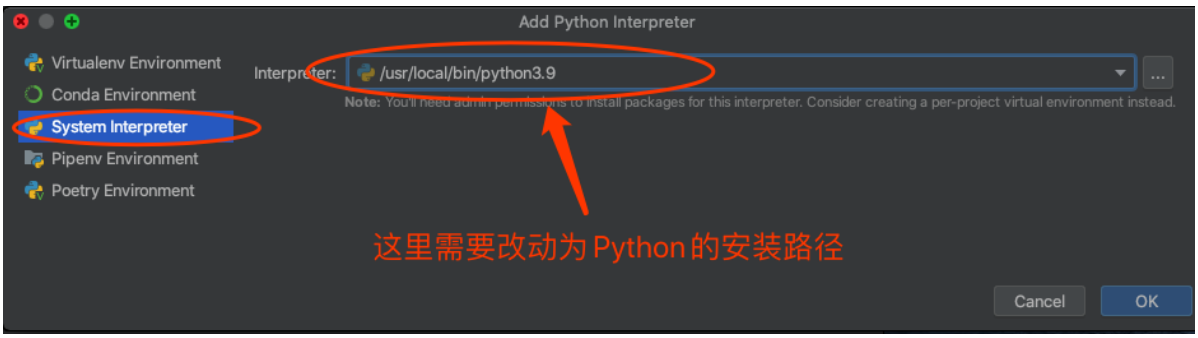
1. 选择Create New Project 新建工程



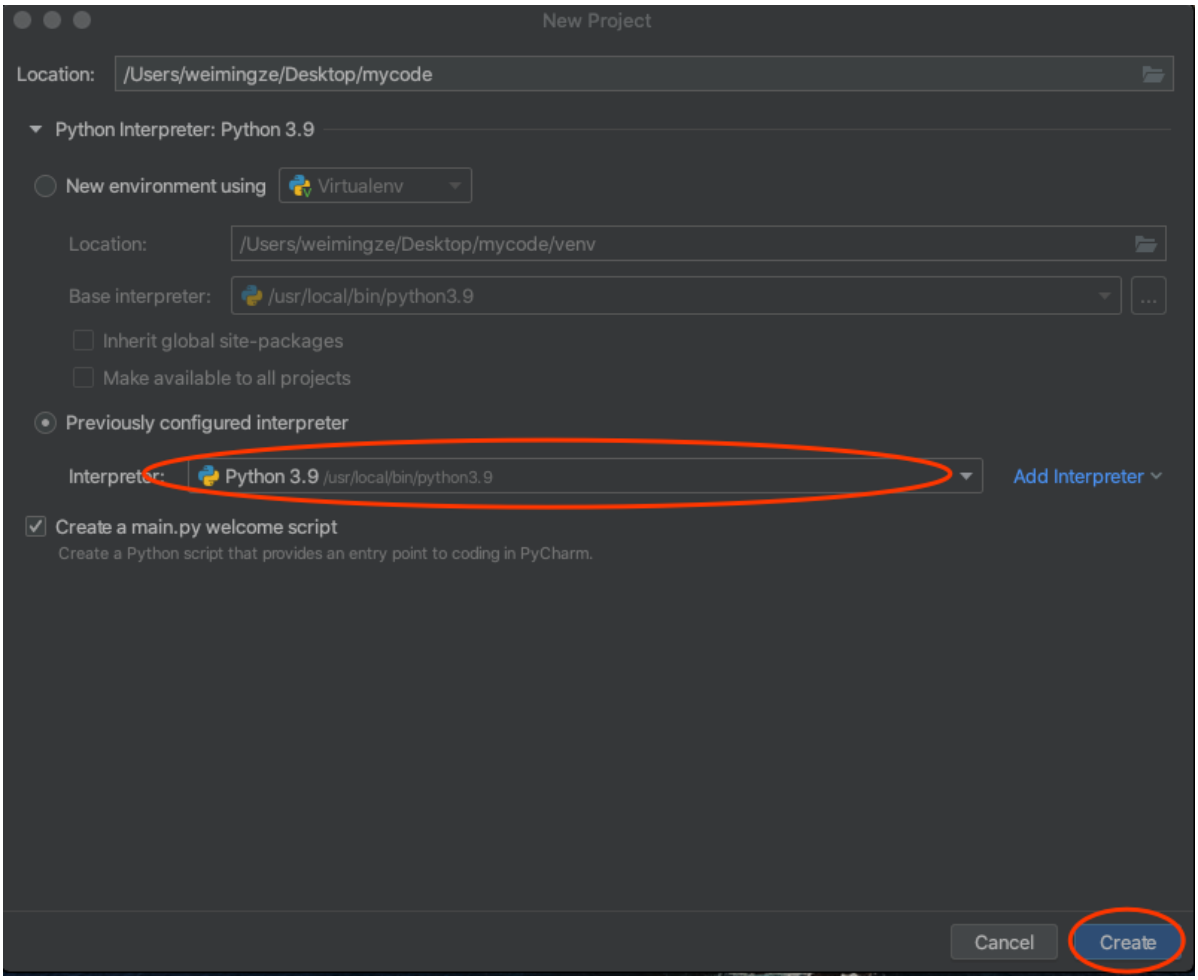
2. 选择项目文件夹的存放位置后。选择解释Python执行器的位置



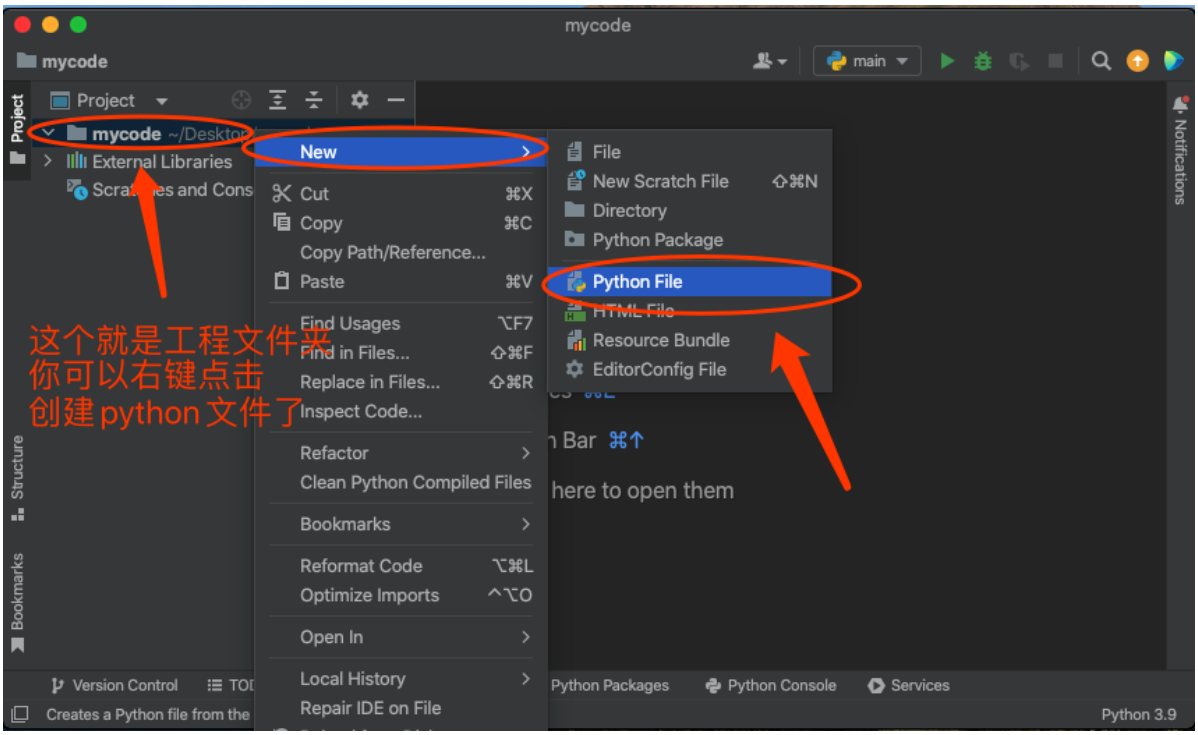
3. 选择 `Add Interpreter` 系统的解释执行器



4. 解释执行器已经选定，点击 Create 进入编辑界面



5. 在以下编辑界面可以新建 Python文件和其他的文件了



4.2 Windows安装PyCharm

如果你已经安装了 PyCharm 请略过此文章进入下一节。

在安装 PyCharm 之前请先安装 Python 解释执行器

本教程安装环境

```
操作系统: Windows 10(或以上)
CPU: Intel i5 处理器
内存: 8G(或以上)
```

本教程分为以下几个步骤讲解

1. 下载 PyCharm 社区版 软件
2. 安装 PyCharm 社区版
3. 启动运行 PyCharm 社区版
4. 新建PyCharm 工程(Project)

第一步：下载 PyCharm 社区版 软件

1. PyCharm windows 版本 安装包如下:

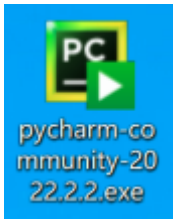
   https://download.jetbrains.com/python/pycharm-community-2024.3.exe

2. PyCharm Mac OS X (苹果mac 系统) 版本 安装包如下:

   https://download.jetbrains.com.cn/python/pycharm-community-2024.3.dmg

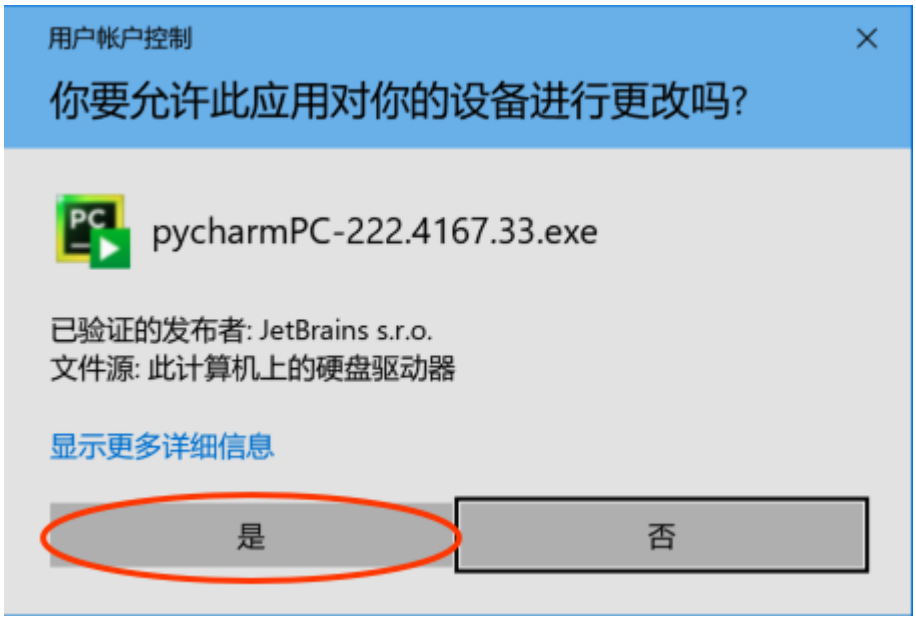
第二步：安装 PyCharm 社区版

1. 双击图标

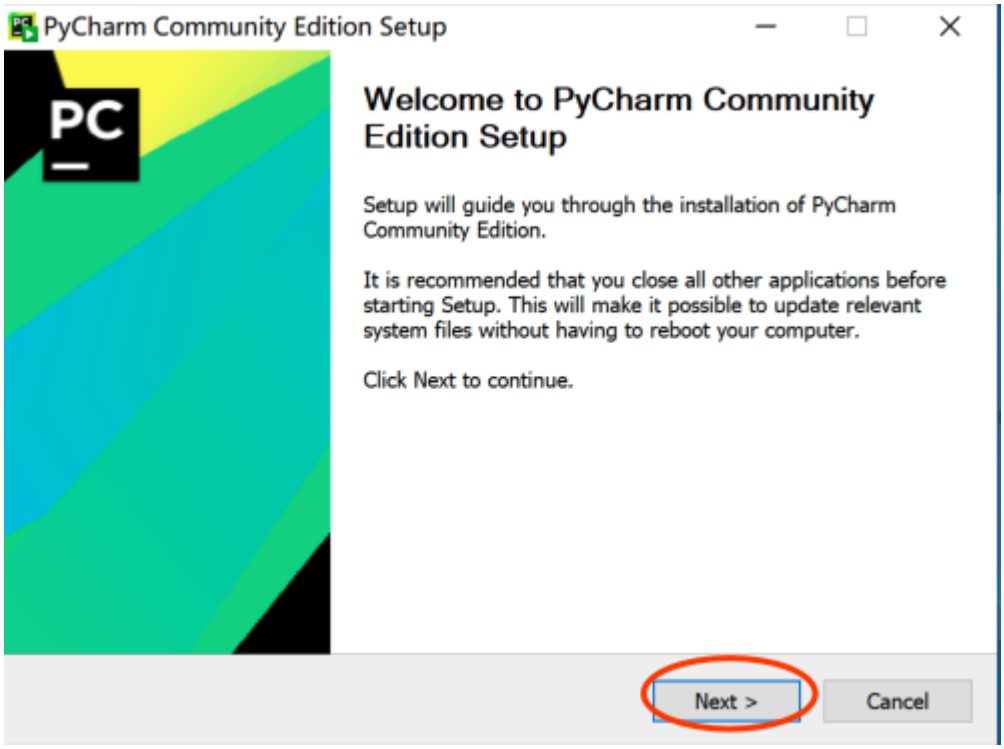
   

   运行Pycharm安装包文件 `pycharm-community-2024.3.exe`

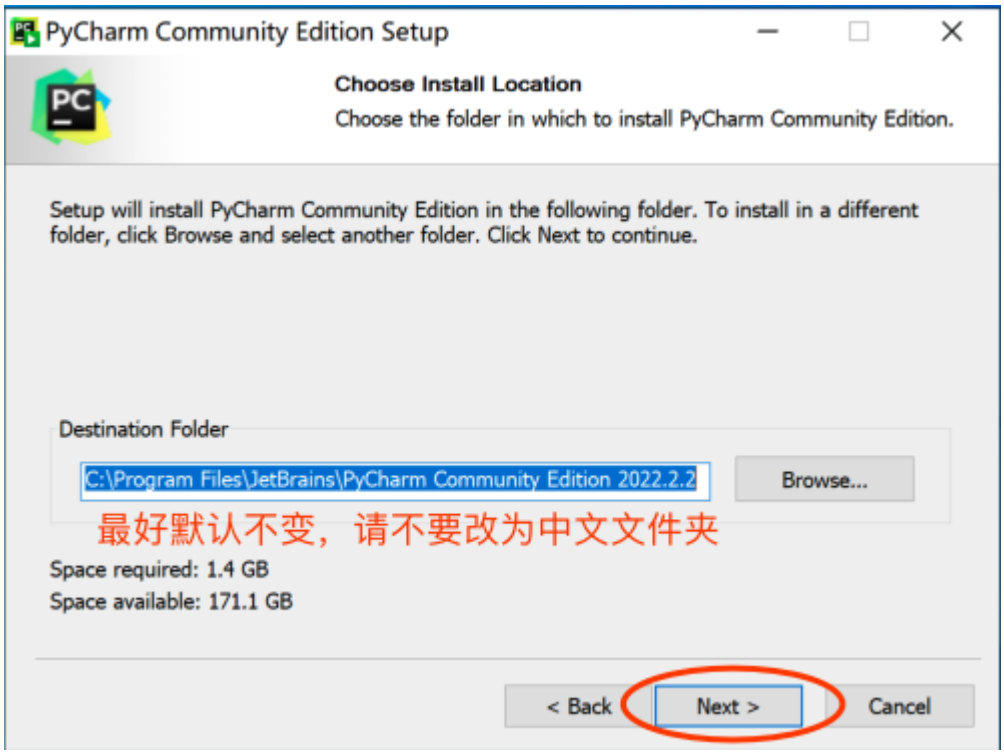
2. 在用户账户控制对话框中选择 "是"



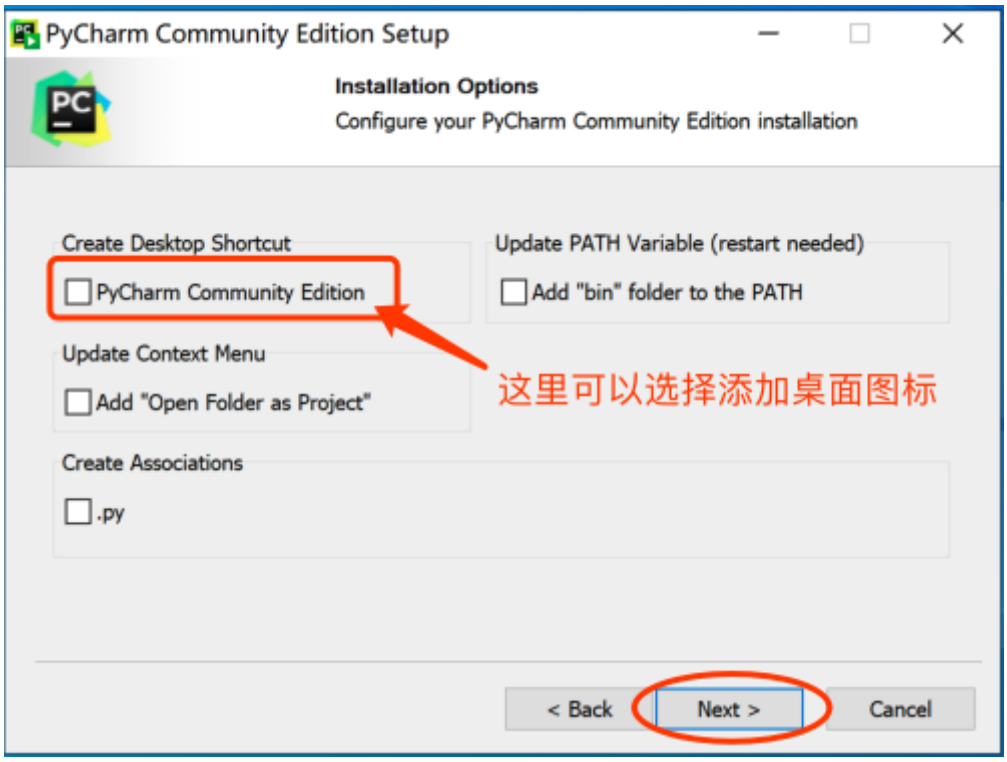
3. 进入安装向导界面， 点击"next" > 以下基本就是一路点击"next" 即可!



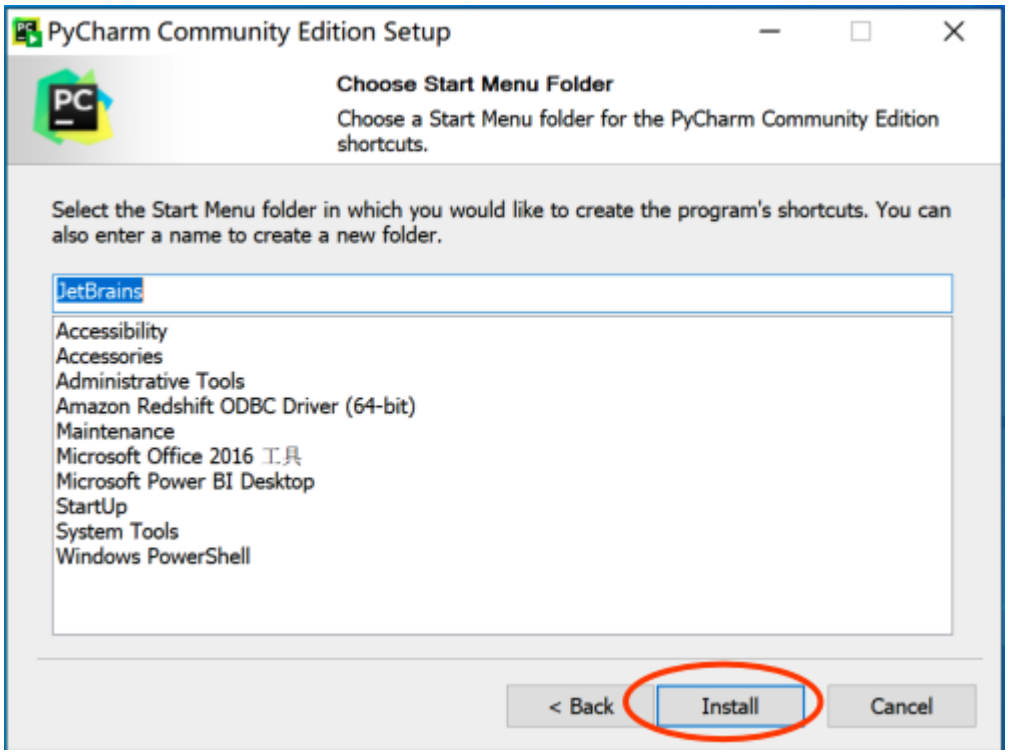
4. 选择安装路径，选择默认路径，点击 "next" 即可



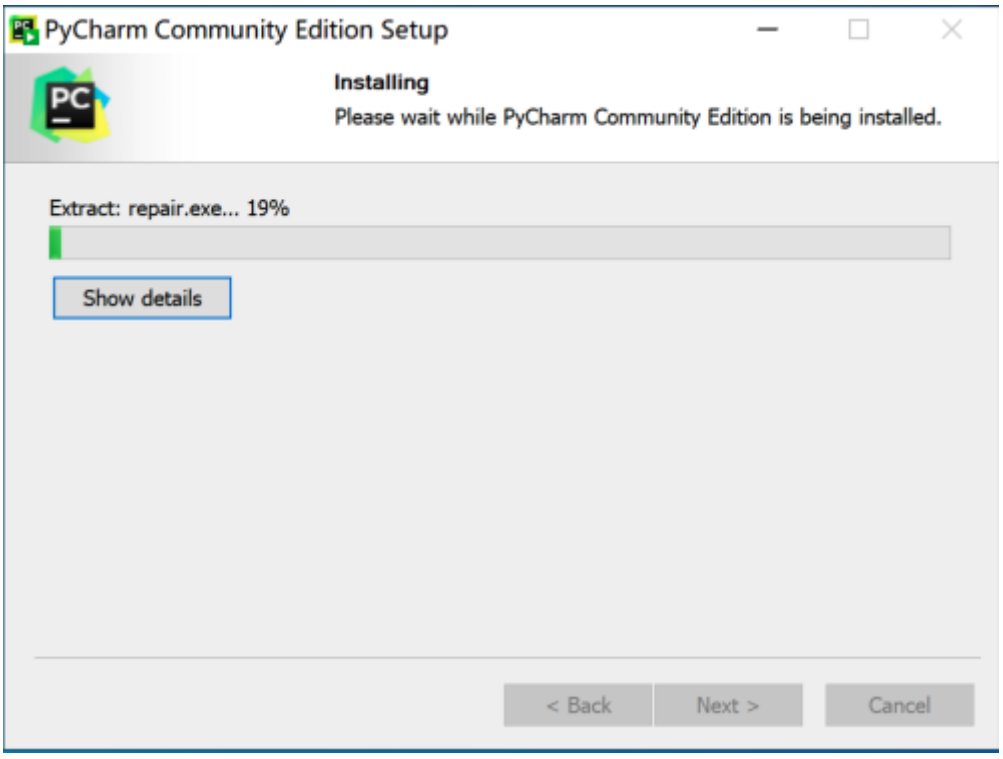
5. 选择安装选项。默认即可



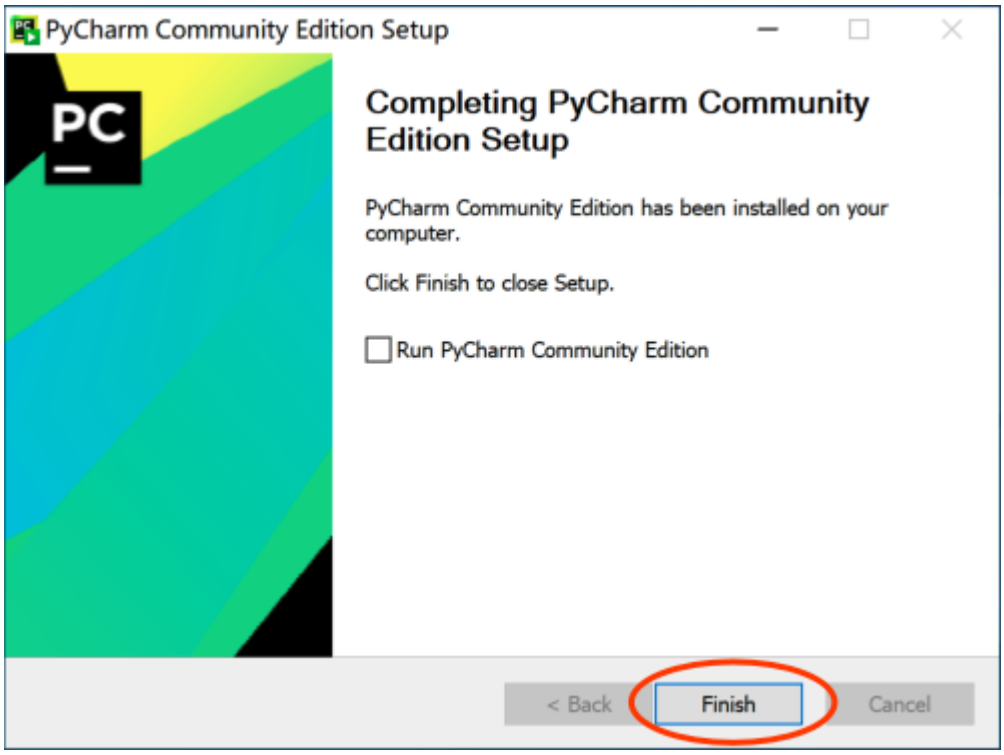
1. 选择一个开始菜单名称，默认即可



2. 进入安装过程

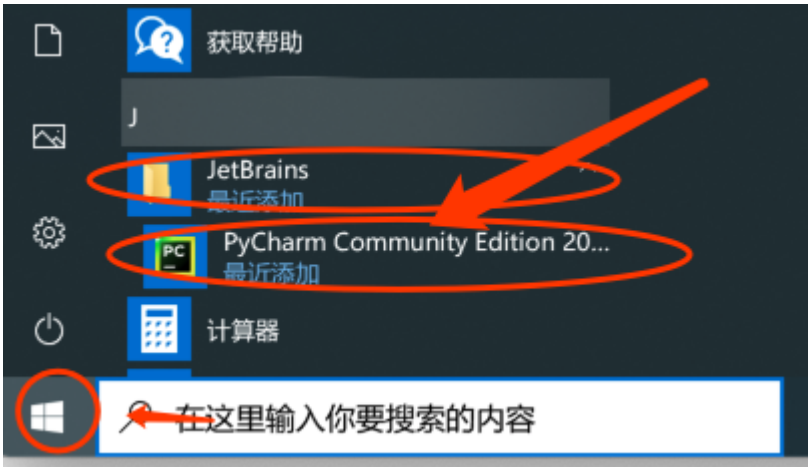


1. 安装结束点击 Finish 按钮结束安装

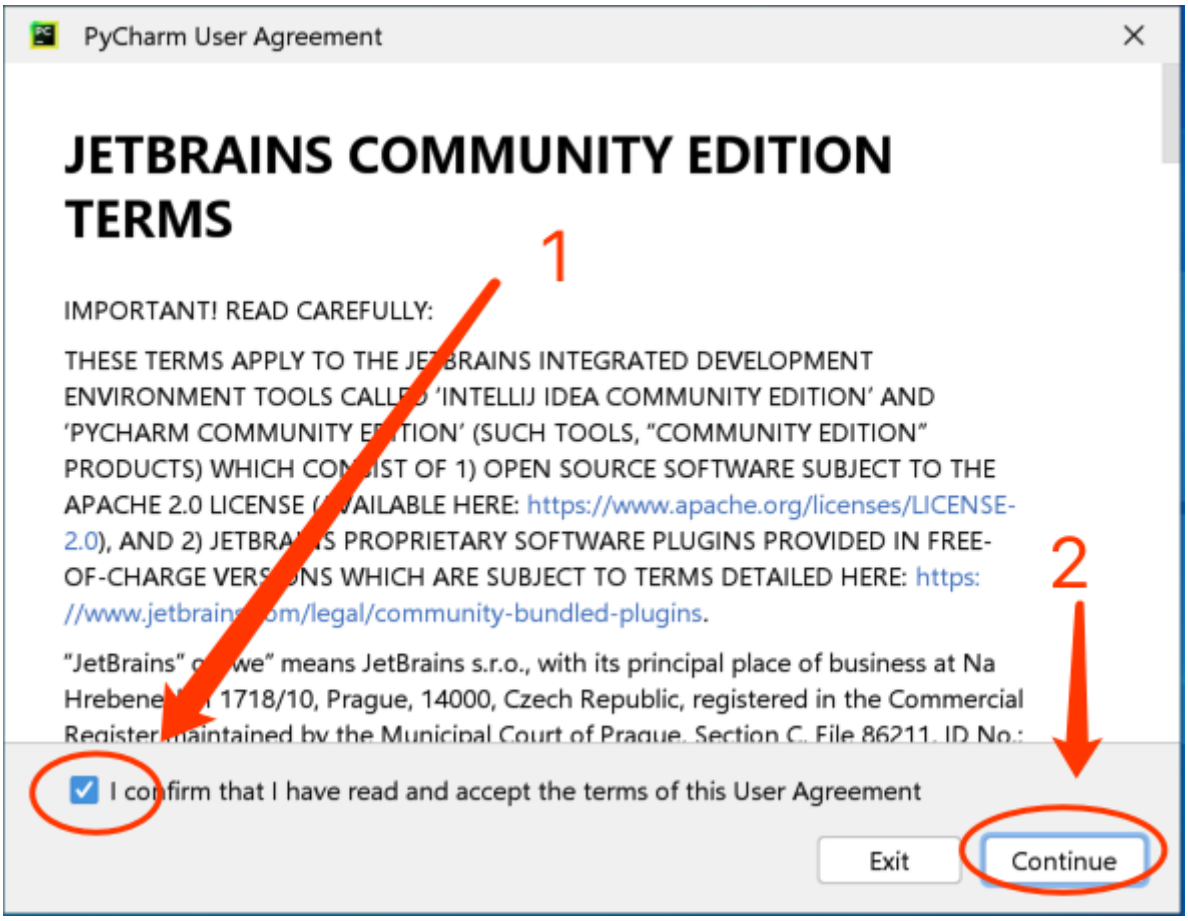


第三步：启动运行 PyCharm 社区版

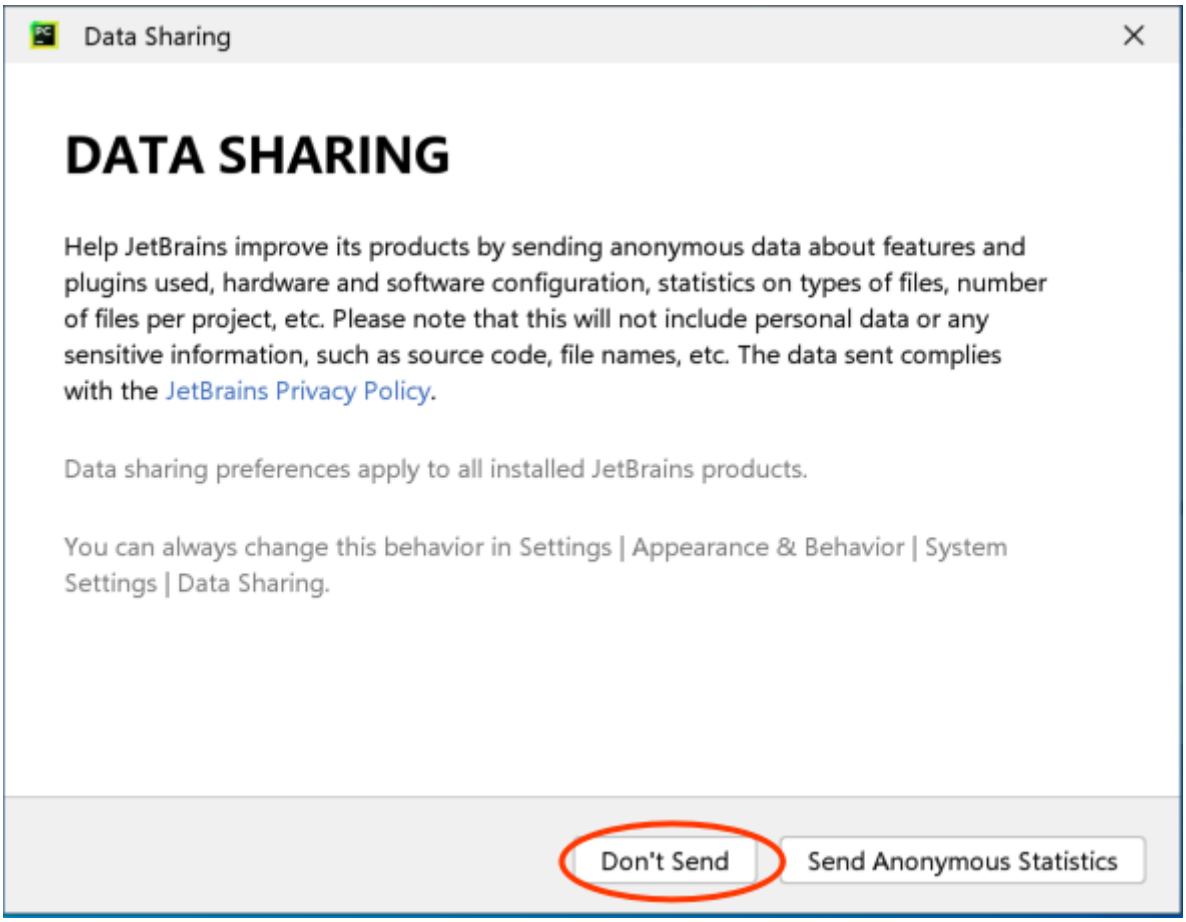
1. 从Windows 开始菜单中找到"PyCharm Community Edition 2020"



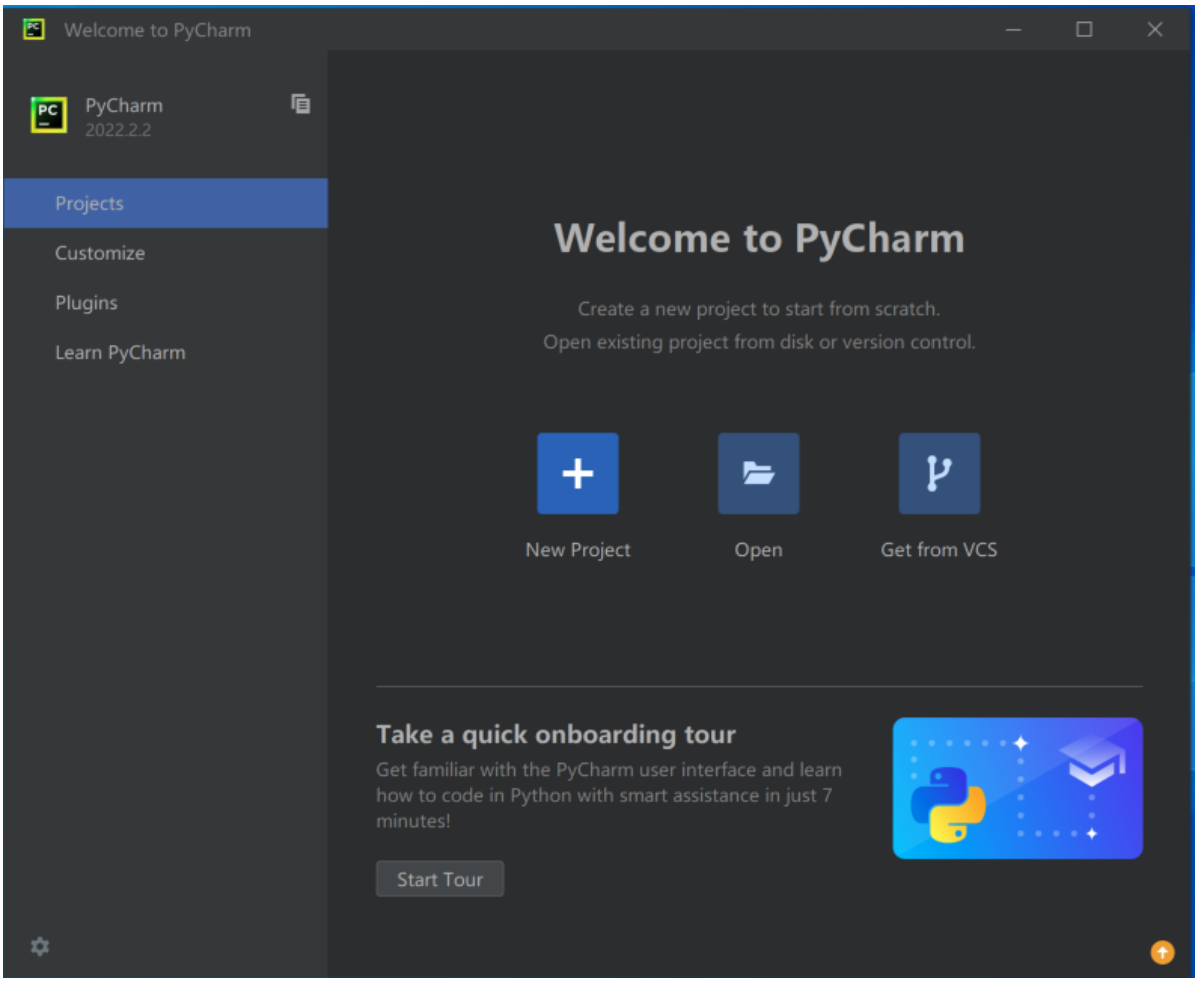
2. 第一次启动需要确认许可协议，勾选同意后点击"Continue" 继续下一步。



3. 数据分型，可以选择 "Don't Send" (不发送)

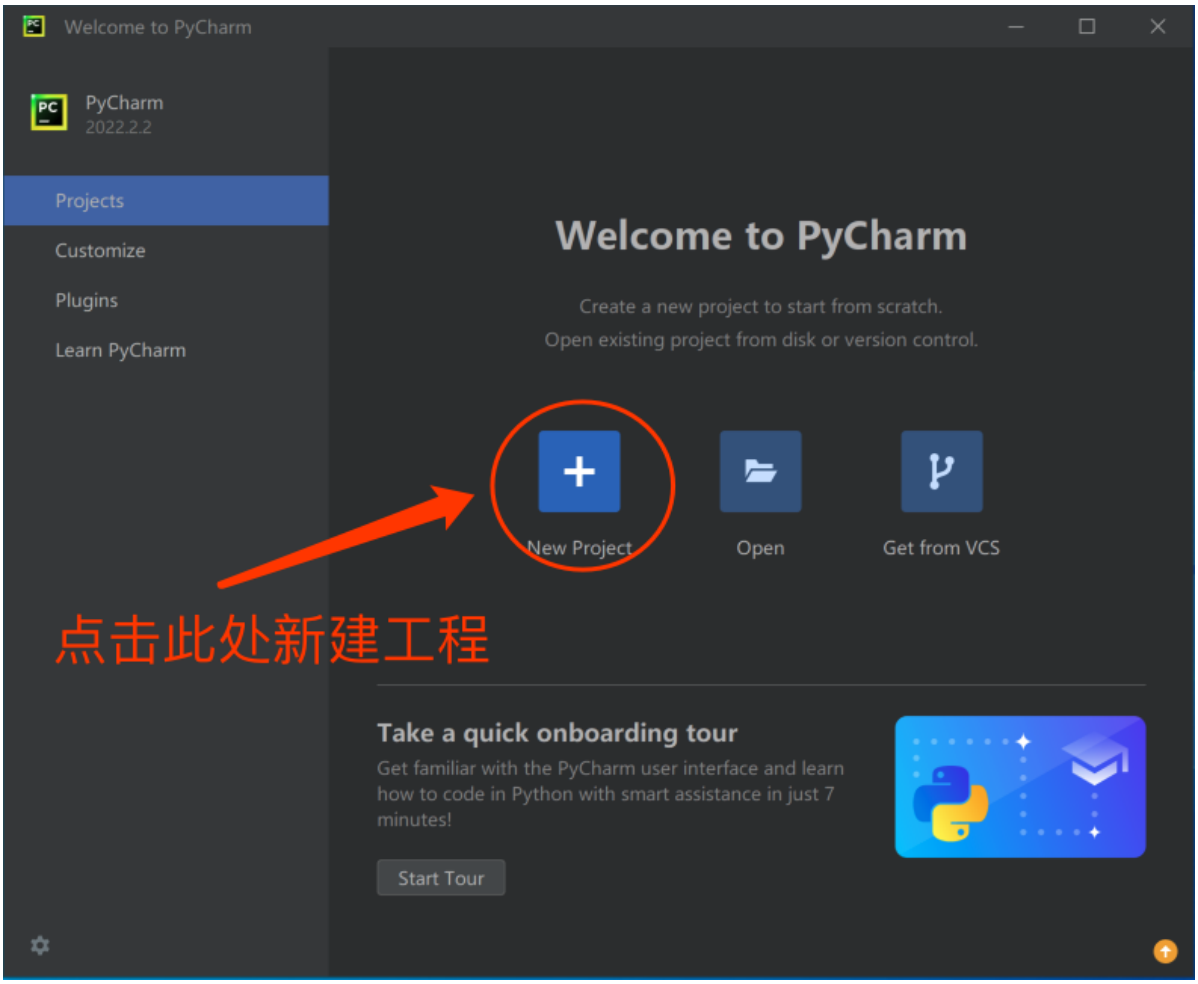


4. 进入新建工程界面

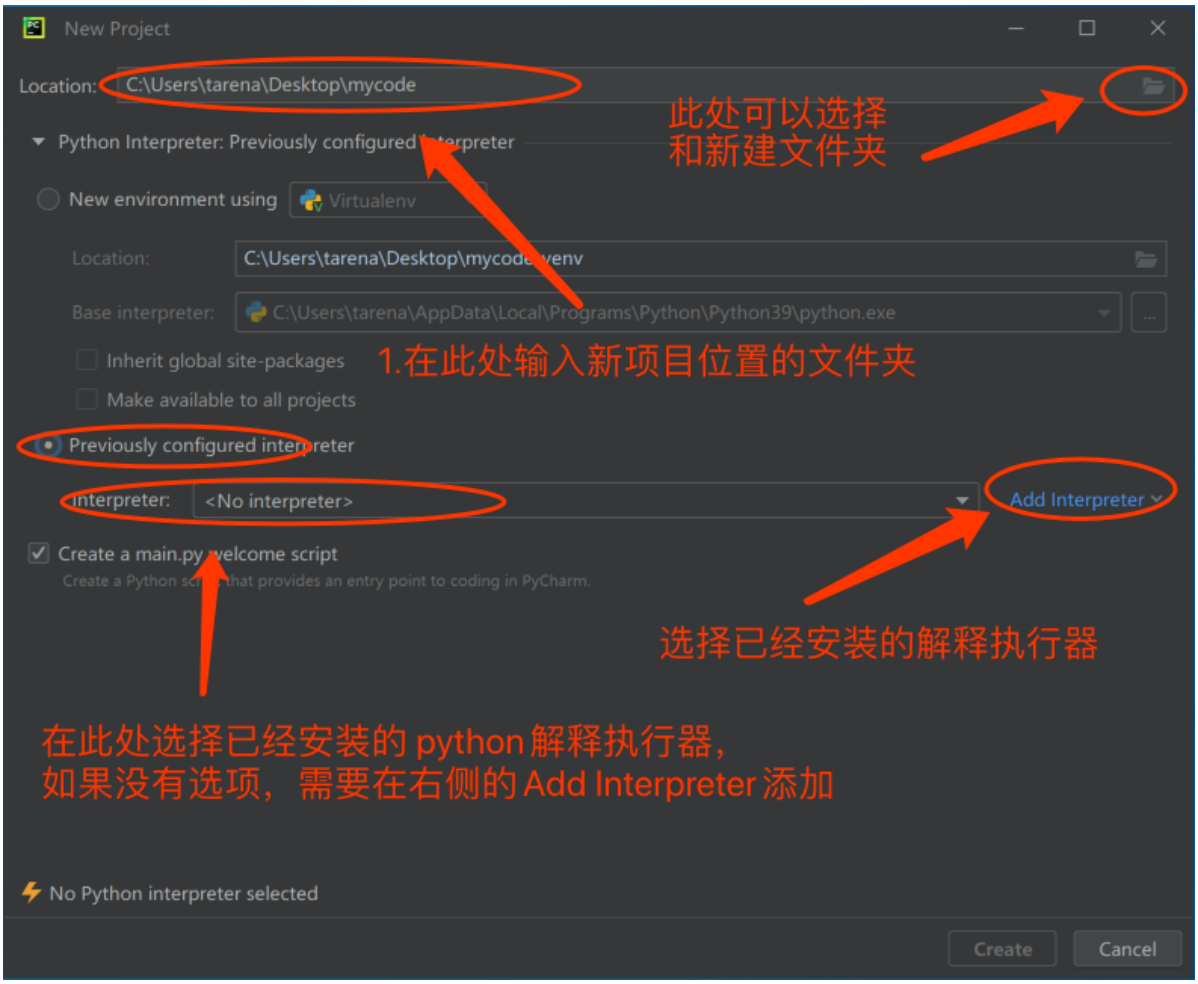


第四步：新建PyCharm 工程(Project)

1. 选择Create New Project 新建工程

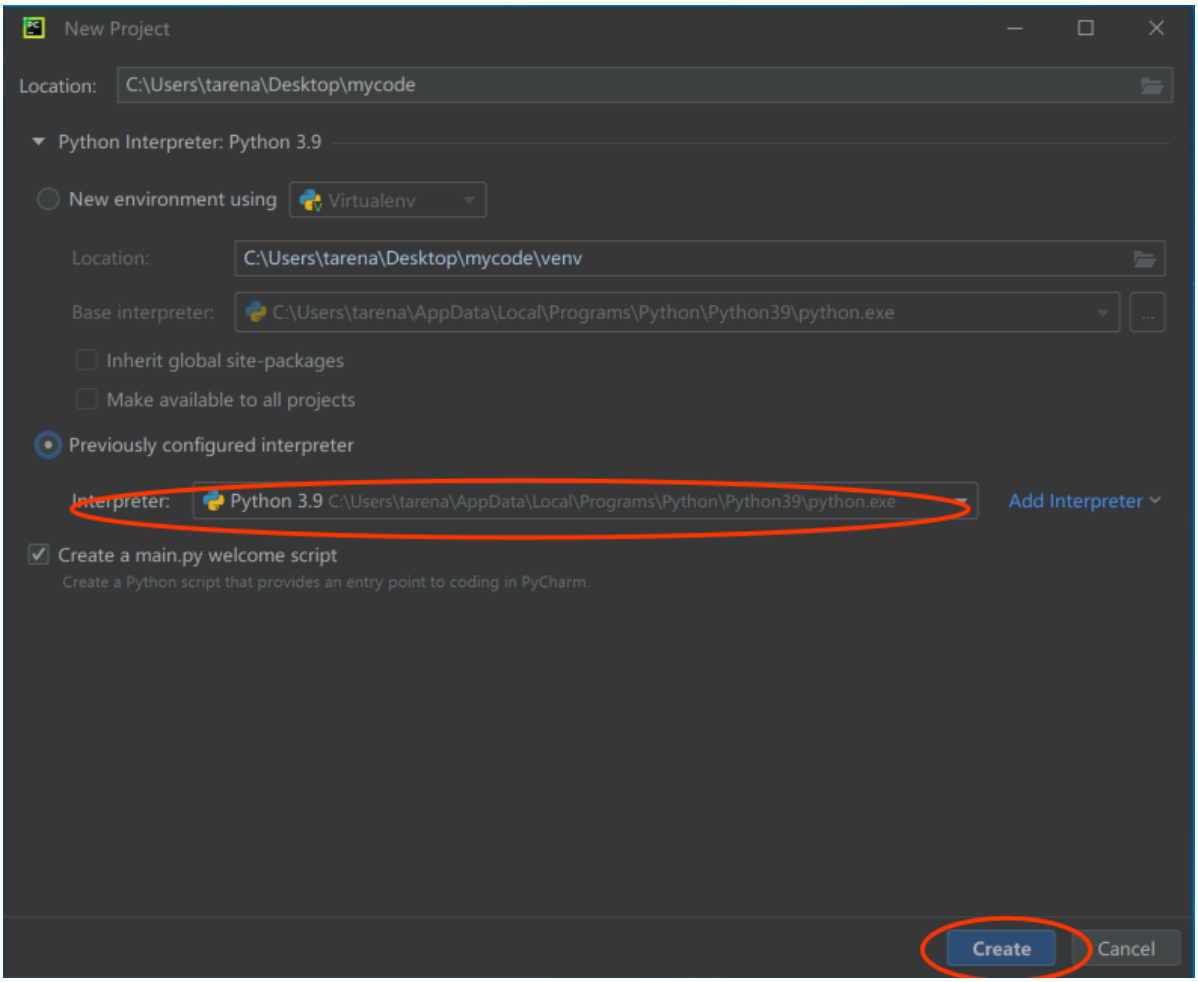


2. 选择项目文件夹的存放位置后。选择解释Python执行器的位置

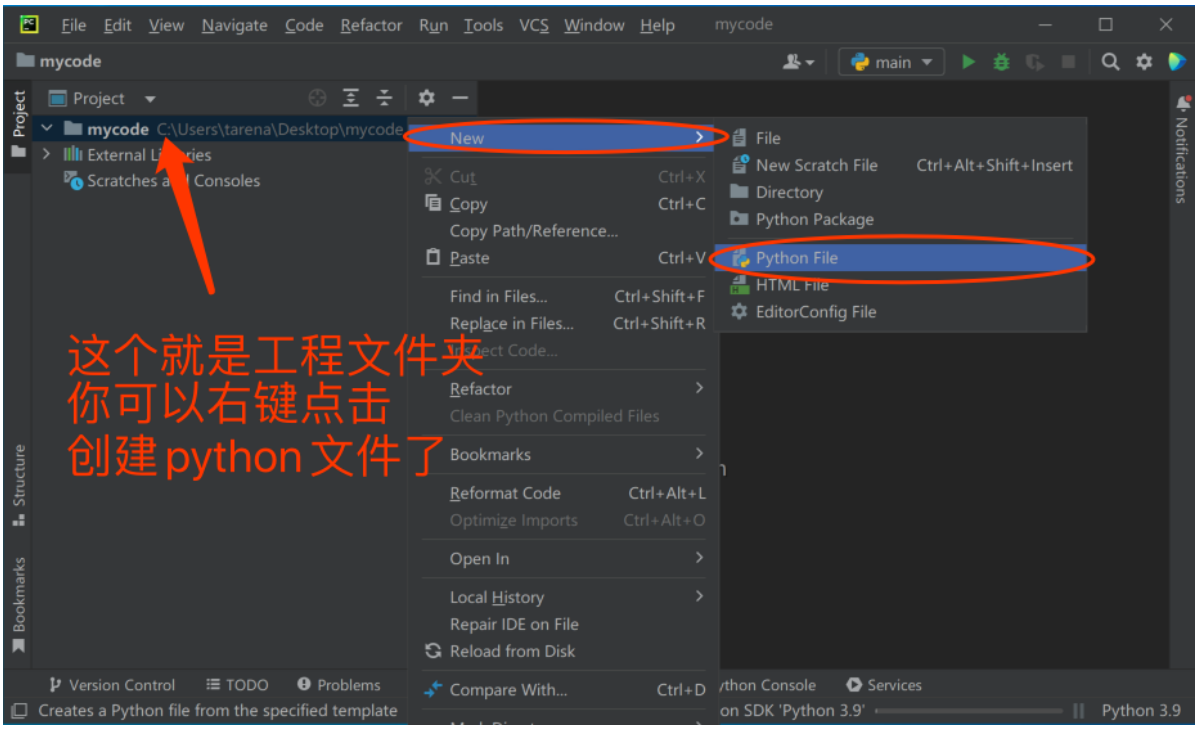


3. 选择 `Add Interpreter` 系统的解释执行器

4. 解释执行器已经选定，点击 Create 进入编辑界面



5. 在以下编辑界面可以新建 Python文件和其他的文件了



5. 第一个Python程序

目标：

1. 学会使用 PyCharm 编写 Python 程序
2. 学会 Python 代码的编写流程及运行方式

步骤一：新建项目

先使用 PyCharm 创建一个项目，项目的名字任意。创建项目的方法参见上一节。

> 项目：一个用于存放 python 源代码文件的文件夹
>
> 使用PyCharm编写代码需要创建一个文件夹用于存放 .py 结尾的文件。这个文件夹被称为项目文件夹。 创建项目时，需要设定当前代码的python 解释执行器。

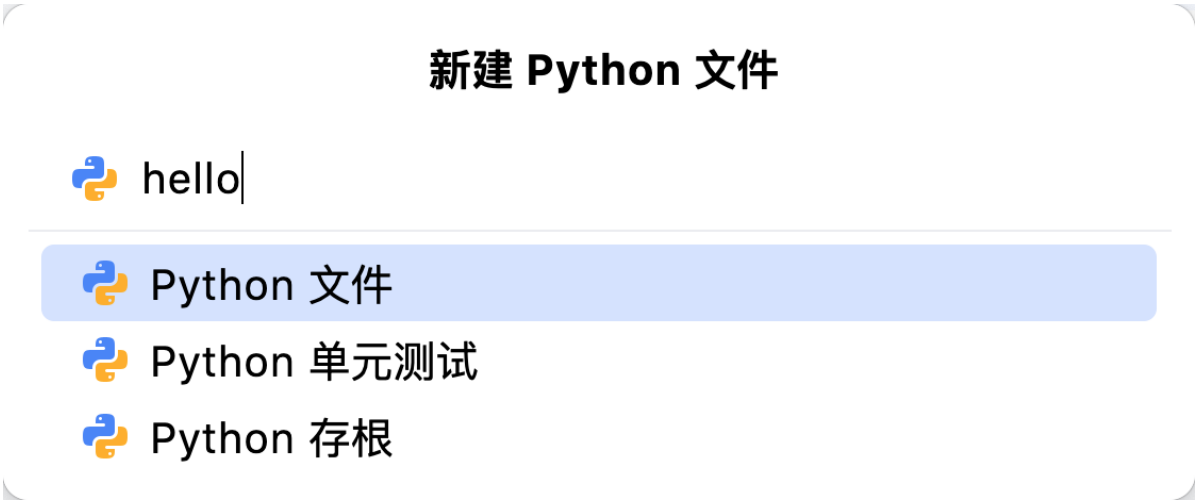
步骤二：创建 `hello.py` 文件

右键点击 `Project` 或 `项目` 当中的目录树，然后选择 `new` 或 `新建` ，再选择 `Python File` 或 `Python 文件`。然后输入文件的名字 `hello` 后按下回车按键。

操作如下图所示:

1. 选择创建文件

2. 输入文件名



Python的文件后缀默认是 `.py` 在 PyCharm 里可以手动添加 `.py`，如果不输入则会自动添加。

步骤三：编写程序内容

目标: 打印一行 `hello world!` 文字到控制台终端

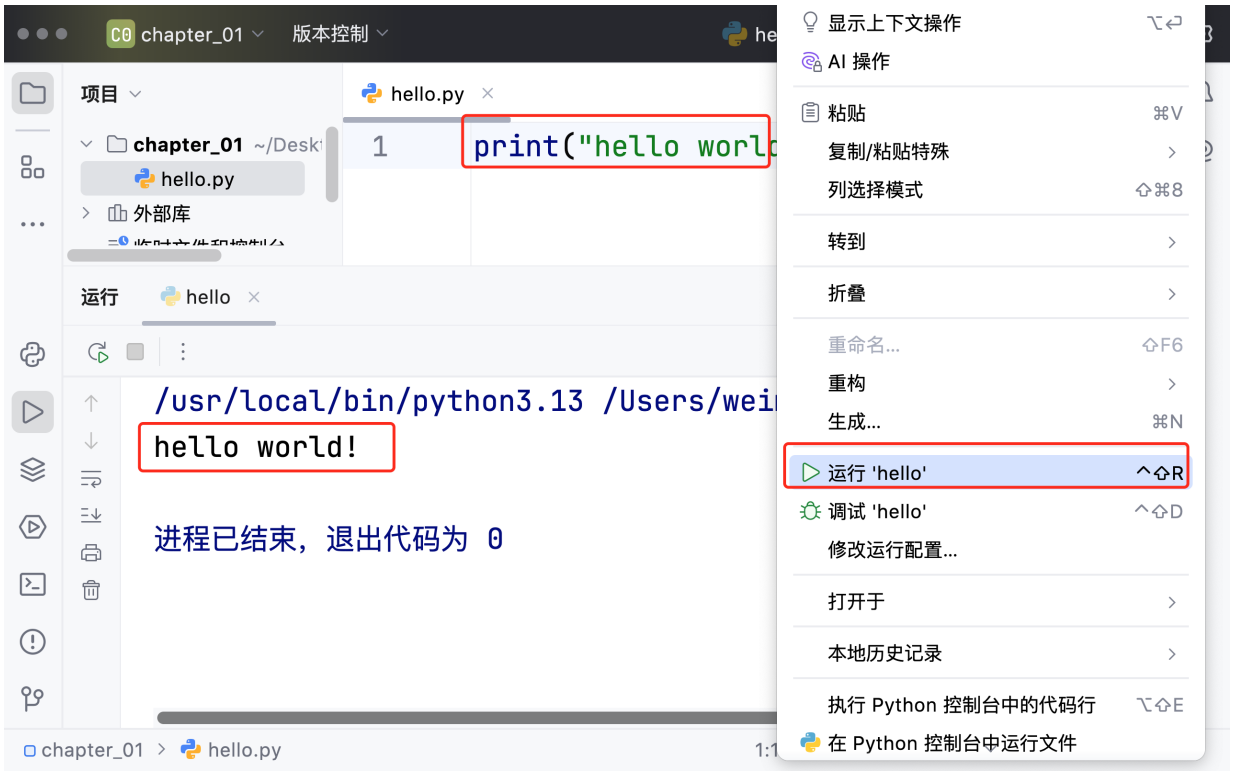
在PyCharm编辑器内输入如下代码

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

步骤四：运行 `hello.py` 这个程序

1. 在 `hello.py` 的文件编辑界面右键点击进入弹出菜单。
2. 选择 `Run 'hello'` 或 `运行 'hello'` 并点击。

操作如下图所示:



> 提示:
>
> 如果运行结果栏出现了红色的文字 `XxxxxxError: .....`，则说明出错了，检查代码再重新尝试。

错误提示示例如下：

```
  File "/Users/weimz/Desktop/chapter_01/hello.py", line 1
    print("hello world!')
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

终端

终端（Terminal）是用户与计算机系统进行文本交互的界面，允许通过输入命令直接操作系统或程序。

linux/Mac OS 终端执行 python 程序

> 在 Linux 或 Mac OS 操作系统下可以使用 "终端" 运行Python程序

```
% python3 hello.py
```

Windows 终端执行 python 程序

> 在Windows 下可以使用 "命令提示符" 运行Python程序

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

以交互模式在终端中运行Python的程序

进交互模式是指一问一答的运行模式，在python终端的 三个大于号提示符后面输入 python 的语句或表达式就能形成结果。

> 关于 语句和表达式后面会讲到，先知道有这么回事，别急。

在 Linux 或 Mac OS 操作系统下可以使用 "终端" 进入 Python 的交互运行模式，方法如下：

```
C:\> python
>>> print("hello world")
hello world
>>> exit()  # 退出交互模式
C:\>
```

Windows 终端执行 python 程序

在Windows 下可以使用 "命令提示符" 进入 Python 的交互运行模式，方法如下：

```
>>> print("hello world")
hello world
>>> exit()
%
```

> 调用exit() 函数可以退出交互运行模式 返回到 Windows 命令提示符

练习

- 写程序打印如下三行文字

```
我爱学习
我喜欢python的语法
我爱最简单的python语言
```

第二章、Python语法基础

1. 语法基础和注释

1.1 语法基础

语法（Syntax） 是指编写Python程序时必须遵循的规则和结构，它定义了如何正确地组合关键字、标识符、运算符和其他元素来形成有效的Python程序。如果违反语法规则，Python解释器会抛出 SyntaxError。

白话文解释

语法就是语言的模版或是套路，你只用按这个套路写，python的解释器才能认得你想表达的东西。

本站语法标注规则

1. 语法都会写在一个代码框内。
2. 代码框内的中文需要用相应的内容替代。
3. 代码框内的英文单词是关键字，不能改动；
4. 代码框内的符号通常是分隔符，如果下方无备注说明也不能改动。

> 本规则适用于本人文档、视频等全部内容。

编译原理中语法相关术语

- 物理行
  - 物理行是指你写入的文字，当有一个回车键输入时，你写入的就是一个物理行，即物理行指你看到的一行（python执行器再读取到 `'\n'` 或 `'\r\n'` 则认为是一个物理行）。

    > 关于 `'\n'` 和 `'\r\n'` 这个后面再解释

- 逻辑行
  - 逻辑行是指由一个或多个物理行组合而成的一个Python语义。Python在读取程序文件时，当一个语义不完整时，Python会读取下一个物理行而形成一个完整的语法逻辑，这样已由多个物理行组成的一个语法称为逻辑行。

- 字面值
  - 字面值是内置类型常量值的表示法。
- 标识符
  - 标识符（也称为 名称），是用来绑住一个数据的名称，就像 `魏明择` 这个标识符绑定的是正在给你们写教程的我一样，这个名字有起名字的规则，不能随便取名，否则不给上户口。当然python中的标识符 也有规则，后面会讲。
- 关键字
  - 关键字也被称为保留字，是特殊的标识符，不可用于普通标识符。就像 `中国` 这个名字一样，你去给孩子或公司取名子都不能用这个名。
- 运算符
  - 表示运算规则的符号。
- 分隔符
  - 用于分隔语法的符号，便于Python解释器识别你编写的内容
- 缩进
  - Python用来表示代码块的包含关系，通常位于行首的1个或多个空格（Python建议使用4个空格表示一个缩进）。
- 表达式
  - 用于计算并一定能返回一个值的字面值、变量、运算式或函数调用等，它一定能返回一个结果(数字、字符串、对象、函数、类或空(None)等。)。
- 语句
  - 一个完整的执行单位，就像你说的一句话一样，你完整的说出来一句话，别人也能按这句话去做事情，你说不完整或有错，别人也无法按你说的做。
- 简单语句
  - 一个逻辑行就能搞定的语句，不用组合。
- 复合语句
  - 语句内部需要有其他语句组合的语句，通常有缩进代表包含关系

python编程语言(基础篇)-语法内容

- 5种基础数据类型：str、int、float、bool、complex
- 5种容器类型: list、tuple、dict、set、frozenset
- 3个内置常量：False、True、None
- 2种表达式
- 10种运算和运算符
- 9种简单语句（共14种）

- 6种复合语句（共10种）
- 37个常用内建函数（共71个）
- 多个标准库和第三方库

准备开始学习吧!

1.2 注释

- 作用:
  - 让python解释执行器忽略注释部分的内容。不让其执行

语法规则

注释以英文的井号 （#） 开头，在物理行末尾截止。

例如

```
print("hello world")  # 我是注释，我不会被解释执行
# 这一整行都是注释
```

> 注: Python 3 只有一种注释，没有多行注释

练习

1. 写程序任意打印三行文字
2. 注释掉第二行打印语句
3. 在第三行打印语句的末尾用注释写上你的名字。

2. 基本输出函数 print

作用

向终端输出文字信息，能让用户看到。

> 当有多个数据要输出时，中间用英文的逗号(,) 分隔开。

函数调用语法格式

```
print(表达式1, 表达式2, ..., sep=' ', end='\n')
```

示例

```
print('你好')
print('你好', 'Python')
print('你好', 'Python', '我不换行', end='')
print('!!!')
print(2025, 5, 1, sep='-')
```

运行结果

```
你好
你好 Python
你好 Python 我不换行!!!
2025-5-1
```

说明

- 不同的函数名代表不同功能的函数
- 函数调用必须有一对小括号跟在函数名的后面
- 把要交给函数的数据放在小括号内
- 函数能够接收多少个数据（参数）由函数决定。

3. 基本输入函数 input

作用

让程序停下来，等待用户输入文字信息，返回用户输入文字的字符串

调用格式

```
input('提示信息')
```

示例

```
name = input('请输入您的姓名: ')
print('你刚才输入的名字是', name)
```

练习

写一个程序 myprog.py 做如下的事情。

1. 让用户输入用户名
2. 让用户输入密码
3. 打印 谁谁谁 的密码是 xxx

要求程序运行效果如下

```
请输入用户名： 魏明择
请输入密码： 123456
魏明择 的密码是 123456
```

4. 赋值语句和变量

赋值语句

用于将表达式执行的结果(重)绑定到一个名称(变量)，供后续使用。

语法

```
变量名 = 表达式
变量名1 = 变量名2 = 变量名3 = 表达式
变量名1, 变量名2, 变量名3 = 表达式1, 表达式2, 表达式3
```

示例

```
one_hundred = 99 + 1
a = b = c = 200
x, y = 300, 400
```

变量

变量就是一个名称，用于绑定python中的对象。这个对象可以是数据对象、函数、模块、类等。

作用

用于绑定python中的对象

白话文解释变量

变量就是魏明择的爷爷家门口的树桩，用来绑定在外面吃草的牛（对象），绳子很长，无论牛走到哪里，你只要找到树桩拽住绳子总能把牛找到。

邻居家的马也拴在爷爷家门口的另外一个树桩，为了区分这个树桩上拴的是什么，于是每个绑定动物的人要在树桩上写下名字，这个名字就是变量名，用来区分不通的家畜（对象）。

这个名子就是标识符(下一节讲解)。

> 扩展
>
> > python的变量等同于C语言中的非空指针和C++中的智能指针。

赋值语句说明

1. 第一次为变量赋值，python 会创建变量，同时绑定表达式执行的结果
2. 第二次或者以后为变量赋值，会改变原有变量的绑定关系
3. python 的变量没有类型，它绑定的对象才有类型
4. 变量赋值是一个自右向左的运算，将等号(=)右边表达式的计算结果，赋值给左边的变量
5. 一个变量只能绑定一个对象
6. 两个变量可以同时绑定同一个对象

练习

如下的几行语句，哪些不是合法的赋值语句？为什么？

```
a = 100
b = c = 200 + 300
x, y, z = 1, 2, 3
d, e = 400
a, b, c, d = 10 + 20, 30 + 40
```

5. 标识符和关键字

标识符

什么是标识符?

标识符是程序员在编写代码时为这些对象赋予的名字。

作用

标识符是用于标识变量名、函数名、类名、模块名和其他对象的名称。

标识符的命名规则

- 可用作为标识符的字符包括大小写形式的字母 A 到 Z，下划线 _ ，以及数字 0 到 9，但数字不能为第一个字符。
- python3 可以使用中文字符作为表示标识符的字符（不推荐）。
- Python标识符区分大小写，如：ABC和abc是两个不同的标识符。
- 不能使用保留字（也称为关键字），用作标识符。

标识符示例

- 合法的标识符

```
a         a1          abc          ABC       a1b2c3d4
one_hundred         print         input     打印机

getNameAge          get_name_age          GetNameAge
# 小驼峰命名法       匈牙利命名法          大驼峰命名法
```

- 不合法的标识符

```
1a        %abc        BB@cc        a100$
```

特殊命名方式的标识符

有四种标识符的写法有特殊含义，建议不要轻易使用

```
_*          # 如 _global_variable
_
__*__       #  如: __file__
__*         #  如 __abc
```

什么是关键字

关键字计算机编程语言中保留的标识符，关键字通常用于语法标识，关键字不能当成普通的标识符使用。

Python 3.13中全部的关键字

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
```

```
assert      del         global      not         with
async       elif        if          or          yield
```

python语法中没有的字符

以下三个字符不用于 Python中。在字符串字面值或注释外使用时，将直接报错

```
$       ?       `
```

练习

以下哪些不是合法的标识符?

```
100         abc123      else        __next__    dog

as          a$b         %abc%       def         from

china       c_h_i_n_a   student     1_0_0       xxx

jupyter_notebook    office@weimingze.com
```

第三章、字符串

数据类型的概念

什么是数据类型

- 数据类型是计算机内部存储数据的方式和方法，不同的数据采用不同的格式来存储。
- 对于不同的数字类型，计算规则通常也不同。

白话文解释

数据类型就是不同种类的东西，比如牛用来耕田，马用来运货，船用来过河。这里牛、马、船都是类型，不通的类型能做的事情自然不一样。

python数据类型的种类

- 字符串（文字)
- 数字
  - 整数、浮点数、复数
- 布尔类型
- 容器类型
  - 列表、字典、集合等

1. 字符串的字面值

什么是字面值

字面值是内置类型常量值的表示法。

字面值（Literal）是一种直接在源代码中表示数据或数值的方式，python在解析源码的过程中会在python解释器内部直接生产数据对象。

字面值可以直接用于赋值给变量、作为函数的参数或在表达式中直接使用。

字符串

字符串是用来记录人类的文字信息的一种数据类型，

字符串字面值写法：用英文的 ' 或 " 或 ''' 或 """ 开始或结束

示例

```
print('我喜欢Python!')
print("我喜欢Python!")
print('''我喜欢Python!''')
print("""我喜欢Python!""")
# 内容: I'm a teacher!
print("I'm a teacher!")
# 内容: I'm a teacher!, I love "Python"
print('''I'm a teacher!, I love "Python"''')
# 内容: 咏鹅鹅鹅鹅，曲项向天歌；白毛浮绿水，红掌拨清波。
print('咏鹅\n鹅鹅鹅，\n曲项向天歌；\n白毛浮绿水，\n红掌拨清波。')
print('''咏鹅
鹅鹅鹅，
曲项向天歌；
白毛浮绿水，
红掌拨清波。''')
```

运行结果

```
我喜欢Python!
我喜欢Python!
我喜欢Python!
我喜欢Python!
I'm a teacher!
I'm a teacher!, I love "Python"
咏鹅
鹅鹅鹅，
曲项向天歌；
白毛浮绿水，
红掌拨清波。
咏鹅
鹅鹅鹅，
曲项向天歌；
白毛浮绿水，
红掌拨清波。
```

字符串字面值总结

- 双引号的字符串的内部的单引号不算是结束符号；
- 单引号的字符串的内部的双引号不算是结束符号；
- 三引号字符串的内部可以包含单引号和双引号；
- 三引号字符串又称为所见即所得字符串，每一次换行会转换成换行符 '\n'。

2. 字符串的转义

字符串的转义

字符串的字面值中，可以用反斜杠 `\` 后加一个或多个字符，代表某一个单个字符。

当有语法字符出现在字符串内容中时可以使用转义字符。

常见的转义字符

```
\'    # 代表一个单引号
\"    # 代表一个双引号
\n    # 代表一个换行符
\\    # 代表一个反斜杠
```

示例

```
# 我是单引号', 我是双引号",我是三重单引号''', 我是三重双引号"""。
sentence = '我是单引号\', 我是双引号\",我是三重单引号\'\'\', 我是三重双引号"""。'
print(sentence)

sentence2 = r'\n\"\t\"'
print(sentence2)
```

结果

```
我是单引号', 我是双引号",我是三重单引号''', 我是三重双引号"""。
\n\"\t\"
```

python中全部的转义字符

未标注 'r' 或 'R' 前缀，字符串和字节串字面值中，转义序列以类似 C 标准的规则进行解释。

转义序列	含意
`\<newline>`	末尾的折行符号
`\\`	反斜杠（\）
`\'`	单引号（'）
`\"`	双引号（"）
`\a`	ASCII 响铃（BEL）
`\b`	ASCII 退格符（BS）
`\f`	ASCII 换页符（FF）
`\n`	ASCII 换行符（LF）
`\r`	ASCII 回车符（CR）
`\t`	ASCII 水平制表符（TAB）
`\v`	ASCII 垂直制表符（VT）
`\ooo`	八进制数 ooo 字符
`\xhh`	十六进制数 hh 字符

原始字符串(raw string)

什么是原始字符串

当字符串字面值前有 'r' 或 'R' 开头时，此字面值内部的转义字符 `'\'` 不再有效。

示例

```
print(r"\'\n")  # 打印\'\n 这四个字符
```

3. 字符串的运算

拼接和重复运算

- `+` 运算符，用于拼接字符串(生成新的字符串)；
- `*` 运算符，用于生成重复的字符串。

示例

```
s1 = '我喜欢'
s2 = "Python编程语言"
s3 = s1 + s2
print(s1)
print(s2)
print(s3)
# * 运算符    字符串 * 整数
s4 = s1 * 3
print(s4)  # 我喜欢我喜欢我喜欢
print('s1 绑定的字符串的长度是:', len(s1))
print('s2 绑定的字符串的长度是:', len(s2))
```

运行结果

```
我喜欢
Python编程语言
我喜欢Python编程语言
我喜欢我喜欢我喜欢
s1 绑定的字符串的长度是: 3
s2 绑定的字符串的长度是: 10
```

len(x) 函数

len(x) 函数可以用于获取字符串的长度（字符的个数）。

调用格式

```
len(x)
```

示例

```
s = 'ABC'
print(len(s))  # 打印 3
s = 'bye bye!'
print(len(s))  # 打印 8
```

4. 字符串的索引

什么是索引

索引（Index）是一种用于访问序列类型数据中单个元素的方法。

序列类型

序列类型是指数据有先后顺序排列的数据类型。

白话文解释

序列可以理解成现实中的火车，火车是一个整体，他是由每一节车厢组成的，每一节车厢是可以单独作为事物存在的。我们把车厢有先后顺序的连载一起就组成了火车（序列）。

python中的序列类型有:

- 字符串 str
- 列表 list
- 元组 tuple
- range
- 字节串
- 字节数组

字符串是序列类型，如:

```
'Python'
```

示意图

> 上面字符串有个字符组成（相当于一列火车每节车厢拉一个字符）。

索引

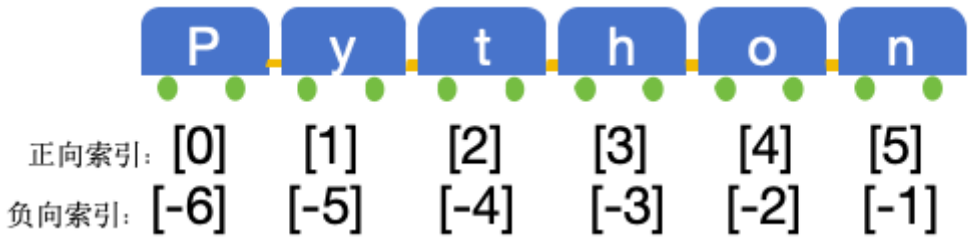
用于定位序列中一个位置的整数，在 Python 语言中以0开始。

白话文解释

索引就是车厢号（1，2，3，4，.....）但python中是(0, 1, 2, 3, ...)

索引的类型

- 正向索引：从序列的开头开始，索引从0递增。
- 负向索引：从序列的末尾开始，索引从-1递减。



索引的语法

```
序列[整数表达式]
```

返回值

对应位置的一个元素(字符)。

示例

```
word = 'Python'

print(word[0])
print(world[-1])
```

运行结果

```
P
n
```

说明

1. 索引必须是整数。
2. 索引越界会引发IndexError类型的错误。

练习

写一个程序，输入一段文字。

1. 打印您输入的第一个字。
2. 打印您输入的最后一个字。

5. 字符串的切片

什么是切片

切片（slicing）是一种用于访问序列类型数据中多个元素的方法。

通过切片操作，你可以获取序列的一部分来创建新的序列或修改原始序列。

白话文解释

你很有钱，你坐高铁你直接包下连续的几节车厢，比如 2～4 节车厢，然后你在第三节车厢打乒乓求！那 2～4 节车厢就是你整列车的切片。

切片的语法

```
序列[开始位置:结束位置:步长]
```

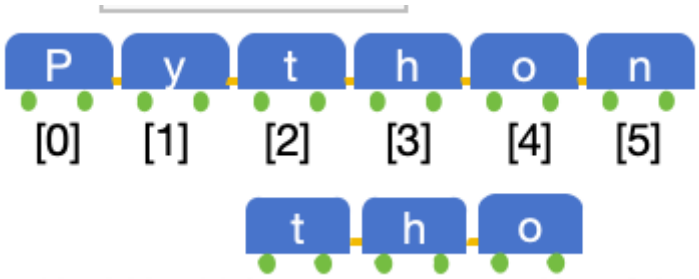
返回值

对应位置的多个元素组成的新的序列。

切片示例

```
'Python'[2:5]
```

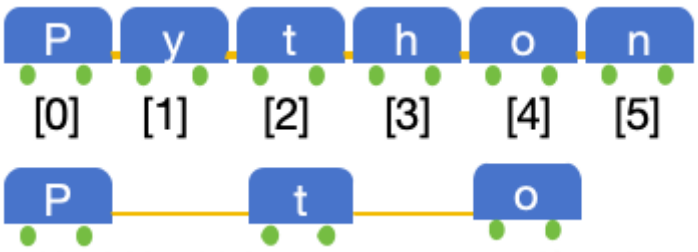
切出效果



切片示例2

```
'Python'[0:5:2]
```

切出效果



> 你来写代码试试吧，不学一定不会，不动手不一定会。

切片语法说明

```
序列[开始位置:结束位置:步长]
```

1. 包含开始位置，一定不包含结束位置。
2. 开始位置和结束位置必须是表示索引位置的整数并可以省略。
3. 第二个冒号和步长可以省略，省略后步长默认为1。
4. 步长可以是不等于零的任意整数。
5. 步长为正数，开始位置省略表示从头开始，结束位置省略表示到末尾（包含）。
6. 步长为负数，开始位置省略表示从末尾开始，结束位置省略表示到头（包含）。

练习

写一个程序，输入一段文字。

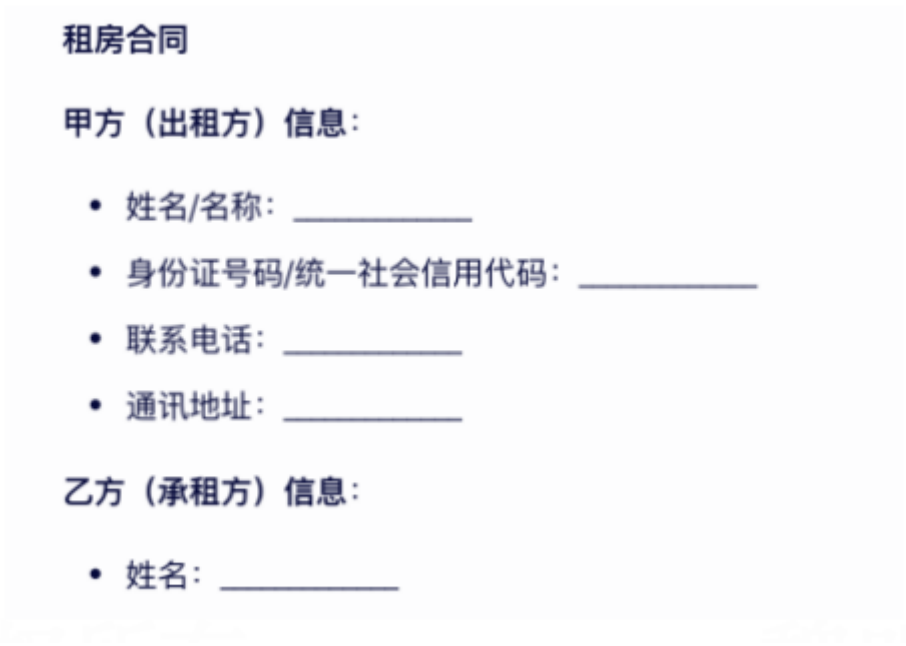
- 打印您输入的前两个字。
- 打印您输入的后两个字。

6. 字符串格式化

什么是格式化

格式化是指用变化的内容，填充固定的模版，生成新的字符串的过程叫作格式化。

例如合同范本，只要填写变化的少部分信息就可以生成完整的合同。如下图：


租房合同

甲方（出租方）信息：

- 姓名/名称：_____
- 身份证号码/统一社会信用代码：_____
- 联系电话：_____
- 通讯地址：_____

乙方（承租方）信息：

- 姓名：_____

字符串的格式化方法

python 提供的三种字符串格式化的方法：

1. str.format() 方法；
2. f 字符串（python 3.6起用）；
3. printf 风格的字符串格式化。

方法的概念和调用

什么是方法

方法是某类数据对象自己的函数。只能由此类数据的对象来调用。

方法的调用语法

```
对象.方法名(调用参数列表)
```

6.1 str.format()方法

占位符：一对大括号{}

占位符的概念

占位符有犹如上面合同范本中的下划线，就是先预留变化数据的位置，后面在用变化的数据来进行填充。

示例1

```
>>> '姓名:{}, 年龄:{}'.format("张三", 18)
'姓名:张三, 年龄:18'
>>> '姓名:{}, 年龄:{}'.format("李四", 16)
'姓名:张三, 年龄:18'
```

> 占位符的个数要等于.format()方法内的参数

示例2

```
template = "甲方:__{}__\n乙方:__{}__\n合同约定如下......"
print(template)
new_str1 = template.format("张三", "李四")
print(new_str1)
```

运行结果

```
甲方:__{}__
乙方:__{}__
合同约定如下......
甲方:__张三__
乙方:__李四__
合同约定如下......
```

练习

写一个程序

1. 输入您的用户名；
2. 输入您的住址；

用str.format方法打印如下格式的内容

```
请输入您的姓名：老魏
请输入您的住址：北京市朝阳区
老魏住在北京市朝阳区。
```

6.2 f字符串

> f字符串 从 python3.6起用，之前的版本不支持。

f-string 占位符：

一对大括号，内部添加变量名`{变量名}`

语法：字符串的字面值写法前面加 'f' 或 'F' 前缀，内部可以加入多个占位符。

如:

```
f'占位符和字符串'
f"占位符和字符串"
f'''占位符和字符串'''
f"""占位符和字符串"""
```

示例

```
>>> name = '张三'
>>> age = 19
>>> f'姓名:{name}, 年龄:{age}'
'姓名:张三, 年龄:19'
```

> 在f字符串运行时，变量必须已经创建。

练习 写一个程序

1. 输入您的用户名；
2. 输入您的住址；

用 f字符串 打印如下格式的内容。

```
请输入您的姓名：老魏
请输入您的住址：北京市朝阳区
老魏住在北京市朝阳区。
```

6.3 printf风格的字符串格式化

占位符：'%' 开始，加宽度、精度等，加转换符。

语法

使用 % 取摸运算符（也叫求余数运算符）。

```
格式化字符串 % 参数
格式化字符串 % (参数1,参数2, 参数3,...)
```

格式化字符串中的转换符

常用的转换符

转换符	含意
'd'	有符号的十进制整数。
'f'	十进制格式的浮点 数(小数)。
's'	使用 str() 转换任何 Python 对象。
'%'	不转换参数，在结果中输出一个'%' 字符。

示例

```
# 此示例示意 C 语言 printf 风格的字符串

# 一个占位符的情况
count = 120
new_str1 = '人数是%d' % count
print(new_str1)

# 多个占位符的情况
title = "圆周率"
pi = 3.141592653589793
new_str2 = '%s的值是%f' % (title, pi)
print(new_str2)
```

运行结果

```
人数是120
圆周率的值是3.141593
```

% 和 转换符 之间的旗标语法

```
%[- + 0 宽度.精度]类型码
```

说明

- `-` : 左对齐(默认是右对齐);
- `+` : 显示正号;
- 0 : 左侧空白位置补零;
- 宽度 : 整个数据输出的宽度;
- 精度 : 保留小数点后多少位(默认6位)。

旗标	含意
`'#'`	值的转换将使用“替代形式”（具体定义见下文）。
`'0'`	转换将为数字值填充零字符。
`'-'`	转换值将靠左对齐（如果同时给出 `'0'` 转换，则会覆盖后者）。
`' '`	(空格) 符号位转换产生的正数（或空字符串）前将留出一个空格。
`'+'`	符号字符 (`'+'` 或 `'-'`) 将显示于转换结果的开头（会覆盖 "空格" 旗标）。

示例

```
>>> "%10d" % 123
'       123'
>>> "%-10d" % 123
'123       '
>>> "%10s" % 'abc'
'       abc'
>>> "%-5s" % 'abc'
'abc  '
>>> "%05d" % 123
'00123'
>>> "%7.3f" % 3.1415926535
'  3.142'
>>> "%07.3f" % 3.1415926535
'003.142'
>>>
```

全部的转换符

转换符	含意
`'d'`	有符号十进制整数。
`'i'`	有符号十进制整数。
`'o'`	有符号八进制数。
`'u'`	过时类型 -- 等价于 `'d'`。
`'x'`	有符号十六进制数（小写）。
`'X'`	有符号十六进制数（大写）。
`'e'`	浮点指数格式（小写）。
`'E'`	浮点指数格式（大写）。
`'f'`	浮点十进制格式。
`'F'`	浮点十进制格式。
`'g'`	浮点格式。 如果指数小于 -4 或不小于精度则使用小写指数格式，否则使用十进制格式。
`'G'`	浮点格式。 如果指数小于 -4 或不小于精度则使用大写指数格式，否则使用十进制格式。
`'c'`	单个字符（接受整数或单个字符的字符串）。
`'r'`	字符串（使用 `repr()` 转换任何 Python 对象）。
`'s'`	字符串（使用 `str()` 转换任何 Python 对象）。
`'a'`	字符串（使用 `ascii()` 转换任何 Python 对象）。
`'%'`	不转换参数，在结果中输出一个 `'%'` 字符。

练习

写一个程序

1. 输入您的用户名；
2. 输入您的住址；

用 C 语言 printf 风格的字符串格式化 打印如下格式的内容。

```
请输入您的姓名：老魏
请输入您的住址：北京市朝阳区
老魏住在北京市朝阳区。
```

7. 字符串的方法

方法的概念和调用

什么是方法

方法是某类数据对象自己的函数。只能由此类数据的对象来调用。

方法的调用语法

```
对象.方法名(参数1， 参数2, ...)
```

字符的方法详见官方文档:

https://docs.python.org/zh-cn/3/library/stdtypes.html

常用的字符串方法

方法	说明
str.title()	返回原字符串的标题，每个单词第一个字母为大写，其余字母为小写。
str.upper()	返回原字符串的副本，其中所有区分大小写的字符均转换为大写。
str.lower()	返回原字符串的副本，其中所有区分大小写的字符均转换为小写。
str.count(sub[, start[, end]])	返回子字符串 sub 在 [start, end] 范围内非重叠出现的次数。
str.strip([chars])	返回原字符串的副本，移除其中的前导和末尾字符。
str.replace(old, new[, count])	返回字符串的副本，其中出现的所有子字符串 old 都将被替换为 new。
str.find(sub[, start[, end]])	返回子字符串 sub 在 s[start:end] 切片内被找到的最小索引。
str.index(sub[, start[, end]])	类似于 find()，但在找不到子字符串时会引发 ValueError。

示例

```
>>> s = 'welcome to BEIJING. welcome to HARBIN.'
...
>>> s.upper()
'WELCOME TO BEIJING. WELCOME TO HARBIN.'
>>> s.lower()
'welcome to beijing. welcome to harbin.'
>>> s.title()
'Welcome To Beijing. Welcome To Harbin.'
>>> s.count('TO')
0
>>> s.count('to')
2
>>> s.strip('welcome ')
'to BEIJING. welcome to HARBIN.'
>>> s.replace('welcome to', 'Welcome To')
'Welcome To BEIJING. Welcome To HARBIN.'
>>> s.find('BEIJING')
11
>>> s.find('SHANGHAI')
-1
>>> s.index('BEIJING')
```

```
11
>>> s.index('SHANGHAI')
Traceback (most recent call last):
  File "<python-input-11>", line 1, in <module>
    s.index('SHANGHAI')
    ~~~~~~~^^^^^^^^^^^^
ValueError: substring not found
>>>
```

练习

写程序输入一串文字

1. 打印出这串文字的长度；
2. 打印出这段文字中 '我' 字的个数；
3. 将这段文字中的英文字母都转换为大些字母并打印；
4. 将这段文字中的英文字母都转换为小些字母并打印；
5. 将这段文字中的每个英文单词第一个字母为大写，其余字母为小写并打印。

第四章、数字

1. 整数（int）

什么是整数

整数是一种数据类型，用于表示没有小数部分的数值。整数可以是正数、负数或零。

示例

```
100        0        -1
```

> 整数的类型名是 int。字符串的类型名是 str。 python的整数几乎不限制最大值。

整数字面值

- 十进制写法
  - 由数字"0"~"9"组成，不能以0开头，可以用下划线分隔数据。
- 八进制写法
  - 以"0o"或 "00"开头，后跟数字"0"..."7"组成，可以用下划线分隔数据。
- 十六进制写法
  - 以"0x"或 "0X"开头，后跟数字"0"..."9"、"a"..."f" | "A"..."F"组成，可以用下划线分隔数据。
- 二进制写法
  - 以"0b"或 "0B"开头，后跟数字"0"或"1"组成，可以用下划线分隔数据。

示例

一模一样的 整数 1000 有不同的写法，在计算机内部也都是相同的数字，如下所示：

```
>>> 1000
1000
>>> 0b1111101000
1000
>>> 0B1111101000
1000
>>> 0o1750
1000
>>> 0O1750
1000
```

```
>>> 0x3e8
1000
>>> 0X3E8
1000
```

正、负号运算符

语法

```
+ 数字      # 原有的数值不变

- 数字      # 将原有的数值取负数
```

示例

```
>>> x = 100
>>> -x
-100
>>> --x
100
```

说明

正、负号运算符都是一元运算符，只有一个元素参加运算。

- 练习：

```
计算0xFFFF      表示多少
计算0xFFFFFFFF 表示多少
计算0b1001      表示多少?
计算0o71        表示多少?
试着把 9 转为二进制表示
    把 64 转为八进制表示
    把 18转为 十六进制表示
```

2. 浮点数（float）

什么是浮点数

浮点数（floating-point number）又称为小数，是一种用于表示具有小数部分的数值类型。

通常用于科学计算、工程计算、金融计算等需要精确小数表示的领域。

浮点数的小数部分可以为零，如：

```
3.1415926       0.0314E2       -0.0001
```

> 浮点数的类型名是 float。

浮点数字面值

十进制表示法

- 直接写出小数部分

```
3.14     0.14   .14     3.0     3.
```

科学计数法

- 使用 e 或 E 来表示10的幂。格式为: 小数 + "e"或"E" + (正负号) + 十进制整数。

```
2.9979E8     # 等于29979000.0
```

示例

```
>>> 3.14
3.14
>>> 3.0
3.0
>>> 3.
3.0
>>> 0.14
0.14
>>> .14
0.14
>>> 0.0
0.0
>>> 0.
0.0
>>> .0
0.0
>>> 2.9979E8
299790000.0
>>> 2.9979e8
299790000.0
>>>
```

3. 复数（complex）

什么是复数

复数（complex number）是一种特殊的数值类型，用于表示具有实部和虚部的数。

复数的形式通常表示为 a + bi，其中 a 是实部（real part），b 是虚部（imaginary part），i 是虚数单位，满足 i的平方等于-1。

复数 字面值

```
1j
(2j)
1+1j
1-1j
(-100+100j)
```

示例

```
1+2j    # 表示数学中的 1+2i
```

> 复数的类型名是 complex。

虚数字面值

```
整数或浮点数 + "j"或"J"
```

示例

```
100J         1j          2.1j
```

复数的属性

什么是属性

属性是数据内部的变量。

属性的语法格式

```
对象.属性名
```

复数的属性

- real：获取复数的实部。
- imag：获取复数的虚部。

示例

```
>>> c = 100 + 200j
>>> c
(100+200j)
>>> c.real
100.0
>>> c.imag
200.0
>>>
```

4. 二元算术运算

python中提供了一些符号用于表示运算的规则，这称之为运算符。

二元算术运算符

二元算术运算符是指有两个数据元素来参加运算并得到一个结果的运算符。

python有七个二元算术运算符如下：

```
+         # 加法
-         # 减法
*         # 乘方
/         # 除法
//        # 整除
%         # 取摸（也叫求余数）
@         # 矩阵乘法（此节不讲）
```

二元算术运算符的语法规则

```
左表达式   运算符   右表达式
```

示例

```
3.14 * 100
```

> 二元是指两个数据元素参加运算；
>
> 二元算术运算符的优先级同数学中的优先级。

整除和求余数

- `//` 运算符用于求除法的整数商结果，小数部分会舍去。
- `%` 运算符用于求除法的整除后的余数。

示意图

示例

```
>>> 14 + 3
17
>>> 14 - 3
11
>>> 14 * 3
42
>>> 14 / 3
4.666666666666667
>>> 14 // 3
4
>>> 14 % 3
2
>>>
```

5. 幂运算

幂运算符 `**` 也是一个二元运算符，用来计算乘方运算，指数和底数可以是整数、小数或复数。

幂运算符的语法

```
左表达式  **  右表达式
```


> x ** y 表示 x 的 y 次方

示例

```
>>> 5 ** 2     # 表示 5 的平方
25
```

> 幂运算符的优先级高于乘法运算符。

6. 数据类型转换函数

数据类型转换函数主要用于某些对类型有要求的场合，比如字符串的索引必须用整数，如果数值为 3.0 作为索引也是不可以的，所以需要用到类型转换函数转化为 3 才可以使用。

数据类型转换函数

函数	说明
str(obj)	返回 obj 对应的字符串，未给出参数则返回空字符串 ''。
int(数字或字符串)	返回基于一个数字或字符串构建的整数。未给出参数则返回 0。
float(数字或字符串)	返回基于一个数字或字符串构建的浮点数。未给出参数则返回 0.0。
complex(数字或字符串)	返回基于一个数字或字符串构建的复数。未给出参数则返回 0j。

示例

```
# 写程序，输入商品的单价，输入商品的数量，计算出商品的总价格并打印。
price = input('请输入商品单价: ')
count = input('请输入商品数量: ')

price = float(price)  # 将price 绑定的浮点数 转化为小数再交给 price绑定
count = int(count)
total = price * count
```

```
print('商品总价格是:', total)
```

运行结果

```
请输入商品单价： 19.9
请输入商品数量： 3
商品总价格是： 59.699999999999996
```

练习

写一个程序，输入您的年龄

1. 打印出去年您几岁?
2. 打印出明年您几岁?

如:

```
请输入您的年龄： 35
去年您 34 岁。
明天您 36 岁。
```

7. 增强赋值语句

增强赋值语句（又叫复合赋值运算符）。

什么是增强赋值语句

增强赋值语句是将二元运算符和赋值语句结合在一起的运算符。

这些运算符用于对变量进行算术运算后，并将结果重新赋值给该变量。

常用的增强赋值运算符：

```
+=   -=   *=   /=   //=   %= **=
```

增强赋值语句用法(示意)

```
x += y      # 等同于 x = x + y
x -= y      # 等同于 x = x - y
x *= y      # 等同于 x = x * y
x /= y      # 等同于 x = x / y
x //= y     # 等同于 x = x // y
```

```
x %= y      # 等同于 x = x % y
x **= y     # 等同于 x = x ** y
```

语法

```
变量 增强赋值运算符 表达式
```

示例

```
x, y = 1, 2
x += 100  # 等同于 x = x + 100
y *= 2    # 等同于 y = y * 2
print(x)
print(y)
```

运行结果

```
101
4
```

练习

写一个程序

1. 输入一个长方形的长边长度。
2. 输入一个长方形的短边长度。

打印出长方形的周长和面积，如:

```
请输入长边长度：6
请输入短边长度：4
长方形的周长是：20
长方形的面积是：24
```

第五章、布尔类型

什么是布尔类型

布尔类型（Boolean type）是一种用于表示真（True）或假（False）的逻辑数据类型。

布尔类型主要用于条件判断、循环控制以及逻辑运算等场景。

例如：

今晚魏老师请大家吃大餐，你去不去？

未成年人禁止入内，你能进去吗？

1. 布尔值

布尔值只用两种

- True：表示真（逻辑上的肯定、成立）。
- False：表示假（逻辑上的否定、不成立）。

True 和 False 都是关键字（不能用作普通的标识符）

在计算机的内部，True 用整数 1 表示，False用整数 0 表示。

2. 比较运算

运算符

```
<     # 小于
>     # 大于
<=    # 小于等于
>     # 大于等于
==    # 等于
!=    # 不等于
```

语法

```
左表达式      比较运算符      右表达式
```

python内建数据类型的比较通常返回布尔值。

比较运算进行“值比较”。

示例

```
>>> 1 + 2 < 3 + 4
True
>>> 5 >= 3 + 4
False
>>> 1 + 2 == 3 + 4
False
>>> 1 + 2 != 3 + 4
True
```

3. 布尔运算

什么是布尔运算

布尔运算又叫逻辑运算，是一种基于布尔值（True或False）进行的运算，它们用于根据条件的逻辑关系来评估表达式。

运算符

```
and       # 逻辑与运算
or        # 逻辑或运算
not       # 逻辑非运算
```

布尔运算符的优先级低于比较运算符的优先级。

> and、or、not 是 python 的关键字。

3.1 逻辑与运算

逻辑与运算

当且仅当所有条件表达式都为True时，整个表达式的结果才为True。 如果任何一个条件表达式为False，则整个表达式的结果为False。

语法

```
x and y
```

> x, y代表表达式。

示例

```
>>> 3 + 4 > 5 and 6 + 7 > 100
False
```

逻辑与运算真值表

x 的值	y 的值	x and y 的值
True	True	True
True	False	False
False	True	False
False	False	False

3.2 逻辑或运算

逻辑或运算

- 只要有一个条件表达式为True，整个表达式的结果就为True。
- 只有当所有条件表达式都为False时，整个表达式的结果才为False。

语法

```
x or y   # x, y代表表达式
```

示例

```
>>> 3 + 4 > 5 or 6 + 7 > 100
True
```

逻辑或运算真值表

x的值	y的值	x or y 的值
True	True	True
True	False	True
False	True	True
False	False	False

3.3 逻辑非运算

逻辑或运算

使用not运算符对一个条件表达式进行逻辑取非操作。

- 如果条件表达式为True，则not运算符将其变为False；
- 如果条件表达式为False，则not运算符将其变为True。

语法

```
not x     # x 代表表达式
```

示例

```
>>> not True
False
>>> not False
True
```

4. 条件表达式

表达式的概念

什么是表达式

表达式（expression）是由运算符、变量、字面值以及函数调用等组成的代码片段，它可以计算并返回一个值。

如：

```
'Python'
100
```

```
len('weimingze') - 2
len('I love python') - 2 > 10
```

条件表达式

语法

```
表达式x if 表达式C else 表达式y
```

作用

表达式 `x if C else y` 首先是对条件 C 求值。 如果 C 为真，x 将被求值并返回其值；否则将对 y 求值并返回其值。

语法说明

- `if`、`else` 是 python 关键字。
- 条件表达式具有最低的优先级。
- 条件表达式，C语言称为“三目运算符”，即: `?:`运算。

示例

```
# 写一个程序，让用户输入驾照科目一的考试成绩，
# 如果大约等于九十则提示为"及格"，否则提示"不及格"
score = int(input('请输入成绩: '))
result = '及格' if score >= 90 else '不及格'
print('结果:', result)
```

运行结果1

```
请输入成绩: 98
结果: 及格
```

运行结果2

```
请输入成绩: 80
结果: 不及格
```

练习

写一个程序

1. 让用户输入语文成绩；
2. 让用户输入数学成绩。

如果用户的语文成绩和数学成绩都大于等于80分，则提示考试“合格”。否则提示“不合格”。

5. 布尔转换函数 bool()

python语言中，任意的对象都能表示真（True）或者假（False）状态。

示例

```
>>> "真" if 666 else "假"
'真'
>>> "真" if 0.0 else "假"
'假'
```

内置函数 bool() 可将任意值转换为布尔值。

调用格式

```
bool(object=False)
```

说明

1. 如果不给出参数，则返回False。
2. 布尔运算（and、or、not）和 条件表达式等表达式通常隐式调用此函数来获取布尔状态。

假值对象

```
None    # 空值对象
False   # 布尔类型的假值
0       # 整数的0
0.0     # 浮点数的0
''      # 字符串的空字符串
[]      # 空列表
{}      # 空字典
...
```

python 中每一种预置数据类型都有一种假值对象，假值对象通常是表示不存在或值为零的对象。

6. 短路求值

什么是短路求值

- 短路求值（short-circuit evaluation）是一种逻辑运算的优化方式，用于在评估布尔表达式时尽早确定结果，从而避免不必要的计算。也就是说在运算到一半的时候就已经确定了结果的情况下，有的表达式可能不会被执行。这种现象叫短路求值。
- 短路求值主要应用在逻辑与（and）和逻辑或（or）运算符中。

and 逻辑与运算

优先返回“假”值对象。

or 逻辑或运算

优先返回“真”值对象。

条件表达式

“表达式x if 表达式C else 表达式y ”中表达式x和y只有一个运行。

示例

```
>>> False and print('我不会被调用')
False
>>> True or  print('我也不会被调用')
True
>>> print('我真不会被调用') if False else print('你好! 我会被调用')
你好! 我会被调用
>>>
```

7. 运算符优先级

什么是运算符优先级 运算符优先级（Operator Precedence）决定了在一个表达式中哪个运算符先被执行。

当表达式中有多个运算符时，优先级高的运算符会先进行计算。

示例

```
x = 1 + 2 * 3 ** 4 - 2
x = 1 + 2 * 81 - 2
```

```
x = 1 + 162 - 2
x = 163 - 2
x = 161
```

> 以上示意一个赋值语句右侧表达式的执行顺序。

python 3.13 的运算符优先级表

运算符	描述
`(expressions...)`,`[expressions...]`, `{key: value...}`, `{expressions...}`	绑定或加圆括号的表达式，列表显示，字典显示，集合显示
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	抽取，切片，调用，属性引用
`await x`	await 表达式
`**`	乘方 [[5]
`+x`, `-x`, `~x`	正，负，按位非 NOT
`*`, `@`, `/`, `//`, `%`	乘，矩阵乘，除，整除，取余 [[6]
`+`, `-`	加和减
`<<`, `>>`	移位
`&`	按位与 AND
`^`	按位异或 XOR
`|`	按位或 OR
`in`, `not in`, `is`, `is not`, `<`, `<=`, `>`, `>=`, `!=`, `==`	比较运算，包括成员检测和标识号检测
`not x`	布尔逻辑非 NOT
`and`	布尔逻辑与 AND
`or`	布尔逻辑或 OR
`if` -- `else`	条件表达式
`lambda`	lambda 表达式
`:=`	赋值表达式

> 上表中优先级自上而下依次降低。

第六章、分支语句

什么是语句

语句是一个完整的执行单位，就像你说的一句话一样，你完整的说出来一句话，别人也能按这句话去做事情，你说不完整或有错，别人也无法按你说的做。

什么是简单语句

一个逻辑行就能搞定的语句，不用组合。

什么是复合语句

语句内部需要有其他语句组合的语句是复合语句，通常复合语句有缩进代表包含关系。

- 复合语句会以某种方式影响或控制所包含的语句的执行。
- 通常，复合语句会跨越多行来书写（虽然在某些简单形式下整个复合语句也可能包含于一行之内）。

python的复合语句:

- if 语句
- match 语句
- while 语句
- for 语句
- def 语句
- class 语句
- try 语句(高级内容)
- with 语句(高级内容)

分支语句

什么是分支语句

分支语句是可以根据不同的条件来执行不同的代码块（语句块）的语句。

分支语句允许程序根据条件判断来做出决策，从而实现更复杂和灵活的逻辑。

Python中的分支语句有 if 语句和 match 语句两种。

1. if 语句

作用

让程序根据条件选择性的执行其中的某一个语句块。

完整语法

```
if 条件表达式1:
    语句块1
elif 条件表达式2:
    语句块2
elif 条件表达式3:
    语句块3
...
elif 条件表达式n:
    语句块n
else:
    语句块(其他)
```

语法说明

- if 主语句必须存在。
- elif 子句可以有0个、1个或多个。
- ... 代表 elif 子句可以有多个。
- else 子句只能有一个且只能放在最后。

执行顺序说明

它通过对表达式逐个求值直至找到一个真值在子句体中选择唯一匹配的一个；然后执行该子句体。

最简语法

```
if 条件表达式1:
    语句块1
```

> 本节先来研究最简语法

示例

小学生放学后家长安排的流程

1. 放学
2. 告诉家长考试成绩
3. 玩两个小时手机(考试成绩大于90分)
4. 做作业
5. 睡觉

用if语句表达

```
print("放学")  # 放学

score = int(input("请输入考试成绩: "))  # 告诉家长考试成绩

if score > 90:
    print("玩两个小时手机")  # 玩两个小时手机(考试成绩大于90分)

print("做作业")  # 做作业
print("睡觉")  # 睡觉
```

2. if 语句的 elif 子句

什么是子句（也叫子语句）

子句是指可以跟随主语句增加的语句。

带有 elif 子句的 if 语句的语法

```
if 条件表达式1:
    语句块1
elif 条件表达式2:
    语句块2
elif 条件表达式3:
    语句块3
...
elif 条件表达式n:
    语句块n
```

语法说明

- if 主语句必须存在。
- elif 子句可以有0个、1个或多个；
- ... 代表 elif 子句可以有多个

elif 子句的作用

为 if 语句增加判断条件及对应执行的语句块，以符合更多的逻辑需求。

示例

小学生放学后家长安排的流程2

1. 放学
2. 告诉家长考试成绩
3. 玩两个小时手机(考试成绩大于90分)
4. 玩一个小时手机(考试成绩大于80分，二选一)
5. 做作业
6. 睡觉

```
print("放学")  # 放学

score = int(input("请输入考试成绩: "))  # 告诉家长考试成绩

if score > 90:
    print("玩两个小时手机")  # 玩两个小时手机(考试成绩大于90分)
elif score > 80:
    print("玩一个小时手机")  # 玩一个小时手机(考试成绩大于80分，二选一)

print("做作业")  # 做作业
print("睡觉")  # 睡觉
```

练习

写一个程序，输入一个整数，用程序判断这个整数是正整数，负整数，还是零。

3. if 语句的 else 子句

else 子句的作用

用于处理if 语句中上述条件表达式都不成立的情况下需要执行的语句块。

完整语法

```
if 条件表达式1:
    语句块1
elif 条件表达式2:
    语句块2
elif 条件表达式3:
```

```
    语句块3
...
elif 条件表达式n:
    语句块n
else:
    语句块(其他)
```

语法说明

- if 主语句必须存在。
- elif 子句可以有0个、1个或多个。
- ... 代表 elif 子句可以有多个。
- else 子句只能有一个且只能放在最后。

课上练习

写一个程序，输入一个整数，用程序判断这个整数是正整数，负整数，还是零。

```
number = int(input("请输入一个整数: "))

if number > 0:
    print(number, "是正整数! ")
elif number <0:
    print(number, "是负整数")
else:  # elif number == 0:
    print(number, "是零!")
```

练习

写一个程序，输入一个整数，用程序判断这个奇数还是偶数，并打印结果。

说明

1. 偶数是能被2整除的数，比如：0、2、4、6、8、10 ……；
2. 奇数是不能被2整除的数，比如：1、3、5、7、9 ……；

4. BMI 指数综合练习

综合练习

BMI指数(Body Mass Index) 又称 身体质量指数, 是由身高和体重计算得到；这个值的大小可以体现出身体的体重状况。

BMI值计算公式:

```
BMI = 体重(公斤) / 身高的平方(米)
```

例如:

一个69公斤的人，身高是173厘米，计算他的 BMI 值：

```
BMI = 69 / 1.73 ** 2 = 23.05
```

标准表:

BMI值	身体状况
BMI < 18.5	体重过轻
18.5 <= BMI < 24	正常范围
BMI > 24	体重过重(超标)

要求

输入身高和体重，打印BMI的值，并打印体重状况。

参考代码

```
# 步骤分析
# 1. 读取体重数据
weight = float(input('请输入体重（公斤）：'))

# 2. 读取身高数据
height = float(input('请输入身高（米）：'))

# 3. 计算 BMI 指数并用变量绑定（记住）
bmi = weight / height ** 2

print("BMI = ", bmi)

# 4. 使用 if 语句判断区间并给出结果
if bmi < 18.5:
    print("体重过轻! ")
elif 18.5 <= bmi <= 24:
    print('标准体重! ')
elif bmi > 24:
    print('体重过重!')
```

5. if 语句嵌套

什么是语句

语句（Statements）是 python 语句执行的基本单元。

汉语语句示例：

```
你去把 PyCharm 打开。
```

> 上面这个一条完整的语句，根据语义别人能完整的做事情。

不是语句的示例

```
你去把
```

> 话没说完，无法无法按着你的意思做任何事。所以不是语句。

Python 语句的分类

- 简单语句
  - 表达式语句，如: `print('python')`
  - 赋值语句，如: `x = 1 + 2`

- 复合语句
  - if 语句

```
if score > 100:
    print('成绩错误!')
```

if 语句也是语句，他可以嵌套到其他的复合语句中。

示例

```
if x > y:
     if y > 0:
          print('aaaa')
     else:
```

```
            print('bbbb')
else:
    print('cccc')
```

if 语句嵌套示例

北京国展车展招聘模特，初步筛选条件为：

- 男，身高在185cm以上（包含185cm)。
- 女，身高在175cm以上（包含175cm)。

写一个程序，输入性别和身高cm，打印出此人是否满足招聘条件。

实现代码

```
gender = input('请输入性别(男/女): ')
height = int(input('请输入身高(厘米): '))

if gender == '男':
    if height >= 185:
        print('满足条件!')
    else:
        print('不满足条件!')
elif gender == '女':
    if height >= 175:
        print('满足条件!')
    else:
        print('不满足条件!')
```

6. match 语句

> 注：match 语句从 python3.10 起用，之前的版本不支持。

作用

match语句是根据一个表达式的值和模式匹配来执行不同的代码块。

match 语句类似于 C 语言中的 switch 语句的功能，但有更强的表达能力和灵活性。

语法

```
match 目标表达式:
    case 匹配模式1:
        语句块1
    case 匹配模式2:
```

```
        语句块2
    ...
    case 匹配模式n:
        语句块n
```

> ... 代表 case 子句可以由有0个、1个或多个。

match 语句的各种匹配模式

1. 字面值模式

match语句根据一个表达式的值来自上而下依次对比各个case块中的字面值，如果表达式与字面值相等，则执行case对应的语句块。

match 语句字面值模式同C语言中的switch语句的功能完全一样。

字面值模式语法：

```
match 目标表达式:
    case 字面值1:
        语句块1
    case 字面值2:
        语句块2
    case 字面值n:
        语句块n
```

示例

写一个程序，输入“春”、“夏”、“秋”、“冬”四个季节中的任意一个，打印出这个季节在哪几个月。

```
season = input('请输入季节(春/夏/秋/冬):')
match season:
    case '春':
        print("春季在 1~3 月!")
    case '夏':
        print("夏季在 4~6 月!")
    case '秋':
        print("秋季在 7~9 月!")
    case '冬':
        print("冬季在 10~12 月!")
```

2. 通配符模式

"_" 是通配符，它可以匹配任何值以及任何模式。

case _ : 通常放在最后，并总会匹配成功。

通配符模式语法：

```
match 目标表达式:
    case 字面值1:
        语句块1
    case 字面值2:
        语句块2
    case 字面值n:
        语句块n
    case _:
        语句块other
```

说明

"_" 是软关键字，只用在match语句中有效。

示例

改写上面的示例，实现如下功能:

写一个程序，输入“春”、“夏”、“秋”、“冬”四个季节中的任意一个，打印出这个季节在哪几个月。如果用户输入的不是上述四个季节则提示“您的输入有误”!

```
season = input('请输入季节(春/夏/秋/冬):')
match season:
    case '春':
        print("春季在 1~3 月!")
    case '夏':
        print("夏季在 4~6 月!")
    case '秋':
        print("秋季在 7~9 月!")
    case '冬':
        print("冬季在 10~12 月!")
    case _:
        print('您的输入有误!')
```

> 与第一个示例相比，增加了一个 `case _:` 子句，这个 `_` 就是通配符，能匹配所有其他值!

其他匹配模式

match 语句有 10 种匹配模式，详见官网文档。

https://docs.python.org/zh-cn/3/reference/compound_stmts.html#patterns

7. match 语句约束项

约束项

- [if 表达式] 是约束项。
- 只有在约束项表达式为真时才会执行此case块，否则跳过此case块。

约束项语法：

```
match 目标表达式:
    case 匹配模式1 [if 表达式1]:
        语句块1
    case 匹配模式2 [if 表达式2]:
        语句块2
    case 匹配模式n [if 表达式n]:
        语句块n
    case _:
        语句块other
```

> 其中 case 子句后面的 [] 内的内容是可选项，这个 `if 表达式` 称为约束项。

示例

北京国展车展招聘模特，初步筛选条件为：

- 男，身高在185cm以上（包含185cm)。
- 女，身高在175cm以上（包含175cm)。

写一个程序，输入性别和身高cm，打印出此人是否满足招聘条件。

```
# match 语句 -- 约束项 示例

gender = input('请输入性别(男/女): ')
height = int(input('请输入身高(厘米): '))

match gender:
    case '男' if height >= 185:
        print('满足条件!')
    case '女' if height >= 175:
        print('满足条件!')
```

```
    case _:
        print('不满足条件!')
```

练习

编写北京出租车计价器程序：

```
收费标准：
3公里以内收费13元
基本单价 2.3元/公里（超出3公里以外）
空驶费：超过15公里后，每公里加收单价的50%空驶费(3.45元/公里)
```

要求，输入公里数，打印出费用金额(以元为单位，元以下的四舍五入)。

第七章、循环语句

问题

写一个程序，打印 10000 行 `hello world`。

```
print('hello world!')
```

如何做到让上述语句执行多次呢？此时需要用到循环语句。

循环语句

循环语句用于重复执行一段代码块，直到满足某个条件为止。

循环语句的种类：

- while 语句
- for 语句

循环相关的语句：

- continue 语句
- break 语句

1. while 语句

作用

用于在表达式保持为真的情况下重复地执行语句块。

语法

```
while 条件表达式:
    语句块1  (此部分可能会重复执行)
else
    语句块2
```

说明

- 先执行条件表达式，测试布尔值是否为True或False。
  - 如果条件表达式测试值为True 则执行语句块1，然后再返回到条件表达式，重复进行测试；
  - 如果条件表达式测试值为False，则执行else子语句中的语句块2，然后结束此while语句，如果没有else子句则直接结束此while语句的执行。
- else 子句可以省略。
- else 子句仅在条件表达式为假时才会执行。

示例

写程序，使用 while 语句打印 5 行 `hello world`

```
times = 1
while times <= 5:
    print("hello world")
    times += 1   # times = times + 1
else:
    print('while 语句结束: times=', times)
```

练习

写一个程序，输入一个整数n，写程序打印如下 n 行文字。

如:

```
请输入: 10
这是第 1 行
这是第 2 行
这是第 3 行
...
这是第 10 行
```

2. pycharm 调试运行

学习 Python 课的各位学者学到此一定是历尽磨难，出了不少的错误；那该怎么判断错误并改正错误呢？那本节将使用一下 PyCharm 的 Debug 功能解决这一问题。

错误的种类

1. 语法错误：
   - 在python程序运行时终止程序并报错！这种错误之间看终端的报错信息就能定位在那个文件的那一行可能是什么错误，都有提示信息，相对好解决。
2. 逻辑错误：
   - 在python程序运行时不报错，但运行逻辑和结果不符合预期！这种错误最痛苦，也是解决起来最耗时的错误。这种错误最好借助于单步调试软件来完成。比如 PyCharm。

PyCharm调试运行的步骤:

1. 设置断点；
2. 调试运行程序；
3. 单步执行，观察程序运行时的变量和步骤是否符合预期，并找出问题。

什么是断点

断点（Breakpoint）是在程序调试运行中的一种调试工具，用于在代码执行过程中暂停执行，以便开发者可以检查当前各个变量的状态。

开发者可以逐步跟踪代码的执行路径，诊断程序中的错误或行为异常。

白话文解释

断点就犹如马路上的红绿灯，当车子遇到红灯时就会停在红灯的前面不能继续前行，断点也是如此。在调试运行时，程序在断点处停止且不运行断点所处的代码，等待操作命令。

常用的操作命令

1.  设置和取消断点
   - 点击每一行的行首就可以设置和取消断点。
2.  调试运行
   - 点击你正在编写的python程序文件，在右键菜单中选择 `调试 文件名` 或 `debug 文件名`。
3.  停止调试运行
   - 停止调试运行。
4.  执行当前行，停到下一行
   - 当程序运行到断点停止后，点击此按就可以向前走一步。

5.  进入函数内部
   - 在调用函数时进入到函数的内部执行。
6.  跳出函数返回到调用处
   - 快速执行完此函数，返回到调用此函数的入口处。
7.  运行到下一断点
   - 在程序调试运行中时，可以一直运行，知道再次遇到断点时停止。

练习

写程序，让用户输入一个整数n，让程序计算n的阶乘(n!) 并打印。

n 必须是大于等于零的整数!

阶乘的概念

```
0! = 1
1! = 1
2! = 1 x 2 = 2
3! = 1 x 2 x 3 = 6
4! = 1 x 2 x 3 x 4 = 24
5! = 1 x 2 x 3 x 4 x 5 = 120
```

> 建议使用 while 语句实现并进行调试练习。

3. for 语句

for 语句 又叫 迭代循环语句，是针对可迭代对象内部的数据进行遍历的语句。

术语

可迭代对象:

- 能够依次获取数据元素的对象。

遍历:

- 经历且只经历一遍。

Python预置的可迭代对象有：

1. 字符串；
2. range() 函数调用后返回的对象；
3. 列表；
4. 元组；
5. 字典；
6. 集合；
7. 固定集合；
8. 字节串等。

白话文解释

你家有一盒抽纸，你在用的时候就每次抽取一张，那这个抽纸就是可迭代对象，你每次抽取一张的行为就是遍历。

抽纸抽空，遍历结束。

作用

用于对序列（例如字符串、元组或列表）或其他可迭代对象中的元素进行迭代。

语法：

```
for 变量列表 in 可迭代对象:
    语句块1
else:
    语句块2
```

语法说明：

- 可迭代对象每次提供一个元素赋值给变量列表中的变量，赋值完毕后执行语句块1，重复执行此步骤直到可迭代对象不能提供数据为止。
- 可迭代对象迭代完所有的元素后，执行else子句部分语句块2，然后退出循环。
- else 子句可以省略。
- else 子句的 语句块2 只有在可迭代对象不再能提供数据的时候才会执行。
- 当在循环内部用 break 终止循环时，else 子句部分语句不会执行.（后面讲 break 语句）。

示例

```
word = "Python"
```

```
for achar in word:
    print("achar:", achar)
else:
    print('迭代循环结束了')
```

练习

写一个程序，读取用户输入的英文字符串，并计算出英文中出现'a'、'e'、'i'、'o'、'u'的总次数平打印出来。

如:

运行效果如下:

```
请输入文字: welcome to china!
'aeiou'的总出现次数是: 5
```

4. range 函数

作用

调用后，返回一个能够得到一系列整数的可迭代对象。

调用格式

```
range(stop)               # stop 停止整数
range(start, stop)        # start 开始整数
range(start, stop, step)  # step 步长
```

说明

- 省略 step 参数，则默认为 1。
- 省略 start 参数，则默认为 0。

示例1

```
range(5)          # 生成 0、1、2、3、4
range(3, 6)       # 生成 3、4、5
range(1, 10, 3)   # 生成 1、4、7
range(10, 0, -2)  # 生成 10、8、6、4、2
```

示例2

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

示例3

写一个程序，输入一个整数n，写程序打印如下 n 行文字 如:

```
请输入: 10
这是第 1 行
这是第 2 行
这是第 3 行
...
这是第 10 行
```

代码

```
line_count = int(input('请输入: '))

for line_num in range(1, line_count+1):
    print('这是第', line_num, '行')
```

练习

思考下列程序打印多少次 hello?

```
for x in range(5):
    for y in range(10):
        print("hello")
```

5. continue 语句

作用

用于循环语句(while 语句和for语句)中， 不再执行本次循环内 continue 之后的语句，开始一次新的循环。

语法

```
continue
```

说明

- 在for 语句中，执行 continue 语句，for语句将会从可迭代对象中获取下一个元素绑定变量后再次进行循环。
- 在while 中，执行continue 语句，将会直接跳转到while 语句的真值表达式处，重新判断循环条件。

示例

打印 1 ~ 10 的整数，但是不打印 4 这个数。

for 语句实现：

```
for number in range(1, 11):
    if number == 4:
        continue
    print(number)
```

while 语句实现：

```
numbers = 1
while numbers < 11:
    if numbers == 4:
        numbers += 1
        continue
    print(numbers)
    numbers += 1
```

练习

打印100以内所有的整数，要求跳过有7或能被7整数的正整数。

如：遇到7、14、17、21、27、28、... 70、71、72都要跳过。

6. break 语句

作用

用于循环语句(while, for 语句)中，当 break 执行时，break 会终止包含它的当前循环语句的执行。

语法

```
break
```

说明

- break 语句只能用在 while 语句或 for 语句的内部。
- 当 break 语句执行后，此循环语句 break 之后的所有语句都不会执行（else 子句里的语句也不执行）
- break 语句只能终止包含他的当前循环，当有循环嵌套时，只能跳出离他最近的一个循环

示例

打印 1 ~ 10 的整数，当遇到4时停止打印过程。

```
for number in range(1, 11):
    if number == 4:
        break
    print(number)
else:
    print('打印完成!')

print('程序结束')
```

练习

写一个用户身份验证程序。

假设一个用户信息如下：

- 用户名是：root
- 密码是：123456

写一个身份验证的程序myprog.py，让用户输入用户名和密码登录，用户名和密码全部匹配，提示登录成功。否则继续，最多尝试3次。3次不匹配以后提示登录失败。

7. 死循环

死循环是指循环条件一直成立的循环。

写法:

```
while True:
    语句块
```

- 死循环通常使用 break 语句来终止循环；
- 死循环的 else 子句中的语句永远不会执行；

- 死循环通常用于循环次数无法确定的循环。

示例

写一个程序，输入任意个学生的成绩，当输入负数时结束输入。打印这些学生的总成绩。

```
total_score = 0  # 记录总成绩

while True:
    score = int(input('请输入成绩: '))
    if score < 0:
        break
    total_score += score

print('总成绩是:', total_score)
```

第八章、列表

什么是容器

容器类型（Container Types）是能够存储多个数据的数据类型。容器类型可以帮助你组织、管理和访问存储在其中的数据。

python 中一个变量只能绑定一个数据对象当有大量的数据需要存储时，可以使用 python 的容器类型。

Python预置的容器类型：

- 列表（list）
- 元组（tuple）
- 集合（set）
- 字典（dict）
- ...

序列类型

序列类型是指数据有先后顺序排列的数据类型。

字符串是序列类型，如：

```
'Python'
```

示意图

python中内建的序列类型

- 列表（list）
- 元组（tuple）
- range对象
- 字符串 （str）
- 字节串（bytes）
- 字节数组（bytearray）

说明

序列类型都支持拼接、重复、索引、切片等序列操作。

1. 列表的创建

列表

Python的列表是一种可变数据类型的容器，其内部可以存储任意类型的数据，且存储的数据有先后顺序排列关系。

python 列表容器有如下的特点：

- 列表是一种可以存储任意个各种类型的序列容器；
- 列表内的数据有先后顺序关系；
- 列表是可变的容器；

列表字面值

python 中可以使用中括号 [] 组成的表达式来创建列表。

```
>>> L1 = []      # 创建一个空的列表
>>> L2 = ['北京', '上海', '广州', '西安']
>>> L3 = [1, 'Two', 3.14, True, False, None]
>>> L4 = [1, 2, [3.1, 3.2], 4]   #  含有四个元素的列表
>>> L5 = [
    ['魏明择', 90, 100],  # 姓名，语文成绩，数学成绩
    ['小魏魏', 59, 99]
]
```

> python 中的 (), [], {} 和 三引号字符串都要成对出现，可以隐式换行，即为一个逻辑行。

创建列表的函数 list

```
list()          # 创建一个空的列表，等同于 []
list(可迭代对象)  # 用可迭代对象创建一个列表
```

示例

```
L1 = list()          # L1 = []
L2 = list("ABC")     # L2 = ['A', 'B', 'C']
L3 = list(range(5))  # L3 = [0, 1, 2, 3, 4]
```

2. 列表的运算

什么是序列类型 序列类型是指数据有先后顺序排列的数据类型。Python 的列表也是序列类型。

序列有共同的特点，也有共同的操作。

Python 中的序列类型如下：

- 序列类型 --- list, tuple, range
- 文本序列类型 --- str
- 二进制序列类型 --- bytes, bytearray, memoryview

序列类型的数据都支持通用序列操作

通用序列操作

运算	结果
s + t	s 与 t 相拼接
s * n 或 n * s	相当于 s 与自身进行 n 次拼接
s[i]	s 的第 i 项，起始为 0
s[i:j]	s 从 i 到 j 的切片
s[i:j:k]	s 从 i 到 j 步长为 k 的切片
len(s)	s 的长度
min(s)	s 的最小项
max(s)	s 的最大项
s.index(x[, i[, j]])	x 在 s 中首次出现项的索引号（索引号在 i 或其后且在 j 之前）
s.count(x)	x 在 s 中出现的总次数
x in s	如果 s 中的某项等于 x 则结果为 True，否则为 False
x not in s	如果 s 中的某项等于 x 则结果为 False，否则为 True

示例

```
>>> list1 = [11, 22, 33]
>>> list2 = [44, 55, 66]
>>> list1 + list2
[11, 22, 33, 44, 55, 66]
>>> list1 * 2
[11, 22, 33, 11, 22, 33]
>>> list1[1]
22
>>> list1[-1]
33
>>> list1[1:2]
[22]
>>> list1[1:]
[22, 33]
>>> list1[::2]
[11, 33]
>>> 22 in list1
True
>>> 99 in list1
False
>>> 99 not in list1
True
>>>
```

通用序列函数

运算	结果：
len(s)	s 的长度
min(s)	s 的最小项
max(s)	s 的最大项

示例

```
>>> s = [1, 2, 3, 4, 2, 2, 3, 4]
>>> len(s)
8
>>> min(s)
1
>>> max(s)
4
```

通用序列方法

运算	结果：
s.index(x[, i[, j]])	x 在 s 中首次出现项的索引号（索引号在 i 或其后且在 j 之前）
s.count(x)	x 在 s 中出现的总次数

示例

```
>>> s = [1, 2, 3, 4, 2, 2, 3, 4]
>>> s.index(3)
2
>>> s.count(2)
3
```

3. 成员检测运算

in 和 not in 用于判断一个数据元素值是否在于可迭代对象中。

语法

```
表达式 in 可迭代对象
# 或
表达式 not in 可迭代对象
```

说明

运算	结果
x in s	如果 s 中的某项等于 x 则结果为 True，否则为 False
x not in s	如果 s 中的某项等于 x 则结果为 False，否则为 True

示例

```
# 此示例示意成员检测运算的用法

print('国人' in "中国人")  # True
print('田' in '中国人' )  # False
print('国人' not in "中国人")  # 等同于 not ('国人' in "中国人")
print('田' not in '中国人' )  # True
```

```
print(200 in [100, 200, 300])  # True
print(500 in [100, 200, 300])  # False
print(1 not in [100, 200, 300])  # True
print(100 not in [100, 200, 300])  # False
```

4. 列表的添加数据运算

可变数据类型

可变数据类型是指创建后还可以修改其内容的数据类型。

- 可变数据类型有：
  - 列表list、字典dict、集合set、字节数组bytearray、内存映射视图memoryview。
- 不可变的数据类型有：
  - 数字（整数int、浮点数float、复数complex）、字符串str、布尔类型bool、空值对象None。

添加数据的运算符

+= 运算符可以追加多个数据元素。

语法

```
列表 += 可迭代对象
```

说明

将可迭代对象的数据依次追加到列表的末尾。

示例

```
>>> lst = [11, 22, 33]
>>> lst += range(44, 70, 11)
>>> lst
[11, 22, 33, 44, 55, 66]
>>> lst += 'ABC'
>>> lst
[11, 22, 33, 44, 55, 66, 'A', 'B', 'C']
>>>
```

添加数据的方法

列表的方法名	说明
list.append(x)	向列表的末尾追加单个数据
list.insert(index, obj)	将某个数据obj 插入到 index这个索引位置的数据之前
list.extend(可迭代对象)	等同于: L += 可迭代对象

示例

```
>>> lst = [11, 22, 33]
>>> lst.append(44)
>>> lst
[11, 22, 33, 44]
>>> lst.insert(0, -1)
>>> lst
[-1, 11, 22, 33, 44]
>>> lst.extend(range(55, 80, 11))
>>> lst
[-1, 11, 22, 33, 44, 55, 66, 77]
>>>
```

练习

写程序，输入一系列学生的成绩，当输入负数时结束输入。

1. 打印出学生人数?
2. 打印出本次考试的平均成绩?
3. 打印出本次考试的最高分是多少?

5. del 语句 删除数据

del 语句

作用

- 删除变量，同时解除目标的绑定。
- 删除列表中指定位置的数据元素。

语法

```
del 变量1,变量2  # 删除变量，同时解除变量绑定的目标
del 列表[i]      # 删除列表内 i 对应的数据
```

```
del 列表[i:j]   # 删除列表内切片i:j 对应的数据
del 列表[i:j:k] # 删除列表内切片i:j:k 对应的数据
```

示例

```
>>> lst = [11, 22, 33, 44, 22, 22, 33, 44]
>>> del lst[0]
>>> lst
[22, 33, 44, 22, 22, 33, 44]
>>> del lst[3:5]
>>> lst
[22, 33, 44, 33, 44]
>>> del lst[::2]
>>> lst
[33, 33]
>>> del lst
>>> lst
Traceback (most recent call last):
  File "<python-input-10>", line 1, in <module>
    lst
NameError: name 'lst' is not defined. Did you mean: 'list'?
>>>
```

删除数据的方法

方法	意义
list.remove(x)	从列表中删除第一次出现在列表中的值
list.pop([index])	删除索引对应的元素，如果不加索引，默认删除最后元素，同时返回删除元素的引用关系
list.clear()	清空列表，等同于 L[:] = []

示例

```
>>> lst = [11, 22, 33, 44, 22, 22, 33, 44]
>>> lst.remove(44)
>>> lst
[11, 22, 33, 22, 22, 33, 44]
>>> lst.pop(-1)
44
>>> lst
[11, 22, 33, 22, 22, 33]
>>> lst.clear()
>>> lst
[]
>>>
```

练习

写程序，输入一系列学生的成绩，当输入负数时结束输入。

删除成绩小于60 和大于 100 的学生的成绩。

打印出剩余学生的平均成绩?

6. 列表的修改数据运算

修改数据

用索引赋值和切片赋值可以改变列表内的数据。

语法

```
列表[i] = 表达式
列表[i:j] = 可迭代对象
列表[i:j:k] = 可迭代对象
```

> i、j、k 表示整数。

说明

对于步长不等于1的切片赋值，赋值运算符的右侧的可迭代对象提供元素的个数一定要等于切片切出的段数。

示例

```
>>> lst = ['A', '2.0001', 'C', "444", "555", 6, 'G']
>>> lst = ['A', '2.0001', 'C', "444", "555", 6, 'G']
>>> lst[1] = 2
>>> lst
['A', 2, 'C', '444', '555', 6, 'G']
>>> lst[3:5] = [4, '五']
>>> lst
['A', 2, 'C', 4, '五', 6, 'G']
>>> lst[::2]
['A', 'C', '五', 'G']
>>> lst[::2] = range(1, 8, 2)
>>> lst
[1, 2, 3, 4, 5, 6, 7]
>>>
```

练习

写程序，输入一系列学生的成绩，当输入负数时结束输入。

1. 将学生成绩小于60 和大于 100 的学生的成绩都改为0分。
2. 打印出0分学生的人数。
3. 打印出不是0分学生的人数。

7. 列表的方法

列表的方法

方法	意义
list.index(v [, begin[, end]])	返回对应元素的索引下标, begin为开始索引，end为结束索引，当 value 不存在时引发ValueError错误
list.count(x)	返回列表中元素的个数
list.append(x)	向列表中追加单个元素
list.insert(index, obj)	将某个元素插放到列表中指定的位置
list.extend(lst)	向列表追加另一个列表
list.remove(x)	从列表中删除第一次出现在列表中的值
list.pop([index])	删除索引对应的元素，如果不加索引，默认删除最后元素，同时返回删除元素的引用关系
list.clear()	清空列表，等同于 L[:] = []
list.copy()	复制此列表（只复制一层，不会复制深层对象)
list.sort(reverse=False)	将列表中的元素进行排序，默认顺序按值的小到大的顺序排列
list.reverse()	列表的反转，用来改变原列表的先后顺序

示例

```
>>> lst = [9, 1, 7, 3, 5]
>>> lst2 = lst1.copy()
>>> lst1 = [9, 1, 7, 3, 5]
>>> lst2 = lst1.copy()
>>> lst1.clear()
```

```
>>> lst1
[]
>>> lst2
[9, 1, 7, 3, 5]
>>> lst2.sort()
>>> lst2
[1, 3, 5, 7, 9]
>>> lst2.reverse()
>>> lst2
[9, 7, 5, 3, 1]
>>>
```

8. 列表推导式

列表推导式

列表推导式（List Comprehension）又叫列表解析式，是用一个可迭代对象（如列表、元组、集合或字符串）来创建新的列表的方式。

它允许你使用简洁的表达式来生成列表，而不需要编写完整的循环语句。

语法

```
[ 表达式 for 变量 in 可迭代对象 ]
# 或
[ 表达式 for 变量 in 可迭代对象 if 真值表达式 ]
```

示例

使用 for 语句创建列表：

```
s1 = []
for x in range(1, 10):
    s1.append(x ** 2)

print("s1:", s1)
```

使用 **列表推导式** 创建同样的列表：

```
s2 = [x ** 2 for x in range(1, 10)]
print('s2:', s2)
```

示例2

```
# 以下生成一个数值为1~9的平方的列表,结果的个位数为4的跳过
# 结果: s1 = [1, 9, 16, 25, 36, 49, 81]
s3 = [x ** 2 for x in range(1, 10) if x ** 2 % 10 != 4]
print('s3:', s3)
```

列表推导式嵌套

语法

```
[ 表达式1
    for 变量1 in 可迭代对象1 if 真值表达式1
        for 变量2 in 可迭代对象2 if 真值表达式2 ]
```

示例

生成如下面列表

['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']

```
s4 = [x + y for x in "ABC" for y in "123"]

print('s4:', s4)
```

第九章、元组

什么是元组

元组是不可改变的列表，同列表 list 一样，元组可以存放任意类型的数据，但是一旦创建将不可修改。

元组的特点

- 不可变性：元组一旦创建，其内容就不能被修改。
- 性能：由于元组是不可变的，因此它们在执行某些操作时可能会比列表更高效且安全。

1. 元组的创建

创建元组的字面值

用小括号() 括起来，单个元素括起来后加逗号来区分单个元素还是元组。

示例

```
t1 = ()          # 空元组
t2 = (100,)      # 含有一个数据元素的元组
t3 = 100,        # 含有一个数据元素的元组
t4 = (1, 2, 3)   # 含有三个数据元素的元组
t5 = 1, 2, 3     # 含有三个数据元素的元组
```

> 注： type(x) 函数用来返回数据x的类型。

创建元组的函数tuple

函数 tuple 的调用语法：

```
tuple()            # 创建一个空的元组，等同于 ()
tuple(可迭代对象)    # 用可迭代对象创建一个元组
```

示例

```
L1 = tuple()          # L1 = ()
L2 = tuple("ABC")     # L2 = ('A', 'B', 'C')
L3 = tuple(range(5))  # L3 = (0, 1, 2, 3, 4)
```

元组的用途

元组常用于存储不可变的数据集合，例如函数的返回值、数据库中的记录等。

2. 元组的运算

元组是序列，他和列表一样支持通用序列操作。

元组是不可变对象，它没有添加数据，删除数据和修改数据的操作方法。

通用序列操作

运算	结果：
x in s	如果 s 中的某项等于 x 则结果为 True，否则为 False
x not in s	如果 s 中的某项等于 x 则结果为 False，否则为 True
s + t	s 与 t 相拼接
s * n 或 n * s	相当于 s 与自身进行 n 次拼接
s[i]	s 的第 i 项，起始为 0
s[i:j]	s 从 i 到 j 的切片
s[i:j:k]	s 从 i 到 j 步长为 k 的切片
len(s)	s 的长度
min(s)	s 的最小项
max(s)	s 的最大项
s.index(x[, i[, j]])	x 在 s 中首次出现项的索引号（索引号在 i 或其后且在 j 之前）
s.count(x)	x 在 s 中出现的总次数

3. 用于序列的内置函数

用于序列的内置函数：

运算	结果
len(s)	返回对象s的长度（元素个数）
min(iterable)	返回可迭代对象中最小的元素，或者返回两个及以上实参中最小的。
max(iterable)	返回可迭代对象中最大的元素，或者返回两个及以上实参中最大的。
sum(iterable)	对 iterable 的项求和并返回总计值。
any(iterable)	如果 iterable 的任一元素为真值则返回 True。 如果可迭代对象为空，返回 False。
all(iterable)	如果 iterable 的所有元素均为真值（或可迭代对象为空）则返回 True 。

示例

```
>>> t = (200, 100, 0, 400, 300)
>>> len(t)
5
>>> min(t)
0
>>> max(t)
400
>>> sum(t)
1000
>>> any(t)
True
>>> all(t)
False
>>>
```

第十章、字典

什么是字典

字典（dictionary）是一种内置的数据结构，用于存储键值对（key-value pairs）。

字典通过键（key）来访问对应的值（value）。

示例

- 一个人
  - 姓名：魏明择
  - 年龄：35
  - 身高：173cm

- 字典表示

```
{
    '姓名': '魏明择',
    '年龄': 35,
    '身高': 173
}
```

1. 字典的创建

创建字典的字面值

字典的表示方式以 `{}` 括起来， 以英文的冒号 (`:`) 分隔键值对，各键值对之间用逗号(`,`)分隔.

示例

```
d = {}  # 创建空字典
d = {'name': "weimingze", "age": 35}
d = {'score': [90, 88, 100]}
d = {1:'壹', 2:'贰', 5:'伍'}
```

创建字典的函数dict

字典的构造(创建)函数dict。

函数	说明
dict()	# 生成一个空的字典 等同于 {}
dict(iterable)	用可迭代对象初始化一个字典
dict(**kwargs)	关键字传参形式生成一个字典
dict(mapping)	(key, value) 对形式的初始化

使用 dict 函数可以创建字典。

```
d = dict()   # d = {}
# 创建字典: {'name': '魏明择', 'age': 35}
d = dict([("name", "魏明择"), ("age", 35)])

# 创建字典: {'a':1, 'b':2, 'c':3}
d = dict(a=1, b=2, c=3)
```

字典说明

- 字典是一种可变的容器，可以存储任意类型的数据；
- 字典的键不能重复，且只能用不可变类型作为字典的键；
- 字典的键必须是不可变类型。

以下4种类型不可以作为字典的键

- 列表（list）
- 字典（dict）
- 集合（set）
- 字节数组（bytearray）

示例

```
# 此示例示意 字典 创建的语法

d1 = {}  # 空字典
d2 = {'name': '魏明择', 'age': 35}  # 含有两个键值对的字典

d3 = dict()  # {}
d4 = dict(name='魏明择', height=173)  # {'name':'魏明择', 'height':173}
d5 = dict([['姓名', '小张'],['年龄', 20]])  # {'姓名':'小张', '年龄':20}

print(d1, d2)
print(d3, d4, d5)
```

2. 字典-内容访问

字典的键索引

用键索引可以查看字典中的数据。

语法

```
字典[键表达式]
```

示例

```
d = {'one': 1, 'two': 2}
value = d['two']
print(value)
```

字典是可迭代对象

字典是可迭代对象，字典只能对所有的键进行迭代访问。

示例

```
>>> d = {'name': 'weimingze', 'age': 35}
>>> for k in d:
...     print(k)
...
name
age
```

in / not in 运算符

作用

- 用 in 可以判断一个键是否存在于字典中，如果存在返回 True，否则返回 False。
- not in 的返回值与 in 相反。

示例

```
>>> d = dict(a=1, b=2)   # d = {'a': 1, 'b': 2}
>>> 'a' in d
True
>>> 1 in d
False
```

```
>>> 'hello' not in d
True
```

3. 字典-添加修改键值对

添加和修改字典的元素

用键索引赋值的方式可以创建或改变键和绑定的值。

语法

```
字典[键表达式 ] = 表达式
```

说明

- 键不存在，会创建键并绑定键对应的值。
- 键存在，会改变键对应的值。

示例

```
d = {}
d['name'] = '魏明择'    # 添加键值对
d['age'] = 35          # 添加键值对
d['age'] = 36          # 改变 'age' 键对应的值
```

4. 字典-删除键值对

删除字典的数据

用del 语句 可以删除字典的键同时解绑键对应的值。

语法

```
del 字典[键表达式]
```

示例

```
>>> d = dict(a=1, b=2)   # d = {'a': 1, 'b': 2}
>>> del d['a']
```

```
>>> d
{'b': 2}
```

5. 字典的常用方法

字典的常用方法

字典的方法	说明
dict.clear()	清空字典
dict.copy()	复制字典
dict.get(key[, default])	如果 key 存在于字典中则返回 key 的值，否则返回 default。 如果 default 未给出则默认为 None，因而此方法绝不会引发 KeyError。
dict.pop(key)	移除键，同时返回键对应的值

字典的常用方法示例

```
>>> d = {'name': 'weimz', 'age': 35}
>>>
>>> d['age']
35
>>> d.get('address', '未填写住址')
'未填写住址'
>>> d.get('age', 0)
35
>>> n = d.pop('name')
>>> d
{ 'age': 36}
>>> n
'tarena'
>>> d['address']  # 报错
```

6. 字典推导式

字典推导式

字典推导式是用一个可迭代对象来创建新的字典的方式。

语法

```
{ 键表达式:值表达式
    for 变量 in 可迭代对象
        if 真值表达式 }
```

示例

创建表示等级的字典，内容如下:

```
{
  'level_1': 1, 'level_2': 2, 'level_3': 3, ...,
  'level_100': 100
}
```

参考答案

```
>>> levels = {f'leve_{x}' for x in range(1, 101)}
>>> levels
{'leve_68', 'leve_29', 'leve_100', 'leve_26', ..., 'leve_12', 'leve_62'}
```

> 字典是无序的数据结构，因此数据可能是乱序的。

7. 字典存储综合练习

字典存储综合练习

写一个程序，在程序内部保存一些学生的姓名和成绩信息。

要求:每个学生使用字典存储，格式为：`{'name':'张三', 'score': 69}`

将表示每个学生信息的字典放入列表中统一保存。形成如下数据结构:

```
[{'name':'张三', 'score': 69},
 {'name':'李四', 'score': 100},
... ]
```

1. 循环录入学生的姓名和成绩，当姓名为空时结束输入;
2. 打印此时的列表的内容;
3. 计算学生的平均成绩。

参考答案

```python
# 创建一个列表用来保存数据
students = []

# 1. 循环录入学生的姓名和成绩，当姓名为空时结束输入;
while True:
    name = input('请输入姓名: ')
    if name == '':
        break
    score = int(input('请输入成绩: '))
    a_student = {'name': name, 'score': score}
    students.append(a_student)

# 2. 打印此时的字典的内容;
print(students)

# 3. 计算学生的平均成绩。
total_score = 0
for a_student in students:
    total_score += a_student['score']

avg_score = total_score / len(students)
print('平均成绩是:', avg_score)
```

第十一章、集合

集合的特征

- 集合是可变的容器；
- 集合相当于只有键没有值的字典；
- 集合是无序的存储结构；
- 集合内的数据必须都是唯一的，不可变的。

1. 集合的创建

使用 set() 函数创建集合。

非空集合 用 {} 括起来，值用逗号分隔开。

示例

```
s1 = set()              # 用函数创建空集合
s2 = {1, 2, 3, 4}       # 创建非空集合的字面值
s3 = set(range(5))      # 调用 set(可迭代对象) 来创建
                        # 集合 s = {0, 1, 2, 3, 4}
s4 = set("ABC")         # s = {'B', 'C', 'A'}
s5 = set("ABCCCCCCC") # s = {'B', 'C', 'A'}
s6 = set(['ABC'])       # s = {'ABC'}

print(s1, s2, s3)
print(s4, s5, s6)
```

> 重复的数据放入集合中会自动去除重复数据

2. 集合的访问

in / not in 运算符

作用

- 用 in 可以判断一个值是否存在于集合中，如果存在返回 True, 否则返回 False
- not in 的返回值与 in 相反

示例

```
>>> s = {11, 22, 33}
>>> 22 in s
True
>>> 666 in s
False
>>> 'hello' not in s
True
```

集合是可迭代对象

集合是可迭代对象。

示例

```
>>> s = {300, 400, 500}
>>> for x in s:
...     print(x)
...
400
300
500
```

3. 集合的运算

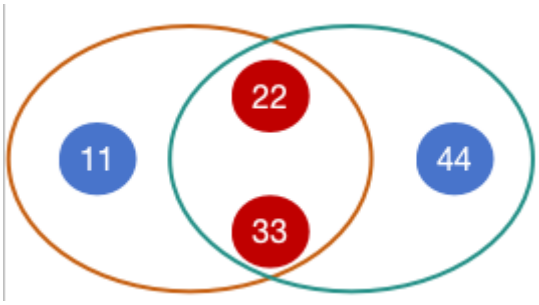
集合的运算

运算符	说明
s1 & s2	求 s1 和 s2 的交集
s1 | s2	求 s1 和 s2 的并集
s1 - s2	求 s1 对 s2 的差集
s1 ^ s2	求 s1 对 s2 的对称差集
s1 > s2	判断 s1 是否是 s2的 真超集
s1 >= s2	检测集合s2中的每个元素是否都在 另一个集合s1 之中
s1 < s2	判断 s1 是否是 s2的 真子集
s1 <= s2	检测集合s1中的每个元素是否都在 另一个集合s2 之中

交集 &

求两组数据的共有部分。

如果所示：



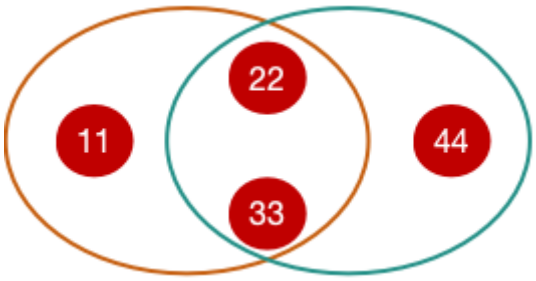
> 注：图中红色背景的数字为运算后的结果。

```
>>> s1 = {11, 22, 33}
>>> s2 = {22, 33, 44}
>>> s1 & s2
{22, 33}
```

并集 |

求两组数据的全部数据(去重后)。

如果所示：

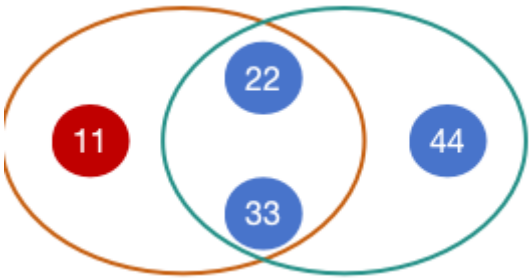



```
>>> s1 = {11, 22, 33}
>>> s2 = {22, 33, 44}
>>> s1 | s2
{11, 22, 33, 44}
```

差集 -

求去除 左表达式 集合中包含 右表达式集合中的数据。

如果所示：

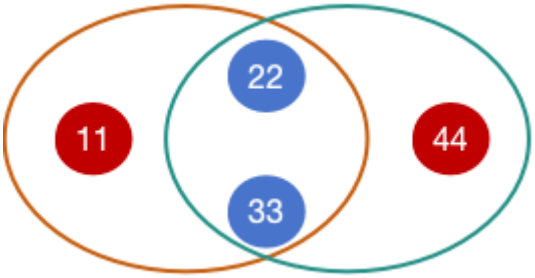



```
>>> s1 = {11, 22, 33}
>>> s2 = {22, 33, 44}
>>> s1 - s2
{11}
```

对称差集 ^

等同于集合 s1 和 s2 的 (s1 - s2) | (s2 - s1) 的运算。

如果所示：

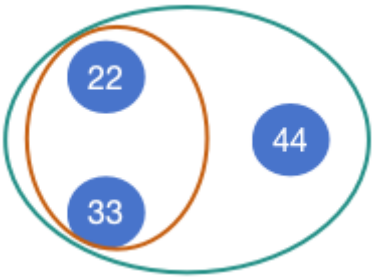



```
>>> s1 = {11, 22, 33}
>>> s2 = {22, 33, 44}
>>> s1 ^ s2
{11, 44}
```

子集 <

计算 左表达式 集合中所有元素 都包含在 右表达式集合中，如果是则返回True，否则返回False。

如果所示：




```
>>> s1 = {22, 33}
>>> s2 = {22, 33, 44}
```



```
>>> s1 < s2
True
```

4. 集合的方法

集合的方法

方法	说明
set.add(elem)	将元素 elem 添加到集合中。
set.remove(elem)	从集合中移除元素 elem。
set.clear()	从集合中移除所有元素。
set.pop()	从集合中移除并返回任意一个元素。 如果集合为空则会引发 KeyError。
set.copy()	复制集合

示例

```
>>> s = set()
>>> s
set()
>>> s.add(100)
>>> s.add(200)
>>> s.add(300)
>>> s
{200, 100, 300}
>>> s.remove(100)
>>> s
{200, 300}
>>> s.pop()
200
>>> s2 = s.copy()
>>> s
{300}
>>> s2
{300}
>>> s.clear()
>>> s
set()
>>> s2
{300}
>>>
```

5. 集合推导式

集合推导式

集合推导式是用一个可迭代对象来创建新的集合的方式。

语法

```
{ 值表达式 for 变量 in 可迭代对象 if 真值表达式 }
```

说明

- if 子句可以省略不写。
- 推导式运算出来的结果会被去重。

示例

使用集合推导式生成包含 1 到 10 之间所有偶数的集合。

参考答案：

```
numbers = {x for x in range(1, 11) if x % 2 == 0}
print(numbers)
```

6. 固定集合

固定集合 frozenset

什么是固定集合

固定集合是不可变的集合。

作用

- 固定集合可以作为字典的键。
- 固定集合可以作为集合的值。

使用 frozenset() 函数创建集合

固定集合构造(创建)函数 frozenset

函数	说明
frozenset()	创建一个空的固定集合对象
frozenset(iterable)	用可迭代对象创建一个新的固定集合对象

示例

```
>>> frozenset()
frozenset()
>>> frozenset('ABC')
frozenset({'B', 'C', 'A'})
>>> frozenset('ABCCCCCC')
frozenset({'B', 'C', 'A'})
```

集合和固定集合的运算

运算符	说明
s1 & s2	求 s1 和 s2 的交集
s1 | s2	求 s1 和 s2 的并集
s1 - s2	求 s1 对 s2 的差集
s1 ^ s2	求 s1 对 s2 的对称差集
s1 > s2	判断 s1 是否是 s2的 真超集
s1 >= s2	检测集合s2中的每个元素是否都在 另一个集合s1 之中
s1 < s2	判断 s1 是否是 s2的 真子集
s1 <= s2	检测集合s1中的每个元素是否都在 另一个集合s2 之中

python内置的数据类型（总结）

不可变数据类型

- 整数 int
- 浮点数 float
- 复数 complex
- 布尔数 bool
- 字符串 str
- 元组 tuple

- 固定集合 frozenset
- 字节串 bytes

可变的数据类型

- 列表 list
- 字典 dict
- 集合 set
- 字节数组 bytearray

第十二章、函数

什么是函数

函数是一段具有特定功能的代码块，它允许你为一段代码定义一个名称（即函数名），并通过这个名称来调用这段代码。并可以重复调用。

作用

- 提高代码的可读性、可维护性和重用性。

1. 函数的定义和调用

def 语句

作用

用来定义（创建）一个函数。

语法

```
def 函数名(形参变量1,形参变量2,形参变量3,...):
    语句块
```

说明

- 函数名字就是语句块的名称，它必须是标识符，并用来描述函数的用途。
- 形参列表，是零个或多个自定义形参变量组成，用于接收调用时传入的值。
- 函数有自己的名字空间，在函数是内部创建的变量（包括形参变量）会在函数调用结束后自动销毁。
- 函数可以有一个返回值，并必须通过return语句实现。

函数调用

使用 def 语句创建的函数，内部的语句并不会执行，只有在调用此函数的时候，函数内部的语句块才能够执行。

语法


```
函数名(实际调用传递参数1,实际调用传递参数2,...)
```

说明

- 函数调用是一个表达式。
- 如果函数内没有 return 语句，函数执行完毕后返回 None 对象。

函数定义和调用示例

```
# 定义一个控制全自动洗衣机洗衣服流程的函数washing_machine，用它来提示洗衣服的全过程
def washing_machine():
    print("放入衣服")
    print("注水")
    print("洗涤20分钟")
    print("排水")
    print("甩干")
    print("报警提示完成")

washing_machine()  # 第一次洗衣服
washing_machine()  # 第二次洗衣服
```

改写上述程序，加入一个形参变量，让此函数能够洗涤不同的衣物。

```
# 使用参数来控制洗衣机洗涤的物品
def washing_machine(something):
    print("放入"+something)
    print("注水")
    print("洗涤20分钟")
    print("排水")
    print("甩干")
    print("报警提示完成")

washing_machine('衣服')
washing_machine('羽绒服')
```

练习

定义一个函数，传入两个参数，让这个函数把最大的值打印到终端。

参考答案：

```
def mymax(a, b):
    if a > b:
        print("最大值是", a)
```

```
    else:
        print("最大值是", b)
```

2. return语句

None 对象

None 是一个特殊的常量，用于表示空值或“无”的概念。

作用

1. 用于函数的返回值，None 代表没有返回任何数据。
2. 用于函数参数和变量的绑定。当一个变量绑定 None 时，表示此变量没有绑定任何值。

说明

- None的布尔值为 False。
- None 是不可变对象，不能对其进行修改。

函数调用

使用 def 语句创建的函数，内部的语句并不会执行，只有在调用此函数的时候，函数内部的语句块才能够执行。

语法

```
函数名(实际调用传递参数1, 实际调用传递参数2, ...)
```

说明

- 函数调用是一个表达式。
- 如果函数内没有 return 语句，函数执行完毕后返回 None 对象。

return 语句

作用

用在函数的内部，用于结束当前函数的执行，返回到调用此函数的地方，同时返回一个数据。

语法

```
return [表达式]
```

> 语法中的 中括号代表 括起来的内容可以省略。

说明

- return 语句后面的表达式可以省略，省略后相当于 return None。
- 如果函数内部没有 return 语句，则函数执行完毕后返回 None，相当于在最后一条语句后有一条 `return None`。

示例

```
# 此实例示意return语句的用法

def washing_machine(something):
    print("放入"+something)
    print("注水")
    print("洗涤20分钟")
    print("排水")
    return '干净的' + something
    print("甩干")  # 此处的代码不会执行
    print("报警提示完成")


value = washing_machine('衣服')
print('获取到的数据是', value)
value = washing_machine('羽绒服')
print('获取到的数据是', value)
```

练习

写一个函数 myadd，实现给出两个数，返回这两个数的和。

如:

```
def myadd(x, y):
    ...  # 此处完成相关代码

a = int(input("请输入第一个数: "))
b = int(input("请输入第二个数: "))
print('您输入的两个数之和是:', myadd(a, b))
```

3. 函数的实参传递

形式参数（形参）和实际参数（实参）的概念

- 实际参数是给函数的数据，主打一个给，递出去。
- 形式参数是拿调用者的数据，主打一个接，用变量来绑定传过来的数据。

形式参数

- 定义：形式参数是在函数定义时声明的参数，他是临时变量（局部变量），用于接收调用函数时传递的参数值。
- 位置：形式参数出现在函数的定义中，位于函数名后面的括号内。

示例：

```
def myadd(x, y):
    ...
```

实际参数：

- 定义：实际参数是在函数调用时传递给函数的值或表达式。
- 位置：实际参数出现在函数调用时的括号中。

示例：

```
result = myadd(100, 200)
```

函数的调用实参传递方法

在调用函数时可以为函数提供数据，称之为实参传递。

Python 实际参数传递方式如下：

- 位置传参:
  - 序列传参：用 * 将序列拆解后按位置进行传递的传参方式。
- 关键字传参
  - 字典的关键字传参：将字典用 ** 拆解后进行关键字传参的传参方式。

位置传参说明

- 实际参数和形式参数通过位置进行传递和匹配。

- 实际参数的个数必须与形式参数的个数相同。

关键字传参说明 - 实参和形参按形参名进行匹配，可以不按位置进行匹配。

示例

```
# 此示例示意函数的实参传递方式

def myfunc(a, b, c):
    print('a:', a)
    print('b:', b)
    print('c:', c)

# 位置传参
myfunc(1, 2, 3)
# 序列传参：用 * 将序列拆解后按位置进行传递的传参方式
mylist = [10, 20, 30]
myfunc(*mylist)

# 关键字传参
myfunc(c=300, b=200, a=100)
myfunc(b=222, c=333, a=111)
# 字典的关键字传参：将字典用 ** 拆解后进行关键字传参的传参方式
mydict = {'c':3000, 'a':1000, 'b':2000}
myfunc(**mydict)

# 综合传参
myfunc(100, c=300, b=200)
mylist2 = (100, 200)
mydict2 = {'c':300}
myfunc(*mylist2, **mydict2)
```

函数综合传参

说明

- 函数的传参方式，在能确定形参能唯一匹配到相应实参的情况下可以任意组合。
- 函数的位置传参要先于关键字传参。

4. 函数形参的定义方式

形式参数（形参）

形式参数（Formal Parameter）是函数定义时，函数名括号内的变量列表里的变量，用于绑定调用函数时传递过来的实际值（实际参数）。

函数定义的语法

```
def 函数名(变量1, 变量2, ...):
    语句块
```

函数形式参数定义方法有以下五种:

- 缺省参数
- 位置形参
- 星号元组形参(*args)
- 命名关键字形参
- 双星号字典形参（**kwargs）

4.1 缺省参数

函数的缺省参数

作用

当函数调用没有实参传递数据时，使用缺省值作为传入的值。

语法

```
def 函数名(形参名1=默认实参1, 形参名2=默认实参2, ...):
    语句块
```

说明

- 位置形参 的缺省参数必须自右向左依次存在。

函数的缺省参数示例

```
def add_numbers(a, b, c=0, d=0):
    return a + b + c + d

print(add_numbers(10, 20, 30, 40))
print(add_numbers(1, 2, 3))
print(add_numbers(100, 200))
```

练习

定义一个函数myrange(start,stop,step)可以传递一个实参，两个实参和三个实参，这个函数返回一个符合range函数规则的列表。

如:

```
def myrange(...):
    ...

print(myrange(3))  # 打印[0, 1, 2]
print(myrange(3, 6))  # 打印[3, 4, 5]
print(myrange(1, 10, 2)  # 打印[1, 3, 5, 7, 9]
```

4.2 位置形参和星号元组形参

位置形参

作用

按位置摆放形参。

语法

```
def 函数名(形参名1, 形参名2, ...):
    语句块
```

示例

```
# 此示例示意函数的位置形参的定义方法

def add_numbers(a, b):
    return a + b

print(add_numbers(1, 2))
```

星号元组形参

作用

收集多余的位置实参。

语法

```
def 函数名(*元组形参名):
    语句块
```

说明

元组形参名一般命名为 args，且只能有一个。

```
# 此示例示意函数的星号元组形参的定义方法

def add_numbers(a, b, *args):
    print('元组的个数:', len(args))
    print('args:', args)
    return a + b + sum(args)

# print(add_numbers(1, 2))
# print(add_numbers(1, 2, 3, 4))
print(add_numbers(11, 22, 33, 44, 55, 66, 77))
```

练习

写一个函数，mul_numbers()，此函数可以传入一个或多个整数作为实参（个数不限制），此函数返回这些整数的乘积。

```
def mul_numbers(...):
    ...

print(mul_numbers(3))  # 打印: 3
print(mul_numbers(3, 4))  # 打印: 12
print(mul_numbers(3, 4, 5)  # 打印: 60
```

4.3 命名关键字形参和双星号字典形参

命名关键字形参

作用

强制，所有的参数都必须用关键字传参。

语法

```
def 函数名(*, 命名关键字形参1, 命名关键字形参2, ...):
    ...
# 或者
def 函数名(*args, 命名关键字形参1, 命名关键字形参2, ...):
    ...
```

示例

```
# 命名关键字形参示例

# def myfunc(a, b, *, c, d):
def myfunc(a, b, *args, c, d):
        print('a:', a, 'b:', b, 'c:', c, 'd:', d)

myfunc(1, 2, 3, 4)  #  强制报错
myfunc(100, 200, d=400, c=300)
myfunc(b=222, a=111, d=444, c=333)
```

双星号字典形参

作用

收集多余的关键字传参。

语法

说明

- 字典形参名 最多有一个。
- 字典形参名 一般命名为 kwargs。

函数的形参定义方法说明

- 位置形参，星号元组形参，命名关键字参数，双星号字典形参，缺省参数可以混合使用。
- 函数的形参定义自左至右的顺序为：位置形参，星号元组形参，命名关键字参数，双星号字典形参。

示例

```
# 双星号字典形参示例

def myfunc(a, b, *args, c, d, **kwargs):
    print('a:', a, 'b:', b, 'c:', c, 'd:', d)
    print('kwargs:', kwargs)

myfunc(100, 200, d=400, c=300, e=500, f=600)
```

5. 局部变量和全局变量

作用域

作用域指的是变量、函数和类等标识符在代码中的可访问区域。

Python主要有四种作用域：

- 局部作用域（Local）
- 嵌套函数作用域（Enclosing Function Local）
- 全局作用域（Global）
- 内置作用域（Built-in）

局部变量和全局变量

局部变量

- 在函数内部定义的变量称为局部变量(函数的形参也是局部变量)。
- 局部变量只能在函数的内部使用。
- 局部变量在函数调用时才能够被创建，在函数调用之后会自动销毁。

全局变量

- 在函数外部、.py文件的内部定义的变量称为全局变量。
- 全局变量可以在整个程序中被访问，包括在函数内部。
- 在函数内部不能直接改变全局变量的绑定关系（不能重绑定)。

局部变量说明:

- 在函数内首次对变量赋值是创建局部变量，再次为变量赋值是修改局部变量的绑定关系。
- 在函数内部的赋值语句不会对全局变量造成影响。
- 局部变量只能在其被声明的函数内部访问，而全局变量可以在整个模块范围内访问。

局部变量示例

```
# 此实例示意什么是局部变量

def fn(a, b):
    c = 10
    print('局部变量abc:', a, b, c)    # a, b, c三个都是局部变量

fn(1, 2)
print('全局变量abc:', a, b, c)        # 报错， 因为a,b,c 在调用后就销毁了
```

全局变量示例

```
# 此实例示意全局变量和用法

a = 100  # 函数外部创建的变量是全局变量，此变量一直有效
b = 200
c = 300
d = 400

def fn(a, b):
    c = 10
    print(a, b, c)  # a, b, c三个都是局部变量，函数内部优先访问局部变量
    print('d:', d)

fn(1, 2)
print(a, b, c, d)  # 报错， 因为a,b,c 在调用后就销毁了
```

6. global语句

global语句

作用

告诉解释器，global语句声明的一个或多个变量，这些变量为全局变量，即在此 `.py` 文件内部有效。

语法

```
global 变量名1, 变量名2, ...
```

说明

- 全局变量如果要在函数内部被赋值，则必须经过全局声明。
- 全局变量在函数内部不经过声明就可以直接访问。
- 不能先声明局部的变量，再用 global 声明为全局变量，此做法不附合规则。
- 函数形参列表里变量是已经定义的局部变量，不允许使用 global 语句声明。

示例

```
# 此实例示意global语句的用法

a = 100
b = 200
c = 300
d = ['三国演义', '红楼梦', '西游记']
```

```
def fn(a, b):
    global c
    c = 666
    print(a, b, c)  # a, b, c三个都是局部变量，函数内部优先访问局部变量
    d.append('水浒')

fn(1, 2)
print('全局的 a:', a, 'b:', b, 'c:', c)
print('d:', d)
```

7. lambda表达式

标识符和变量

python的标识符都是变量，它可以绑定数据、函数、类和模块等。

函数名也是变量，它绑定一个函数。

```
def 函数名(形参变量1,形参变量2,形参变量3,...):
    语句块
```

示例

```
# 此实例示意函数名是一个变量，它绑定一个函数

def f1():
    print('f1被调用!')

def f2():
    print('f2被调用!')

f1, f2 = f2, f1
f1()  # 调用谁?
f2()  # f1被调用!
```

python 创建函数的方法有两种

1. def 语句
2. lambda 表达式

lambda 表达式

lambda表达式通常用于需要函数对象的地方，但函数又足够简单以至于不需要用标准的def语法来定义。

作用：

通常用于函数的传参并需要短小函数的地方，随时创建，随时销毁。

语法

```
lambda  函数的参数列表: 表达式
```

说明

- 函数的参数列表可以为空。
- lambda 表达式 的创建函数只能包含一个表达式。
- lambda 比函数简单且可以随时创建和销毁，有利于减少程序的偶合度。

lambda 表达式示例

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

print('1 + 2 =', myadd(1, 2))  # 3

# myadd 函数可以改写成
myadd2 = lambda x, y: x + y
print('3 + 4 =', myadd2(3, 4))  # 7
```

第十三章、类和对象

1. pass语句

pass语句

作用:

用来填充语法空白。

语法

```
pass
```

示例

```
# 如果 成绩在 0~100 什么都不做, 其他提示"您的输入有误"

score = int(input('请输入成绩:'))
if 0 <= score <= 100:
    pass
else:
    print('您的输入有误!')
```

2. 类和对象

python内置的类型

- 整数-int
- 浮点数-float
- 复数-complex
- 布尔数-bool
- 字符串-str
- 列表-list
- 元组-tuple
- 字典-dict
- 集合-set

- 固定集合-frozen-set

类（class）和对象（object）

- 整数-int:
  - 666
  - 999
- 浮点数-float
  - 3.14
  - 0.618
- 车-Car
  - 京A.88888
  - 沪A.66666
- 狗-Dog
  - 张三牵着的二哈
  - 李四拴着的藏獒

class语句

作用

创建一个自定义的类型。

语法

```
class 类名:
    实例方法的定义
```

说明

- 类名必须是一个标识符，它是一个变量，它绑定一个类。
- 按编码规范，一般自定义的类名都使用首字母大写的变量名（即大驼峰命名法）。

构造函数

使用类来创建一个对象（实例）。

语法

```
类名(实际参数列表)
```

返回值

此类型的对象。

示例

```
# 创建类Dog
class Dog:
    pass

# 创建对象
dog1 = Dog()  # 创建一个对象
dog2 = Dog()  # 创建另外一个对象

list1 = list()  # 创建一个列表
list2 = list()  # 创建另外一个列表
```

3. 对象的属性

属性（Attribute）

在Python中，与对象相关联的数据或方法通常用这个对象自身的变量绑定，这些变量称为属性（Attribute）。

实例属性

- 对象自身的变量。

属性的操作

添加和修改属性

语法

```
对象.属性名 = 表达式
```

访问属性

语法

```
对象.属性名
```

删除属性

语法

```
del 对象.属性名
```

示例

```
# 对象的属性示例

class Dog:
    pass

dog1 = Dog()
dog2 = Dog()

dog1.kind = '哈士奇'
dog1.color = '灰色'
dog2.kind = '藏獒'
dog2.color ='棕色'
dog1.color = '黑色'  # 修改
# del dog1.color  # 删除属性

print(dog1.color, '的', dog1.kind)
print(dog2.color, '的', dog2.kind)
```

4. 对象的方法

对象的方法

对象的方法（又称实例方法）是与对象相关联的函数。

方法是定义在类中的函数，用于操作对象的数据或执行特定任务。

方法可以通过对象调用，也可以使用类调用，并且可以访问和修改对象的属性。

作用:

用于描述一个对象的行为，让此类型的全部对象都拥有相同的行为。

定义语法

```
class 类名(继承列表):
    def 方法名(self, 参数1, 参数2, ...):
        语句块
```

说明

- 对象的方法的实质是函数，是定义在类内的函数。
- 对象的方法至少有一个形参，第一个形参绑定调用这个方法的对象，一般命名为"self"。

对象方法的调用语法

调用方法

```
实例.实例方法名(调用传参)
```

或

```
类名.实例方法名(实例, 调用传参)
```

方法和函数的区别

- 方法属于类，仅供此类的对象使用（调用）。
- 函数属于全局，供全部程序使用。

示例

```
# 对象的方法示例

class Dog:
    def eat(self, food):
        print(self.color, '的', self.kind, '吃', food)

dog1 = Dog()
dog2 = Dog()

dog1.kind = '哈士奇'
dog1.color = '灰色'
dog2.kind = '藏獒'
dog2.color ='棕色'

dog1.eat('包子')  # 等同于 Dog.eat(dog1, '骨头')
Dog.eat(dog2, '牛肉')
```

5. 初始化方法

作用

对新创建的对象添加统一的属性。

语法

```
class 类名(继承列表):
    def __init__(self, 形参1, 形参2, 形参3, ...):
        语句块
```

说明

- 初始化方法名必须为 **init** 不可改变。
- 他会被构造函数自动调用，并可以将参数传入到__init__方法中。
- 初始化方法如果要提前返回则必须返回None。

示例

```
# 对象的初始化方法示例

class Dog:
    def __init__(self, k, c):
        self.kind = k  # 种类
        self.color = c  # 颜色
    def eat(self, food):
        print(self.color, '的', self.kind, '吃', food)

dog1 = Dog('哈士奇', '灰色')
dog2 = Dog('藏獒', '棕色')

dog1.eat('包子')  # 等同于 Dog.eat(dog1, '骨头')
dog2.eat('牛肉')
```

6. 析构器方法

对象的生命周期

对象的周期基本分为3步：

1. 创建
2. 使用

3. 销毁

如图所示：

析构方法

作用

- 在对象被销毁时自动调用，用于执行一些清理操作，例如释放资源、关闭文件或断开网络连接等。

语法

```
class 类名(继承列表):
    def __del__(self):
        语句块
```

说明

- 析构方法名必须为 `__del__` 不可改变，此方法在对象销毁前自动调用。
- 对象销毁时调用：当对象的引用计数为 0 或程序结束时，`__del__` 方法会被自动调用。

示例

```
# 对象的析构方法示例

class Dog:
    def __init__(self, k, c):
        self.kind = k  # 种类
        self.color = c  # 颜色
        print(self.color, '的', self.kind, '小狗被创建')
    def __del__(self):
        print(self.color, '的', self.kind, '小狗被销毁')
    def eat(self, food):
        print(self.color, '的', self.kind, '吃', food)
```

```
dog1 = Dog('哈士奇', '灰色')
dog2 = Dog('藏獒', '棕色')

dog1.eat('包子')  # 等同于 Dog.eat(dog1, '骨头')
dog2.eat('牛肉')

del dog1  # 删除变量

print('程序结束')
```

7. is 和is not运算

Python中任何的对象（数字，字符串以及自己写的类创建的对象）都存在于计算机内存（注意是运行内存）中。内存你可以认为是村里的联排别墅，自东到西每一户都有一个整数的门牌号1，2，3，4....，内存也是如此，我们要把对象放在别墅中，每个别墅只能放一个对象。

内存地址

id() 函数

作用

返回对象的内存地址的标识值。该值是一个整数，在此对象的生命周期中保证是唯一且恒定的。

> 你可以认为是返回别墅的门牌号。

调用格式

```
id(object)
```

is 和 is not 运算

作用

- is 运算符是根据id进行身份测试运算，如果是同一个对象则返回True，否则返回False。

- is not 运算结果与is运算的布尔值相反。

> id 相同就一定是同一个对象。

语法

```
x is y
或
x is not y
```

> x, y 表示表达式或变量返回的数据对象

示例

```
# is / is not 运算示例

class Dog:
    def __init__(self, k, c):
        self.kind = k  # 种类
        self.color = c  # 颜色
        print(self.color, '的', self.kind, '小狗被创建')
    def eat(self, food):
        print(self.color, '的', self.kind, '吃', food)

dog1 = Dog('哈士奇', '灰色')
dog2 = Dog('藏獒', '棕色')
dog3 = dog1
print(dog1 is dog2)  # False
print(dog1 is dog3)  # True
```

None对象的判断

在计算机内存中，None对象只有一个且一直存在，因此判断一个变量是否绑定None，经常使用 is / is not 运算而不使用 == 和 != 的值比较运算。

示例

```
if obj is None:
    print('obj变量绑定的是None对象')
```

8. 类型相关的函数

类型相关的函数是用来返回对象的类型或者判断某个对象是否是某种类型的函数。

函数	说明
type(obj)	返回对象的类（类型）
isinstance(obj, class_or_tuple)	返回这个对象obj 是否是 某个类的对象，或者某些类中的一个类的对象，如果是返回True,否则返回False

示例

```
# 此实例示意和类型相关的函数的用法

def sum_of_numbers(obj):
    # 此函数返回obj 绑定的所有数字的和，如果obj绑定一个数字，则直接返回
    # 如果obj绑定的是含有一些列整数的可迭代对象则返回可迭代对象内所有数字的和
    # if type(obj) is float:
    if isinstance(obj, float):
        return obj
    # elif type(obj) is list:
    elif isinstance(obj, list):
        return sum(obj)

print(sum_of_numbers(3.14))  # 3.14
print(sum_of_numbers([100, 200, 300]))  # 600
```

第十四章、面相对象编程

1. 面向对象编程概述

面向对象编程

面向对象编程（Object-Oriented Programming，简称OOP）是一种软件设计思想。他是用对象来描述现实世界的各种关系和操作，用类来规范对象的行为的编程思想。

面向对象的编程语言的特征：

- 封装：隐藏内部细节，提供安全访问。
- 继承：代码复用，扩展父类功能。
- 多态：同一方法不同实现。
- 抽象：定义接口，隐藏复杂逻辑。

优点：

- 提升了代码的可维护性、复用性和扩展性，适用于复杂系统的开发。

面向对象示例

有两个人

- 姓名: 张三；年龄: 35
- 姓名: 李四；年龄: 10

行为

- 教别人学东西 teach；
- 赚钱 work；
- 借钱 borrow；
- 显示自己的信息 show_info；

事情

- 张三 教 李四 学 Python；
- 李四 教 张三 学 王者荣耀；
- 张三 工作赚了 1000 元钱；

- 李四 向 张三 借 200 元钱；
- 35 岁的 张三 有钱 800 元，他学会的技能: ['王者荣耀']；
- 10 岁的 李四 有钱 200 元，他学会的技能: ['Python']；

类

用下面的类来描述人类的行为。

```
class Human:
    def __init__(self, n, a):
    def teach(self, other, subject):
    def works(self, money):
    def borrow(self, other, money):
    def show_info(self):
```

示例

```
class Human:
    def __init__(self, n, a):
        self.name = n  # 姓名
        self.age = a  # 年龄
        self.money = 0  # 钱数为0
        self.skills = []  # 学到的技能

    def teach(self, other, subject):
        other.skills.append(subject)
        print(self.name, "教", other.name, '学', subject)

    def works(self, money):
        self.money += money
        print(self.name, '工作赚了', money, '元钱')

    def borrow(self, other, money):
        '描述一个人向其它人借钱的行为,它人有钱必借,不够不借'
        if other.money > money:
            other.money -= money
            self.money += money
            print(self.name, "向", other.name, '借', money, '元钱')
        else:
            print(other.name, '没有', money, '元钱，不能借给', self.name)

    def show_info(self):
        print(self.age, '岁的', self.name,
            '有钱', self.money, '元，他学会的技能:', self.skills)

# 事情
zhang3 = Human('张三', 35)
li4 = Human('李四', 10)
# 张三 教 李四 学 python
zhang3.teach(li4, 'Python')
```

```
# 李四 教 张三 学 王者荣耀
li4.teach(zhang3, '王者荣耀')
# 张三 工作赚了 1000 元钱
zhang3.works(1000)
# 李四 向 张三 借 200 元钱
li4.borrow(zhang3, 200)
# 35 岁的 张三 有钱 800 元，它学会的技能: ['王者荣耀']
zhang3.show_info()
# 8 岁的 李四 有钱 200 元，它学会的技能: ['Python']
li4.show_info()
```

2. 封装

封装是指将数据（属性）和操作数据的行为（方法）封装在类中，并控制外部对内部数据的访问权限，让用户通过特定的方法才能操作这些对象。

封装的目的

- 隐藏类的内部实现细节，让使用者不用关心这些细节，从而保证数据的安全性。
- 防止外部直接修改对象的内部状态。
- 提供清晰的接口（尽可能少的方法(或属性)）供外部使用。

封装的实现

私有属性和方法

- python类中以双下划线(__)开头，不以双下划线结尾的标识符为私有成员，私有成员只能使用此类的方法来进行访问和修改。
  - 以__开头的属性为类的私有属性，在子类和类外部无法直接使用。
  - 以__开头的方法为私有方法，在子类和类外部无法直接调用。

示例

```
# 此示例示意封装的用法

class Human:
    def __init__(self, n, a):
        self.name = n  # 姓名
        self.age = a  # 年龄
        self.__money = 0  # 钱数为0
        self.skills = []  # 学到的技能

    def teach(self, other, subject):
        other.skills.append(subject)
```

```
        print(self.name, "教", other.name, '学', subject)

    def works(self, money):
        self.__money += money
        print(self.name, '工作赚了', money, '元钱')

    def borrow(self, other, money):
        '描述一个人向其它人借钱的行为,它人有钱必借,不够不借'
        if other.__money > money:
            other.__money -= money
            self.__money += money
            print(self.name, "向", other.name, '借', money, '元钱')
        else:
            print(other.name, '没有', money, '元钱，不能借给', self.name)

    def show_info(self):
        print(self.age, '岁的', self.name,
            '有钱', self.__money, '元，他学会的技能:', self.skills)

# 事情
zhang3 = Human('张三', 35)
li4 = Human('李四', 10)
# 张三 教 李四 学 python
zhang3.teach(li4, 'Python')
# 李四 教 张三 学 王者荣耀
li4.teach(zhang3, '王者荣耀')
# 张三 工作赚了 1000 元钱
zhang3.works(1000)
# 李四 向 张三 借 200 元钱
li4.borrow(zhang3, 2000000)
# zhang3.__money -= 2000000
# li4.__money += 2000000
# 35 岁的 张三 有钱 800 元，它学会的技能: ['王者荣耀']
zhang3.show_info()
# 8 岁的 李四 有钱 200 元，它学会的技能: ['Python']
li4.show_info()
```

3. 继承

继承允许一个类（子类）基于另一个类（父类或基类）创建，并继承父类的属性和方法。

通过继承，子类可以复用父类的代码，并可以扩展或修改父类的行为。

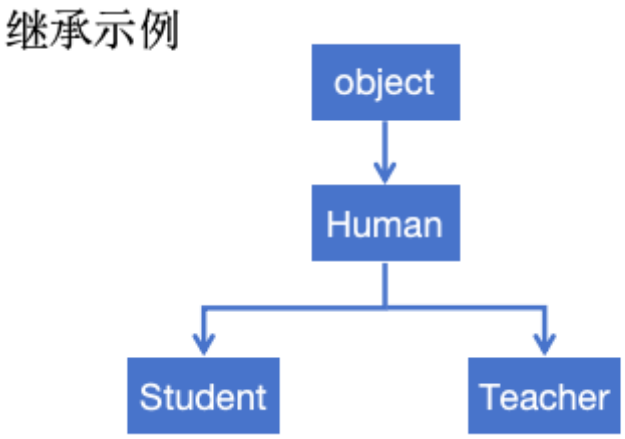
语法

```
class 类名(基类1,基类2,...):
    语句块
```

> 类名后面括号和内部的类名称之为继承列表。

说明

- 如果不写继承列表，则新类继承自object类。
- Python3任何类都直接或间接的继承自object类。



继承的目的

1. 代码复用：
   - 子类公用的方法和属性提取出来放入父类，提高代码的复用性。

2. 扩展功能：
   - 子类可以在父类的基础上添加新的属性和方法。

3. 多态性：
   - 子类可以重写父类的方法，实现不同的行为。

示例

```
# 继承示例
class Human:
    def eat(self, what):
        print("吃了", what)
    def sleep(self, hour):
        print("睡了", hour, "小时")

class Student(Human):
    def study(self, subject):  # 学习
        print("学习:", subject)
```

```
class Teacher(Human):
    def teach(self, language):
        print("教:", language)


h1 = Human()
h1.eat("油条")
h1.sleep(12)

s1 = Student()
s1.eat('鸡蛋')
s1.sleep(7)
s1.study("python")

t1 = Teacher()
t1.eat('胡辣汤')
t1.sleep(8)
t1.teach("面向对象")
```

4. 覆盖

什么是覆盖

覆盖是指在有继承关系的类中，子类中实现了与基类同名的方法，在子类的实例调用该方法时，实际调用的是子类中的同名的方法，这种现象叫覆盖。

作用

- 实现和父类同名，但功能不同的方法。

示例

```
# 此示例示意覆盖的效果
class Student:
    def study(self):
        print("小学生学习! ")

class Doctor(Student):
    def study(self):
        print("博士生学习! ")

# s1 = Student()
# s1.study()

d1 = Doctor()
d1.study()
```

> Student类内的 study 方法被覆盖（不会被调用）

子类显式调用基类的覆盖方法

子类对象显式调用基类方法的方式一：

```
基类名.方法名(实例，实际调用参数, ...)
```

子类对象显式调用基类方法的方式二：

- 使用super函数

super函数

函数	说明
super(cls, obj)	返回绑定超类的实例(要求obj必须为cls类型的实例)
super()	返回绑定超类的实例，等同于:super(**class**, 实例方法的第一个参数)，必须用在方法内调用

作用

借助super() 返回的实例间接调用其父类的覆盖方法。

示例

```
# 此示例示意如何显示调用被覆盖的方法
class Student:
    def study(self):
        print("小学生学习! ")

class Doctor(Student):
    def study(self):
        print("博士生学习! ")
    def rest(self):  # 休息时间
        # self.study()
        # Student.study(self)
        super().study()

d1 = Doctor()
# d1.study()
# 在方法外部显式调用被覆盖的方法1(使用类调用):
# Student.study(d1)
# 在方法外部显式调用被覆盖的方法2(使用Super函数调用):
```

```
# super(Doctor, d1).study()
d1.rest()
```

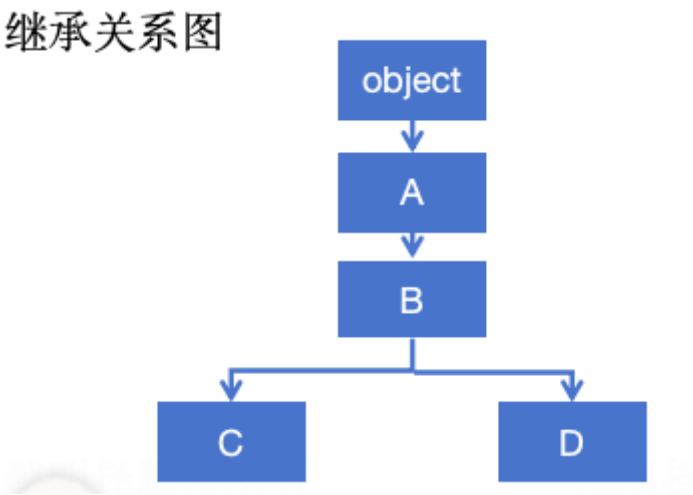
示例

```
# 显式调用基类的初始化方法
class Human:
    def __init__(self, n, a):
        self.name, self.age = n, a
    def infos(self):
        print("姓名:", self.name)
        print("年龄:", self.age)

class Student(Human):
    def __init__(self, n, a, s=0):
        super().__init__(n, a)
        self.score = s
    def infos(self):
        super().infos()
        print("成绩:", self.score)

# h1 = Human('张三', 20)
# h1.infos()
s1 = Student("魏明择", 35, 100)
s1.infos()
```

5. issubclass函数

继承相关的函数

函数	说明
issubclass(cls, class_or_tuple)	判断一个类是否继承自其它的类，如果此类cls是class或 tuple中的一个派生子类则返回True,否则返回False



如图

1. B 是 A 的子类
2. C 和 D 是 B的子类
3. C 和 D 也是 A 的子类（子类一定是父类类型）

```
>>> class A:
...     pass
...
>>> class B(A):
...     pass
...
>>> class C(B):
...     pass
...
>>> class D(B):
...     pass
...
>>> issubclass(C, B)
True
>>> issubclass(C, A)
True
>>> issubclass(B, A)
True
>>> issubclass(C, int)
False
>>> issubclass(C, (int, str, object))
True
>>>
```

6. 多态

什么是多态

多态（Polymorphism）是指不同类的对象可以使用相同的接口进行操作，并根据不同对象的实际类型执行不同方法的现象称为多态。

特点

- 接口一致性：不同类的对象可以通过相同的接口调用方法。
- 运行时绑定：具体调用哪个方法是在运行时根据对象的类型决定的。

好处

- 多态特性使得代码更加灵活、可扩展和易于维护。

示例

```
# 多态示例
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print('旺! ')

class Cat(Animal):
    def speak(self):
        print('喵! ')

def animal_speek(animal):
    animal.speak()

# 创建对象
dog = Dog()
cat = Cat()

# 调用相同的接口
animal_speek(dog)  # 旺!
animal_speek(cat)  # 喵!
```

7. 抽象

什么是抽象

抽象（Abstraction） 是指隐藏复杂实现细节、仅暴露必要接口的设计思想。

通过抽象，程序员可以定义类的接口，而不必关心这些接口的具体实现。

抽象有助于减少程序的复杂性，提高代码的可维护性和可读性。

实现方法

- Python -- 抽象类
- Java -- 抽象类或接口
- C++ -- 抽象类

抽象类的实现

Python中的抽象通过抽象基类（ABC-Abstract Base Classes）和抽象方法（@abstractmethod）实现，用于定义规范和隐藏实现细节。

说明

抽象（Abstraction） 是一种隐藏复杂实现细节、仅暴露必要接口的设计思想。它允许开发者专注于对象的行为（做什么），而不是具体的实现（怎么做）。

示例

```
# 此示例示意抽象类的用法
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):  # 面积
        pass
    @abstractmethod
    def perimeter(self):  # 周长
        pass

class Circle(Shape):  # 圆
    def __init__(self, radius):
        self.radius = radius
    def area(self):  # 面积
        return 3.1415 * self.radius ** 2
    def perimeter(self):  # 周长
        return 2 * 3.1415 * self.radius

class Rectangle(Shape):  # 矩形
    def __init__(self, w, h):
        self.w, self.h = w, h
    def area(self):  # 面积
        return self.w * self.h
```

```
    def perimeter(self):  # 周长
        return 2 * self.w + 2 * self.h

s = Shape()
```

> 此实例使用了装饰器（`@abstractmethod`），我们高级的内容才会学到装饰器，先用着再说！

第十五章、模块

1. 模块概述

什么是模块

- 模块是一个包含有一系列数据、函数、类等定义的程序文件，此文件供其他程序使用。
- python模块是一个.py结尾文件。

> 提示：python允许使用C语言编写模块（此课程不涉及）

模块的主要作用：

- 让一些相关的数据，函数，类等有逻辑的组织在一起，可在多个程序中重复使用。
- 提供命名空间管理，全局变量只在模块内部有效，避免模块间命名冲突。
- 将代码按功能拆分到不同模块，提升可读性和可维护性。

模块的分类：

1. 标准库模块（在python3 程序内部，可以直接使用）。
   - 内置模块（C语言编写，集成在python解释执行器中）。
   - python编写的预安装模块（.py结尾的文件）。
2. 第三方模块（需要下载安装后才能使用）。
3. 自定义模块（也可以作为他人的第三方模块）。

示例

下面这个文件 `mymod.py` 就是一个模块。

```
# 文件名: mymod.py
'''自定义模块示例:
作者: 张三
此模块内有两个变量: galaxy、home，一个函数welcome和一个类Dog。
此模块共有四个全局变量，这四个全局变量绑定的数据可以被其他模块使用'''

galaxy = '银河系'
home = '地球'

def welcome(somebody):
    '此函数用来欢迎某星外来客！'
```

```
    print('欢迎', somebody, '来到', home)

class Dog:
    '此类用于描述地球上的一种小动物，此种小动物是人类忠实的朋友'
    def speak(self):
        '此方法用来展示此种小动物的叫声！'
        print('旺！')

if __name__ == '__main__':
    print('测试模块')
    dog1 = Dog()
    dog1.speak()
```

2. import语句

作用：

在本地模块内建立变量，来绑定其他模块或数据。

import 语句有三种形式:

1. `import` 语句
2. `from import` 语句
3. `from import *` 语句

import 语句的语法

- 导入一个或多个模块到当前程序

```
import 模块名1 [as 新名字1],模块名2 [as 新名字2]
```

- 导入一个模块内部的部分属性到当前程序

```
from 模块名 import 模块属性名 [as 属性新名]
```

- 导入一个模块内部的全部属性到当前程序

```
from 模块名 import *
```

示例

```
# 此示例示意import语句的用法
# 此文件为李四写的主模块，此模块最先调用并调用其他模块的函数和类。

import mymod
dog1 = mymod.Dog()
dog1.speak()
mymod.welcome('外星人')
print(mymod.home)
import mymod as mm
mm.welcome('超人')

from mymod import welcome
welcome('阿凡达')
from mymod import welcome as w, Dog as d
w('魏明择')
dog2 = d()
dog2.speak()

from mymod import *
print(galaxy, home)
welcome('xxxxx')
dog3 = Dog()

print("程序运行完毕")
```

模块的存放位置

- 自定义模块必须放在当前主模块（最先启动的模块）的 .py 文件夹内才能够被主模块导入。
- 系统模块和第三方模块会放在python安装是指定的文件夹中，这些文件存放在sys.path列表中。

打印模块的存放位置：

```
import sys
for p in sys.path:
    print(p)
```

3. 模块的属性

dir函数

作用

返回名字空间内所有变量组成的列表。

调用格式

函数	说明
dir() dir(object)	如果没有实参，则返回当前本地作用域中的名称列表。如果有实参，它会尝试返回该对象的有效属性列表。

模块内系统定义的属性

`__file__` 属性：

用于绑定模块的路径。

__name__属性：

1. 用来记录模块自身的名字；
2. 如果自己是主模块（最先启动的模块）内的__name__属性，则绑定字符串`'__main__'`；
3. 如果自己不是主模块，则`__name__`属性绑定模块的名字。

示例

```
# 此实例示意模块的属性

import mymod

print(dir())
print(dir(mymod))
print(__file__)
print(mymod.__file__)

print(__name__)  # '__main__'
print(mymod.__name__)  # 'mymod'
```

4. 文档字符串

什么是文档字符串

在模块、函数、类中，第一个没有赋值给任何变量的字符串称为文档字符串，此字符串会绑定在`__doc__`属性上，供PyCharm的提示或 help() 函数使用。

help()函数

作用 - 启动内置的帮助系统（此函数主要在交互式中使用）。

说明

- 如果没有实参，解释器控制台里会启动交互式帮助系统。如果实参是一个字符串，则在模块、函数、类、方法、关键字或文档主题中搜索该字符串，并在控制台上打印帮助信息。
- 如果实参是其他任意对象，则会生成该对象的帮助页。

示例

```
# 此实例示意文档字符串的用法

import mymod
print(mymod.__doc__)
mymod.welcome('超人')
print(mymod.welcome.__doc__)
dog1 = mymod.Dog()
print(mymod.Dog.__doc__)
dog1.speak()
print(mymod.Dog.speak.__doc__)
```

用法：在交互执行模式下：

```
>>> help(mymod)
```

或

```
>>> help(mymod.Dog)
```

此时查看的就是文档字符串。

5. 标准库模块

什么是标准库模块

python 中有二百多个已经写好的模块，这个模块会在安装python的时候直接被安装。且可以直接使用import 语句导入并使用。

标准库模块官方网址:

https://docs.python.org/zh-cn/3/library/index.html

示例


```
>>> import math
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.sin(math.pi / 4) # 求 sin(45度)
0.7071067811865475
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)
5
>>> random.randint(1, 6)
1
>>>
```

6. 第三方模块

什么是第三方模块

第三方模块是由 Python 社区或第三方开发者创建的、不属于 Python 标准库的模块。这些模块提供了丰富的功能，扩展了 Python 的能力。

特点: - 非官方：不由 Python 官方维护，而是由社区或个人开发者开发和维护。 - 功能丰富：提供了标准库未涵盖的许多高级功能。 - 开源：大多数第三方模块是开源的，可以自由使用和修改。

常用的第三方模块

1. 数据处理与分析
   - NumPy
   - Pandas
   - SciPy

2. 数据可视化
   - Matplotlib
   - Seaborn

3. 机器学习与深度学习
   - Scikit-learn
   - TensorFlow PyTorch

4. Web开发
   - Django
   - Flask

第三方模块的官网：

https://pypi.org/

第三方模块需要使用 pip 命令在线安装后才能够使用。

Windows 下安装第三方模块

Windows 下需要使用命令提示符安装。

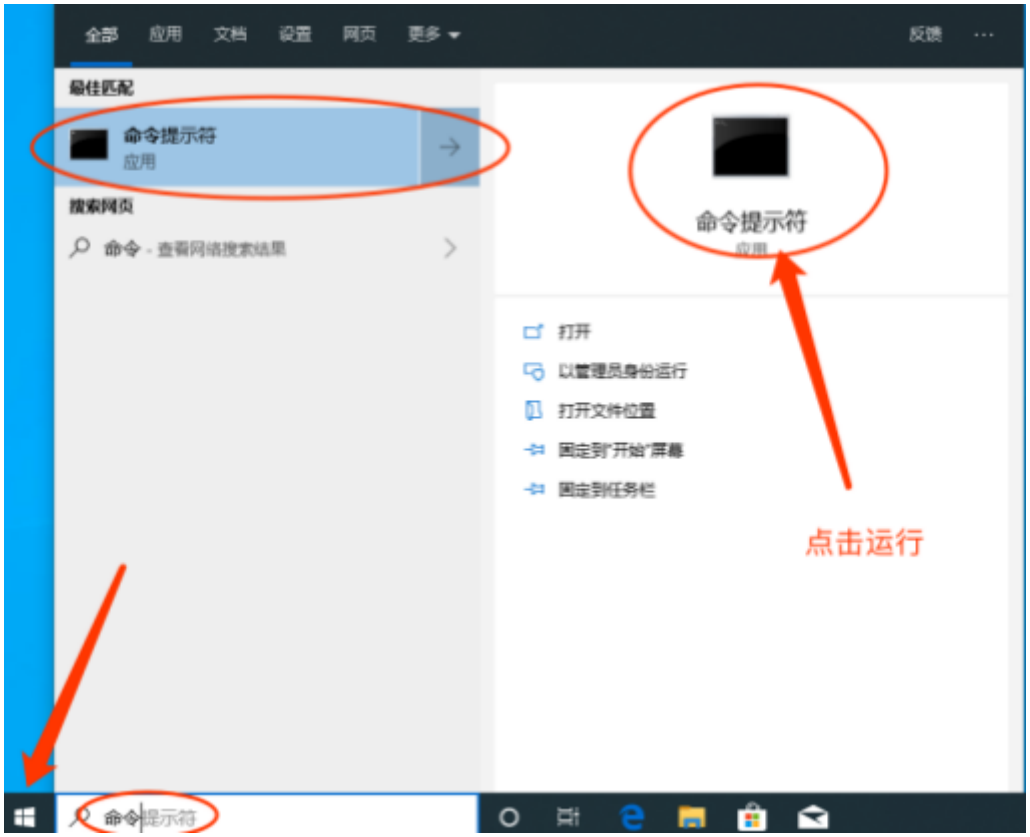
命令格式如下

```
pip install 模块名
```

如安装 numpy 模块需要执行如下命令：

```
pip install numpy
```

Windows 启动 "命令提示符" 如图所示：



Mac/Linux 下安装第三方模块

Mac 下需要使用 终端 安装。

命令格式如下

```
pip3 install 模块名
```

如安装 numpy 模块需要执行如下命令：

```
pip3 install numpy
```

Mac 启动 "终端" 如图所示：

安装指定版本的模块

使用pip命令可以安装指定版本的模块和包，命令格式如下：

```
# Windows 下执行的pip命令
pip install 模块名[==版本号]
# Mac OS 下执行的pip3命令
pip3 install 模块名[==版本号]
```

示例

```
# 安装2.2.0 版本的django
pip3 install django==2.2.0
# 安装3.2.0 版本的django
pip3 install django==3.2.0
```

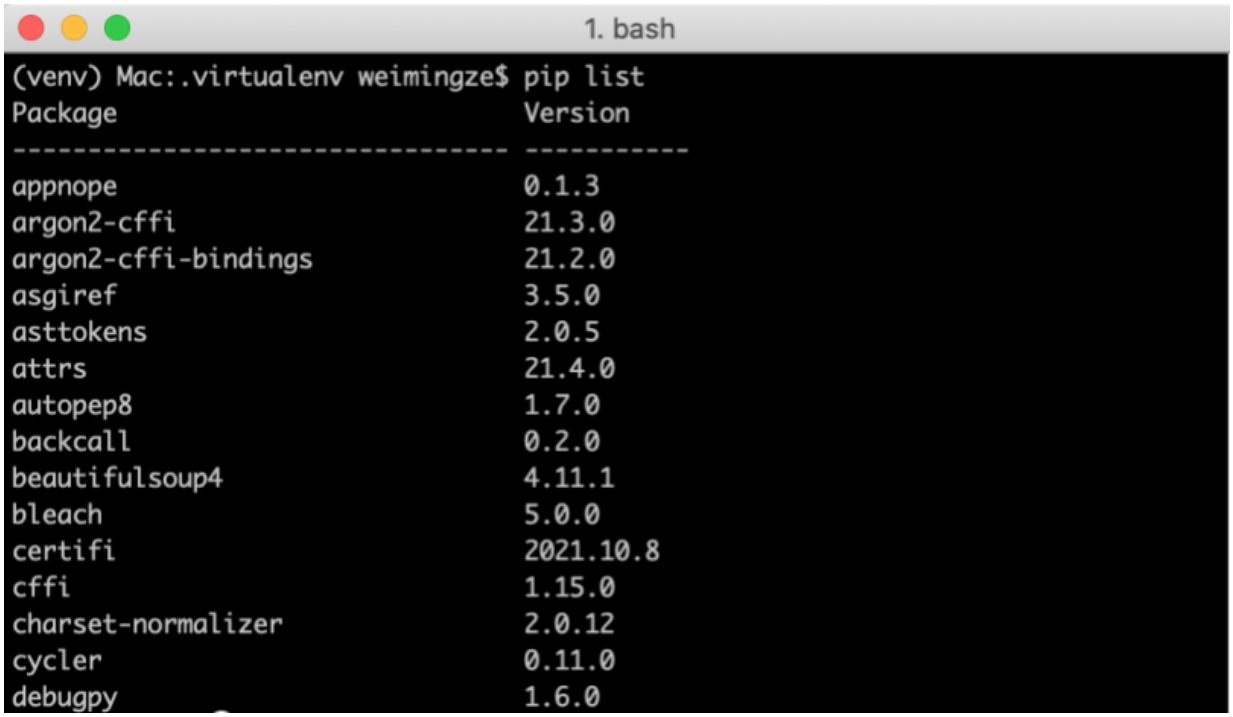
查看已经安装的第三方模块

在使用 pip 命令后面加上list选项可以查看已经安装的安装包和版本。

命令格式：

```
# Windows 系统下
pip list
# mac os 系统下
pip3 list
```

效果如图：



卸载 Python 第三方模块

在 Linux 终端或 Window 命令提示符下输入如下 pip 命令可以卸载模块或包。

命令格式：

```
# Windows 系统下
pip uninstall 模块名
# mac os 系统下
pip3 uninstall 模块名
```

示例

```
# 卸载 pandas 模块
pip3 uninstall pandas
```

练习

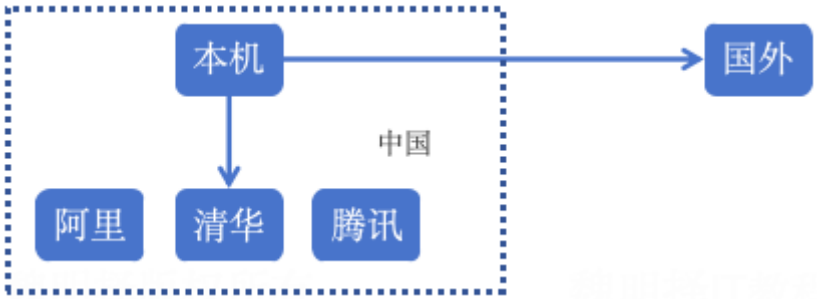
- 安装numpy、pandas、matplotlib模块并查看其官方文档；
- 安装 jupyter 模块；
- 安装 tensorflow 模块。

7. pip源

什么是源

源是放python源码的服务器，默认的源服务区在国外，访问比较慢，在安装第三方模块时可以更换源服务器。

如图:



pip源地址

国内的源地址:

```
清华大学 ： https://pypi.tuna.tsinghua.edu.cn/simple/
阿里云：http://mirrors.aliyun.com/pypi/simple/
中国科学技术大学 ： http://pypi.mirrors.ustc.edu.cn/simple/
华中科技大学：http://pypi.hustunique.com/
豆瓣源：http://pypi.douban.com/simple/
腾讯源：http://mirrors.cloud.tencent.com/pypi/simple
华为镜像源：https://repo.huaweicloud.com/repository/pypi/simple/
```

pip 换源

在使用 pip 命令安装第三方模块时可以使用 -i 选项后跟源地址进行换源。

命令格式：

```
# Windows 系统下
pip install 模块名 -i 源的服务器地址
```

```
# mac os 系统下
pip3 install 模块名 -i 源的服务器地址
```

示例

```
# 安装 pandas 模块
pip3 install pandas -i https://pypi.tuna.tsinghua.edu.cn/simple
```

第十六章、包

1. 包

什么是包

包（Package） 是一种组织代码的方式，用于将相关的模块（Module）组织在一起。

作用:

通过包，可以更好地管理大型项目中的模块，避免命名冲突。

包的分类

1. 常规包
   - 包是一个文件夹（也叫目录）。
2. 命名空间包（此课程不讲解）
   - 包是一个或多个.zip结尾的压缩文件包。

常规包

常规包是一个文件夹，包内的文件夹称为子包，包内的.py 文件则称为包内的模块。

说明

包一定是模块，模块一定不是包。

常规包示例

以下描述一个包的组织结构

```
mypack/
├── chat_tools
│   ├── chat_menu.py
│   ├── dingding.py
│   ├── __init__.py
│   └── wechat.py
├── __init__.py
├── menu.py
└── work_tools
    ├── __init__.py
```

```
    ├── pycharm.py
    └── wps.py
```

以下列出包文模块文件的名称和内容

文件: `mypack/menu.py`

```python
def show_menu():
    print('1) 打开钉钉')
    print('2) 打开微信')
    print('3) 打开PyCharm')
    print('4) 打开Word')
    print('5) 打开Excel')
```

文件: `mypack/__init__.py`

```python
'''
这是一个自己创建的包
这个包包含两个子包:
chat_tools子包存放的是聊天工具相关的模块;
work_tools子包存放的是工作相关的模块;
'''

def show_describe():
    '''此函数是包mypack内的函数'''
    print('mypack包内函数的示例!')
```

文件: `mypack/chat_tools/chat_menu.py`

```python
def show_chat_menu():
    while True:
        print('1) 启动钉钉')
        print('2) 启动微信')
        print('3) 启动wps里面的Excel')
        print('0) 退出')
        sel = int(input('请输入选项: '))
        match sel:
            case 1:
                pass
            case 2:
                pass
            case 3:
                pass
            case 0:
                break
            case _:
                print('您的输入有误，请重新选择! ')
```

文件: mypack/chat_tools/dingding.py

```
def start_dingding():
    '钉钉的启动函数'
    print('正在启动钉钉程序...')
```

文件: mypack/chat_tools/wechat.py

```
def start_wechat():
    '微信的启动函数'
    print('正在启动微信程序...')
```

文件: mypack/chat_tools/__init__.py

```
'''
这个是魏老师创建的聊天相关的子包
'''
```

文件: mypack/chat_tools/pycharm.py

```
'''
PyCharm启动相关的函数
'''

def start_pycharm():
    'PyCharm的启动函数'
    print('正在启动PyCharm程序...')
```

文件: mypack/chat_tools/wps.py

```
'''
金山的wps启动相关的函数
'''

__all__ = ['start_word']

def start_word():
    'word的启动函数'
    print('正在启动wps的word程序...')

def start_excel():
    'excel的启动函数'
    print('正在启动wps的excel程序...')
```

文件: mypack/chat_tools/__init__.py

```
'''
这个是魏老师创建的办公相关的子包
'''

__all__ = ['pycharm', 'wps']
```

2. 包的导入

导入包和导入模块类似，以下介绍包的导入的语法。

语法

```
# 同模块的导入规则
import 包名 [as 包别名]
import  包名.模块名 [as 模块新名]
import  包名.子包名.模块名 [as 模块新名]

from 包名 import 模块名 [as 模块新名]
from 包名.子包名 import 模块名 [as 模块新名]
from 包名.子包名.模块名 import 属性名 [as 属性新名]

# 导入包内的所有子包和模块
from 包名.模块名 import *
```

示例

```
# 此示例示意包的导入

import mypack
mypack.show_describe()
import mypack.menu
# mypack.menu.show_menu()
import mypack.menu as me
# me.show_menu()
import mypack.work_tools.pycharm
# mypack.work_tools.pycharm.start_pycharm()

from mypack.work_tools.pycharm import start_pycharm
start_pycharm()

from mypack.work_tools.wps import *
start_word()
start_excel()
```

`__init__.py` 文件

常规包内可以含有一个 `__init__.py` 文件，用这个文件用来编写包的内容。

- `__init__.py` 是常规包内的初始化文件，此文件会在包加载时被自动调用。

作用：

1. 编写此包的内容。
2. 在内部填写包的文档字符串。

示例

文件: `mypack/__init__.py`

```
'''
这是一个自己创建的包
这个包包含两个子包:
chat_tools子包存放的是聊天工具相关的模块;
work_tools子包存放的是工作相关的模块;
'''
```

3. 包的相对导入

什么是包的相对导入

包的相对导入是指包内模块的相互导入。

语法

```
from 相对路径包或模块 import 属性或模块名
# 或
from 相对路径包或模块 import *
```

包的相对路径

在 `from xxxx import` 语句中可以使用相对导入。

- 在 from 和 import 间相对路径的格式

```
. 代表当前目录
.. 代表上一级目录
... 代表上二级目录
.... 以此类推
```

> 注：相对导入时不能超出包的外部。

示例

文件: mypack/chat_tools/chat_menu.py

改写后：

```
# 此示例示意包的相对导入

from .dingding import start_dingding
from .wechat import start_wechat
from ..work_tools.wps import start_excel
def show_chat_menu():
    while True:
        print('1) 启动钉钉')
        print('2) 启动微信')
        print('3) 启动wps里面的Excel')
        print('0) 退出')
        sel = int(input('请输入选项: '))
        match sel:
            case 1:
                start_dingding()
            case 2:
                start_wechat()
            case 3:
                start_excel()
            case 0:
                break
            case _:
                print('您的输入有误，请重新选择! ')
```

主模块调用

```
# 此实例示意模块的相对导入

from mypack.chat_tools.chat_menu import show_chat_menu

show_chat_menu()
```

4. __all__列表

`__all__` 列表

`__all__`列表作用：

用来记录此包或模块中，在使用 from import * 语句导入时，有哪些子包、模块或属性需要导入（列表之外的属性将不导入）。

说明

- `__all__` 属性必须绑定一个列表。
- `__all__` 列表只在 `from xxxx import *` 语句中起作用。

示例

文件: `mypack/chat_tools/__init__.py`

```
'''
这个是魏老师创建的办公相关的子包
'''

__all__ = ['pycharm', 'wps']
```

主模块

```
# 此示例示意__all__列表的用法

from mypack.work_tools import *

pycharm.start_pycharm()
wps.start_word()

from mypack.work_tools.wps import *
start_word()
# start_excel()
```

第十七章、文件

1. 文件操作

什么是文件

- 文件是存储在计算机上的数据集。内部可能是文档、图片、视频、程序等。
- 文件通常用来长期存储数据。
- 文件由文件名和文件中的数据两部分组成。

文件存储的特点

- 存储数据量大。
- 断电不易丢失。

文件的操作流程

1. 打开文件；
2. 读或写文件；
3. 关闭文件。

打开文件

文件只有打开才能进行操作，打开文件的 open 函数如下表：

函数	说明
open(file, mode='r', encoding=None, newline=None)	打开文件并返回对应的文件对象file，如果该文件不能被打开，则引发 OSError类型的错误。

参数

1. file：文件路径名。
2. mode：打开模式，默认是’r’读取文件，’w’是创建新文件并写入文件。
3. encoding：文本文件的编码，中文是’utf-8’或’gb2312’/’gbk’/’gb18030’。
4. newline：换行符号，windows是CRLF(’\r\n’),mac和Linux是LR(’\n’)。

关闭文件

关闭文件是为了释放内存，将内存的数据写入到磁盘上。

方法	说明
file.close()	关闭文件，一个文件只能关闭一次。关闭后的文件不能再进行读写操作

> 任何的操作系统，一个应用程序同时打开文件的数量有最大数限制，因此使用完毕以后都要关闭。

示例

文件：`poem.txt`

```
赋得古原草送别
白居易
离离原上草，一岁一枯荣。
野火烧不尽，春风吹又生。
远芳侵古道，晴翠接荒城。
又送王孙去，萋萋满别情。
```

主模块

```python
# 此实例示意文件的基本操作

# 读文件操作
# 1. 打开文件
file = open('poem.txt', 'r')
# 2. 读写文件
s = file.read()
print('s:', s)
# 3. 关闭文件
file.close()


# 写文件操作
# 1. 打开文件
file2 = open('myfile.txt', 'w')
# 2. 写文件
file2.write('python')
file2.write('是最简单的编程语言')
file2.write('\n')
file2.write('这是第二行!')
```

```
# 3. 关闭文件
file2.close()
```

2. 读写文件

读文件的方法

读文件的打开模式必须为mode='r'。

方法	说明
file.read(size=-1)	从一个文件流中最多读取size个字符，如果不给出参数，则默认读取文件中全部的内容并返回。
file.readline()	读取一行数据，如果到达文件尾则返回空行
file.readlines(max_chars=-1)	返回每行字符串的列表,max_chars为最大字符数

示例

```
# 此示例示意读文件的方法

fr = open('poem.txt')

# s1 = fr.read(2)
# print('s1:', s1)
# s2 = fr.read()
# print('s2:', s2)
# s3 = fr.read()  # s3 = ''
# print('s3:', s3)

# aline = fr.readline()
# aline2 = fr.readline()

lines = fr.readlines() # ['...\n', 'xxxx\n']
print('lines:', lines)

fr.close()
```

写文件的方法

写文件的打开模式必须为mode='w'或mode='x'或mode='a'。

方法	说明
file.write(text)	写一个字符串到文件流中，返回写入的字符数
file.writelines(lines)	将字符串的列表或字符串的列表中的内容写入文件

示例

```
# 此示例示意写文件的方法

fw = open('test_write.txt', 'w')

fw.write('a line text!\n')
fw.write('second line text!\n')

fw.writelines(['3th line text\n', '4th line text!'])

fw.close()
```

3. open函数的参数

文件只有打开才能进行操作。

函数	说明
open(file, mode='r', encoding=None, newline=None)	打开文件并返回对应的文件对象file，如果该文件不能被打开，则引发 OSError类型的错误。

参数

1. file：文件路径名。
2. mode：打开模式，默认是’r’读取文件，’w’是创建新文件并写入文件。
3. encoding：文本文件的编码，中文是’utf-8’或’gb2312’/’gbk’/’gb18030’。
4. newline：换行符号，windows是CRLF(’\r\n’),mac和Linux是LR(’\n’)。

编码格式 `encoding`

encoding参数在有汉字的文件中起作用。

汉字编码

1. 国际编码（UNICODE）：
   - 'utf-8'。
2. 信息产业部国标（GB）系列编码:
   - 'gb2312' -- 6000+个字。
   - 'gbk' -- 21003个汉字。
   - 'gb18030'-- 27000+个汉字。

换行符newline

newline 参数在有不同的操作系统中有不同的值。

1. Windows系统--- 回车换行：CRLF('\r\n')。
2. 新Mac/Linux系统 --- 换行：LF('\n')。
3. 经典Mac OS 系统 --- 回车：CR('\r')。

示例

```
# 此示例示意 open函数的参数的用法

fw = open('myfile2.txt', 'w',
          encoding='utf-8'
          , newline='\r\n')

fw.write('this is first line!\n')
fw.write('这是第二行!\n')

fw.close()
```

4. csv文件

CSV (Comma Separated Values) 格式是电子表格和数据库中最常见的文件格式。

文件格式:

- 以行为单位的文本文件，微软的Excel要求文件的行结尾是CRLF（'\r\n'）。
- 列使用英文的逗号','作为分隔符，如果内容中出现逗号使用双引号（"）将此单元格的内容括起来。
- 微软的Excel要求以国标系列的编码（GB2312/GBK或GB18030）；金山的WPS可以使用国标系列的编码，也可是使用UNICODE转换编码UTF-8。

例如表格为：

姓名	班级	语文成绩	数学成绩
魏明择	1班	80	90
冯华	2班	99	88

csv 文件格式为：

文件：`mydata.csv`

```
姓名,班级,语文成绩,数学成绩
魏明择,1班,80,90
冯华,2班,99,88
```

csv模块

作用:

1. 读取csv格式的文件。
2. 写入csv格式的文件。

> csv模块是标准库模块。

使用 csv 模块读取 csv 文件示例

```
# 此实例示意使用csv模块读取csv文件

import csv

csv_file = open('mydata.csv')
csv_reader = csv.reader(csv_file)
for row in csv_reader:
    print('row:', row)


csv_file.close()
```

使用 csv 模块写入 csv 文件示例

```
# 此实例示意使用csv模块写入csv文件
```

```
import csv

headline = ['姓名', '年龄', '身高']
data = [['张三', '18', '1.75'],
        ['李四', '20', '1.88']]

csv_file = open('student.csv', 'w')

csv_writer = csv.writer(csv_file)
csv_writer.writerow(headline)
csv_writer.writerows(data)

csv_file.close()
```

第十八章、常用内置函数

内置函数

内置函数通常由C语言编写的函数，它定义在标准库中的 `__builtins__` 模块中。

标准库模块的导入:

```
from __builtins__ import *
```

> 以上操作在 python程序启动前就已经完成，不用开发者自己导入。

查看 `__builtins__` 内置模块

```
>>> help(__builtins__)
```

学过的内置函数

类型	函数
基本输入输出函数	print()、input()
类的构造函数	int()、bool()、float()、complex()、str()、list()、tuple()、dict()、set()、frozenset()
容器统计函数	len()、sum()、max()、min()
布尔运算函数	any()、all()
整数生成器函数	range()
类和对象函数	id()、dir()、isinstance()、issubclass()、type()、super()
文件打开函数	open()
帮助函数	help()

本章要学的内置函数

类型	函数
数学运算函数	abs()、pow()、round()、divmod()
整数进制转换函数	bin()、oct()、hex()
字符编码函数	chr()、ord()
排序函数	sorted()

1. 数学运算相关的内置函数

数学运算相关的内置函数

函数	说明
abs(x)	返回一个数字的绝对值。 参数可以是整数、浮点数。 如果参数是一个复数，则返回它的模。
pow(base, exp, mod=None)	返回 base 的 exp 次幂；如果 mod 存在，则返回 base 的 exp 次幂对 mod 取余（比 pow(base, exp) % mod 更高效）。 两参数形式 pow(base, exp) 等价于乘方运算符: base**exp。
round(number, ndigits=None)	返回 number 舍入到小数点后 ndigits 位精度的值。 如果 ndigits 被省略或为 None，则返回最接近输入值的整数。
divmod(a, b)	接受两个（非复数）数字作为参数并返回由当对其使用整数除法时的商和余数组成的数字对。

示例

```
>>> abs(100)
100
>>> abs(-100)
100
>>> pow(5, 2)
25
>>> pow(5, 2, 3)
1
>>> round(3.1415926, 3)
3.142
>>> round(3.1415926, 2)
3.14
>>> round(3.1415926)
3
>>> divmod(14, 3)
(4, 2)
```

2. 进制转换相关的内置函数

函数	说明
bin(x)	将一个整数转换为带前缀 "0b" 的二进制数字符串。 结果是一个合法的 Python 表达式。
oct(x)	将整数转换为带前缀 "0o" 的八进制数字符串。 结果是一个合法的 Python 表达式。
hex(x)	将整数转换为带前缀 "0x" 前缀的小写十六进制数字符串。

示例

```
>>> bin(1000)
'0b1111101000'
>>> oct(1000)
'0o1750'
>>> hex(1000)
'0x3e8'
>>>
```

3. 字符编码相关的内置函数

文字编码

文字编码是指每个文字的整数编号。

python字符串使用UNICODE编码（0～1114111）

- 英文编码：0~127；
- 其他编码：大于等于128。

字符编码相关的内置函数：

函数	说明
ord(c)	对表示单个 Unicode 字符的字符串，返回代表它 Unicode 编码的整数。
chr(i)	返回 Unicode 码位为整数 i 的字符的字符串格式。这是 ord() 的逆函数。实参的合法范围是 0 到 1,114,111（16 进制表示是 0x10FFFF）。

示例

```
>>> ord('A')
65
>>> ord('B')
66
>>> ord('明')
26126
>>> ord('择')
25321
>>> chr(26126)
'明'
>>> chr(65)
'A'
```

打印全世界所有的文字

```
for code in range(1114111):
    print(chr(code), end='')
```

4. 排序相关的内置函数

排序相关的内置函数

函数	说明
sorted(iterable, key=None, reverse=False)	根据 iterable 中的项返回一个新的已排序列表。 key 指定带有单个参数的函数，用于从 iterable 的每个元素中提取用于比较的键。 默认值为 None reverse 为一个布尔值。 如果设为 True，则每个列表元素将按反向顺序比较进行排序。

示例

```
# 此示例示意排序函数sorted的使用

data1 = (4, 9, 1, 3, 8)  # 整数
# result = sorted(data1)
# result = sorted(data1, reverse=True)

data2 = ['123', '23', '1000', '900', '888']  # 字符串
# result = sorted(data2)
# result = sorted(data2, key=int)

data3 = [  # 字典数据
    {'name': '张三', 'age':18, 'score': 88},
```

```
    {'name': '李四', 'age': 15, 'score': 100},
    {'name': '王五', 'age': 19, 'score': 60},
    {'name': '赵六', 'age': 12, 'score': 70},
]
def get_age(stu):
    return stu['age']
# result = sorted(data3, key=get_age)
result = sorted(data3, key=lambda stu: stu['score'])
print('result:', result)
```

> 自己打开注释试一下吧。

第十九章、校园信息管理系统项目

信息管理系统

信息管理系统（Information Management System, IMS）是一种用于收集、存储、处理和分发信息的软件系统，旨在帮助组织有效管理和利用信息资源，支持决策制定和业务操作。

应用示例:

- CRM -- 客户关系管理（Customer Relationship Management）
- ERP -- 企业资源计划（Enterprise Resource Planning）
- SCM -- 供应链关系管理（Supply Chain Management）
- HIS -- 医院信息系统(Hospital Information System)

信息管理系统的核心功能

- 数据收集：从内部或外部来源获取数据（如传感器、用户输入等）。
- 存储与管理：通过文件、数据库或云存储管理结构化或非结构化数据。
- 处理与分析：清洗、转换数据，并通过算法或工具（如BI工具）生成洞察。
- 信息传递：以报告、仪表盘、通知等形式向用户提供有用信息。
- 控制与安全：确保数据完整性、访问控制和合规性。

1. 校园信息管理系统项目简介

项目目标

- 实现对一个学校的班级、学生信息及学生成绩的管理；
- 实现小学校园数据的电子化，有利于查阅和数据分析；
- 数据电子存档，减少纸张利用，节能减排。

项目功能

学校功能

- 添加班级
- 删除班级
- 进入管理班级
- 列出所用班级

- 列出班级排名（平均成绩高-低）
- 保存班级信息
- 加载班级信息
- 退出程序

班级功能

- 添加学生
- 修改学生的语文成绩
- 修改学生的数学成绩
- 删除学生
- 列出所有学生的成绩
- 按语文成绩从高到低显示学生成绩
- 按数学成绩从高到低显示学生成绩
- 退出班级

操作界面（主界面）

功能展示

```
         xxxx小学信息管理系统
+-------------------------------------+
| 1) 添加班级                          |
| 2) 删除班级                          |
| 3) 进入管理班级                      |
| 4) 列出所用班级                      |
| 5) 列出班级排名（平均成绩高 - 低）|
| 6) 保存班级信息                      |
| 7) 加载班级信息                      |
| 0) 退出程序                          |
+-------------------------------------+
请选择:
```

操作界面（班级管理界面）

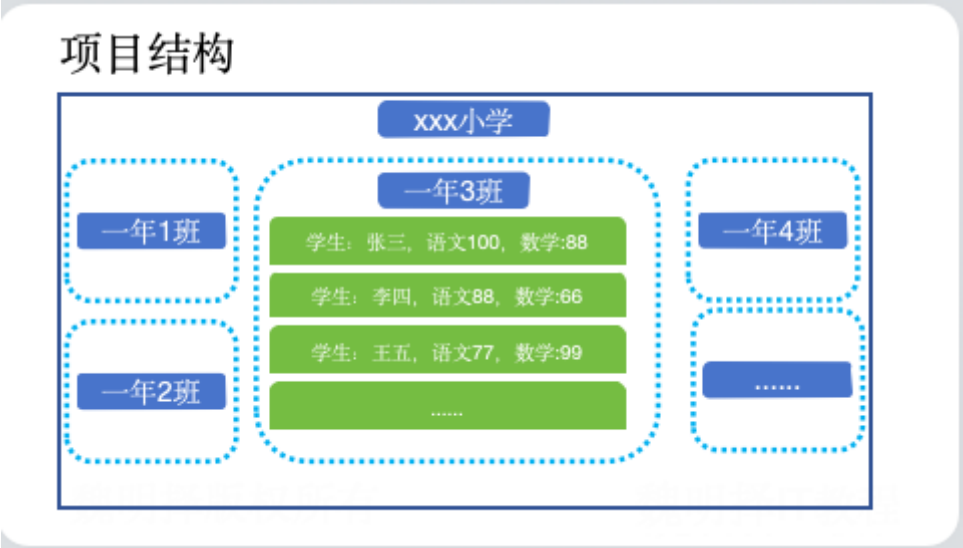
功能展示

```
         一年1班-班级管理
+-------------------------------------+
| 1) 添加学生                          |
| 2) 修改学生的语文成绩                |
| 3) 修改学生的数学成绩                |
| 4) 删除学生                          |
| 5) 列出所有学生的成绩                |
```

```
| 6) 按语文成绩从高到低显示学生成绩 |
| 7) 按数学成绩从高到低显示学生成绩 |
| 0) 退出班级                       |
+-----------------------------------+
请选择:
```

项目结构

一个学校有多个班级，一个班级有多个学生，一个学生有姓名，语文成绩和数学成绩两项。如图：



项目文件

1. 学校相关模块：`school.py`；
2. 班级相关模块：`class_room.py`；
3. 学生相关模块：`student.py`。

初始内容如下

- 学校相关模块：`school.py`

```python
class_rooms = []  # 用于存放班级对象

def show_school_menu():
    print('        xxxx小学信息管理系统')
    print('+-----------------------------------+')
    print('| 1) 添加班级                       |')
    print('| 2) 删除班级                       |')
    print('| 3) 进入管理班级                   |')
    print('| 4) 列出所用班级                   |')
    print('| 5) 列出班级排名（平均成绩高 - 低）|')
    print('| 6) 保存班级信息                   |')
    print('| 7) 加载班级信息                   |')
```

```
    print('| 0) 退出程序                          |')
    print('+-------------------------------------+')

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                pass
            case '2':  # 2) 删除班级
                pass
            case '3':  # 3) 进入管理班级
                pass
            case '4':  # 4) 列出所用班级
                pass
            case '5':  # 5) 列出班级排名（平均成绩高 - 低）
                pass
            case '6':  # 6) 保存班级信息
                pass
            case '7':  # 7) 加载班级信息
                pass
            case '0':  # 0) 退出程序
                return
            case _:
                print('不存在的选项，请重新输入')
                import time
                time.sleep(2)  # 让程序睡眠2秒


if __name__ == '__main__':
    class_manager()
```

- 班级相关模块：`class_room.py`

```
import student
from student import Student


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

    def show_class_menu(self):
        '此函数用来显示操作菜单'
        print(f'        {self.class_name}-班级管理')
        print('+-------------------------------------+')
        print('| 1) 添加学生                          |')
        print('| 2) 修改学生的语文成绩                |')
        print('| 3) 修改学生的数学成绩                |')
```

```
        print('| 4) 删除学生                          |')
        print('| 5) 列出所有学生的成绩                |')
        print('| 6) 按语文成绩从高到低显示学生成绩 |')
        print('| 7) 按数学成绩从高到低显示学生成绩 |')
        print('| 0) 退出班级                          |')
        print('+--------------------------------------+')

    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    pass
                case '2':  # 2) 修改学生的语文成绩
                    pass
                case '3':  # 3) 修改学生的数学成绩
                    pass
                case '4':  # 4) 删除学生
                    pass
                case '5':  # 5) 列出所有学生的成绩
                    pass
                case '6':  # 6) 按语文成绩从高到低显示学生成绩
                    pass
                case '7':  # 7) 按数学成绩从高到低显示学生成绩
                    pass
                case '0':  # 0) 退出班级
                    return
                case _:
                    print('不存在的选项，请重新输入')
                    import time
                    time.sleep(2)  # 让程序睡眠2秒


if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    cr1.student_manager()
```

- 学生相关模块：`student.py`

```
class Student:
    '''学生类型'''
    def __init__(self, name, chinese, math):
        self.name = name
        self.chinese_score = chinese
        self.math_score = math
```

2. 添加班级功能的实现

实现方法

> 以下只显示修改内容，请自行比对修改部分。

新增 tools 模块：`tools.py` 用于各个模块都使用的工具函数。

```
def get_display_width(s):
    '''
    返回字符串s的显示宽度
    如:
    'abc' -->3
    '中文' --> 4
    'ABC中文' --> 7
    '''
    return sum([1 if ord(ch) < 128 else 2 for ch in s])
    # display_width = 0
    # for ch in s:
    #     if ord(ch) < 128:
    #         display_width += 1
    #     else:
    #         display_width += 2
    # return  display_width
```

- 学校相关模块：`school.py`

```
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

def add_class_room():
    '''添加班级'''
    class_name = input('请输入班级名称: ')
    # 判断 class_name 不能超过 10 个显示字符的宽度
    if tools.get_display_width(class_name) <= 10:
        cr = ClassRoom(class_name)
        class_rooms.append(cr)
        print('添加班级', class_name, '成功!')
    else:
        print('添加班级失败，班级名太长!')
    import time
    time.sleep(2)

def show_school_menu():
    print('         xxxx小学信息管理系统')
    print('+-------------------------------------+')
    print('| 1) 添加班级                          |')
```

```
    ...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                pass
...
```

3. 列出所有班级功能的实现

实现方法

- 工具模块：`tools.py`

```
def get_display_width(s):
    return sum([1 if ord(ch) < 128 else 2 for ch in s])


def center_to_display_width(s, width):
    '''将字符串s的左右两端添加空格，使其达到width的显示宽度!
    s = 'ABC中文', width = 10
    返回:' ABC中文  '
    '''
    s_width = get_display_width(s)  # 得到当前字符的显示宽度
    # 计算需要补充的空格数
    fill_blank_count = width - s_width
    return s.center(len(s) + fill_blank_count)
```

- 学校相关模块：`school.py`

```
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def list_all_class_room():
    print('+------+------------+')
    print('| 序号 |  班级名称  |')
    print('+------+------------+')
    number = 1
    for cr in class_rooms:
        print('| %4d | %s |' % (number, tools.center_to_display_width(cr.class_n
```

```
ame, 10)))
        print('+------+------------+')
        number += 1

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
...
```

4. 删除班级功能的实现

实现方法

- 学校相关模块：`school.py`

```
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def del_class_room():
    '''删除班级'''
    list_all_class_room()
    number = int(input('请输入删除班级的序号: '))
    index = number - 1  # 对应列表的索引
    if 0 <= index < len(class_rooms):
        del class_rooms[index]
        print('删除成功!')
    else:
        print('您输入的序号有误，删除失败!')
    import time
    time.sleep(2)

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
```

```
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
            case '3':  # 3) 进入管理班级
                pass
            case '4':  # 4) 列出所用班级
                list_all_class_room()
...

if __name__ == '__main__':
    class_rooms.append(ClassRoom('一年一班'))
    class_rooms.append(ClassRoom('2年2班'))
    class_manager()
```

5. 管理班级功能的实现

实现方法

- 学校相关模块：`school.py`

```
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def enter_class_manager():
    '''进入管理班级界面'''
    list_all_class_room()
    number = int(input('请输入一个要管理班级的序号: '))
    index = number - 1
    if index < 0 or index >= len(class_rooms):
        print('您输入的班级序号有误! ')
        return
    cr = class_rooms[index]
    cr.student_manager()

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
```

```
            case '3':  # 3) 进入管理班级
                enter_class_manager()
            case '4':  # 4) 列出所用班级
                list_all_class_room()
...

if __name__ == '__main__':
    class_rooms.append(ClassRoom('一年一班'))
    class_rooms.append(ClassRoom('2年2班'))
    class_manager()
```

6. 添加学生功能的实现

实现方法

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

    def add_student(self):
        student_name = input('请输入学生姓名: ')
        if tools.get_display_width(student_name) >= 20:
            print('学生的名字太长，添加失败!')
            return
        chinese_score = int(input('请输入学生的语文成绩: '))
        if chinese_score < 0 or chinese_score > 100:
            print('学生成绩不在合法范围内，添加失败!')
            return
        math_score = int(input('请输入学生的数学成绩: '))
        if math_score < 0 or math_score > 100:
            print('学生成绩不在合法范围内，添加失败!')
            return
        stu = Student(student_name, chinese_score, math_score)
        self.student.append(stu)
        print('添加学生', student_name, '成功!')

    def show_class_menu(self):
        '此函数用来显示操作菜单'
        print(f'        {self.class_name}-班级管理')
        print('+-------------------------------------+')
        print('| 1) 添加学生                          |')
        print('| 2) 修改学生的语文成绩                |')
```

```
        print('| 3) 修改学生的数学成绩              |')
        print('| 4) 删除学生                        |')
        print('| 5) 列出所有学生的成绩              |')
        print('| 6) 按语文成绩从高到低显示学生成绩 |')
        print('| 7) 按数学成绩从高到低显示学生成绩 |')
        print('| 0) 退出班级                        |')
        print('+------------------------------------+')

    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    pass
                case '3':  # 3) 修改学生的数学成绩
                    pass
                case '4':  # 4) 删除学生
                    pass
                case '5':  # 5) 列出所有学生的成绩
                    pass
                case '6':  # 6) 按语文成绩从高到低显示学生成绩
                    pass
                case '7':  # 7) 按数学成绩从高到低显示学生成绩
                    pass
                case '0':  # 0) 退出班级
                    return
                case _:
                    print('不存在的选项，请重新输入')
                    import time
                    time.sleep(2)  # 让程序睡眠2秒


if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    cr1.student_manager()
```

7. 列出所有学生信息功能的实现

实现方法

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def list_all_student_info(self, student_list):
        '''显示所有学生的信息'''
        print('+------+----------------------+--------+--------+')
        print('| 序号 |         姓名         |语文成绩|数学成绩|')
        print('+------+----------------------+--------+--------+')
        number = 1
        for stu in student_list:
            print('| %4d | %s |  %4d  |  %4d  |' % (
                number, tools.center_to_display_width(stu.name, 20),
            stu.chinese_score, stu.math_score))
            number += 1
        if len(student_list):
            print('+------+----------------------+--------+--------+')
...

    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    pass
                case '3':  # 3) 修改学生的数学成绩
                    pass
                case '4':  # 4) 删除学生
                    pass
                case '5':  # 5) 列出所有学生的成绩
                    self.list_all_student_info(self.student)
                case '6':  # 6) 按语文成绩从高到低显示学生成绩

...

if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
```

```
    cr1.student.append(Student('赵六', 80, 91))
    cr1.student_manager()
```

8. 删除学生功能的实现

实现方法

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def del_student(self):
        '''删除学生信息'''
        self.list_all_student_info(self.student)
        number = int(input('请选择要删除学生的序号: '))
        index = number - 1
        if index < 0 or index >= len(self.student):
            print('您输入的序号有错，删除失败!')
            return
        del self.student[index]
        print('删除学生成功!')

...

    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    pass
                case '3':  # 3) 修改学生的数学成绩
                    pass
                case '4':  # 4) 删除学生
                    self.del_student()
                case '5':  # 5) 列出所有学生的成绩
```

```
                    self.list_all_student_info(self.student)
...
```

9. 修改学生成绩功能的实现

实现方法

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def modify_chinese_score(self):
        '''修改语文成绩功能'''
        self.list_all_student_info(self.student)
        number = int(input('请输入要修改的语文成绩的学生的序号: '))
        index = number - 1
        if number < 0 or number > len(self.student):
            print('您输入的序号有错，修改失败!')
            return
        a_student = self.student[index]
        new_score = int(input('请输入学生' + a_student.name + '的新的语文成绩:'))
        a_student.chinese_score = new_score
        print('修改', a_student.name, '的成绩成功!')

    def modify_math_score(self):
        '''修改数学成绩功能'''
        self.list_all_student_info(self.student)
        number = int(input('请输入要修改的数学成绩的学生的序号: '))
        index = number - 1
        if number < 0 or number > len(self.student):
            print('您输入的序号有错，修改失败!')
            return
        a_student = self.student[index]
        new_score = int(input('请输入学生' + a_student.name + '的新的数学成绩:'))
        a_student.math_score = new_score
        print('修改', a_student.name, '的成绩成功!')

...

    def student_manager(self):
```

```
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    self.modify_chinese_score()
                case '3':  # 3) 修改学生的数学成绩
                    self.modify_math_score()

...
```

10. 按成绩排序功能的实现

实现方法

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...
    def list_student_info_order_by_chinese_score(self):
        '''按语文成绩从高到低显示学生成绩'''
        def get_chinese_score(a_student):
            return a_student.chinese_score
        result = sorted(self.student, key=get_chinese_score, reverse=True)
        print('按语文成绩排序的列表')
        self.list_all_student_info(result)

    def list_student_info_order_by_math_score(self):
        '''按数学成绩从高到低显示学生成绩'''
        def get_chinese_score(a_student):
            return a_student.chinese_score
        result = sorted(
            self.student,
            key=lambda a_stu: a_stu.math_score,
            reverse=True)
        print('按数学成绩排序的列表')
        self.list_all_student_info(result)

...
```

```
    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    self.modify_chinese_score()
                case '3':  # 3) 修改学生的数学成绩
                    self.modify_math_score()
                case '4':  # 4) 删除学生
                    self.del_student()
                case '5':  # 5) 列出所有学生的成绩
                    self.list_all_student_info(self.student)
                case '6':  # 6) 按语文成绩从高到低显示学生成绩
                    self.list_student_info_order_by_chinese_score()
                case '7':  # 7) 按数学成绩从高到低显示学生成绩
                    self.list_student_info_order_by_math_score()
                case '0':  # 0) 退出班级
                    return
                case _:
                    print('不存在的选项，请重新输入')
                    import time
                    time.sleep(2)  # 让程序睡眠2秒

if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    cr1.student_manager()
```

11. 班级排名的功能的实现

实现方法

- 学校相关模块：`school.py`

```
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def list_class_room_order_by_average_score():
    '''列出班级排名（平均成绩高 - 低）'''
```

```
    def get_avg_score(class_r):
        return class_r.get_average_score()
    result = sorted(class_rooms, key=get_avg_score, reverse=True)
    print('+------+------------+--------+')
    print('| 序号 |  班级名称  | 平均分 |')
    print('+------+------------+--------+')
    number = 1
    for cr in result:
        print('| %4d | %s |  %5.1f |' % (
            number,
            tools.center_to_display_width(cr.class_name, 10),
            cr.get_average_score()
        ))
        number += 1
    if len(result) :
        print('+------+------------+--------+')

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
            case '3':  # 3) 进入管理班级
                enter_class_manager()
            case '4':  # 4) 列出所用班级
                list_all_class_room()
            case '5':  # 5) 列出班级排名（平均成绩高 - 低）
                list_class_room_order_by_average_score()
            case '6':  # 6) 保存班级信息
...

if __name__ == '__main__':
    from student import Student
    cr1 = ClassRoom('一年一班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    class_rooms.append(cr1)
    cr2 = ClassRoom('一年2班')
    cr2.student.append(Student('张3', 7, 1))
    cr2.student.append(Student('李4', 8, 2))
    cr2.student.append(Student('王5', 9, 3))
    cr2.student.append(Student('赵7', 6, 4))
    class_rooms.append(cr2)
    cr3 = ClassRoom('一年三班')
    cr3.student.append(Student('aaa', 100, 99))
    cr3.student.append(Student('bbb', 98, 98))
```

```
    cr3.student.append(Student('cccc', 99, 97))
    class_rooms.append(cr3)
    cr4 = ClassRoom('一年4班')
    cr4.student.append(Student('xxx', 60, 80))
    cr4.student.append(Student('yyy', 70, 50))
    class_rooms.append(cr4)
    class_manager()
```

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def get_average_score(self):
        if len(self.student) == 0:
            return 0
        total_score = 0
        for a_student in self.student:
            total_score += a_student.chinese_score + a_student.math_score
        return total_score / (len(self.student) * 2)

...
```

12. 保存班级信息功能的实现

csv文件格式

文件名:

- students.csv

格式

班级名称	学生姓名	语文成绩	数学成绩
一年一班	张三	100	99
一年一班	李四	98	97
一年2班	王五	96	95
一年2班	赵六	94	93

实现方法

- 学校相关模块：`school.py`

```
import csv
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def save_to_csv_file(path_name='students.csv'):
    '''保存班级信息'''
    header = ['班级名称', '学生姓名', '语文成绩', '数学成绩']
    file = open(path_name, 'w')
    writer = csv.writer(file)
    writer.writerow(header)
    for cr in class_rooms:
        cr.writer_to_csv_writer(writer)

    file.close()

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
            case '3':  # 3) 进入管理班级
                enter_class_manager()
            case '4':  # 4) 列出所用班级
                list_all_class_room()
            case '5':  # 5) 列出班级排名（平均成绩高 - 低）
                list_class_room_order_by_average_score()
            case '6':  # 6) 保存班级信息
                save_to_csv_file('students.csv')
```

```
            case '7':  # 7) 加载班级信息
...

if __name__ == '__main__':
    from student import Student
    cr1 = ClassRoom('一年一班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    class_rooms.append(cr1)
    cr2 = ClassRoom('一年2班')
    cr2.student.append(Student('张3', 7, 1))
    cr2.student.append(Student('李4', 8, 2))
    cr2.student.append(Student('王5', 9, 3))
    cr2.student.append(Student('赵7', 6, 4))
    class_rooms.append(cr2)
    cr3 = ClassRoom('一年三班')
    cr3.student.append(Student('aaa', 100, 99))
    cr3.student.append(Student('bbb', 98, 98))
    cr3.student.append(Student('cccc', 99, 97))
    class_rooms.append(cr3)
    cr4 = ClassRoom('一年4班')
    cr4.student.append(Student('xxx', 60, 80))
    cr4.student.append(Student('yyy', 70, 50))
    class_rooms.append(cr4)
    class_manager()
```

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def writer_to_csv_writer(self, csv_writer):
        '''写入一个班级的学生数据'''
        for a_stu in self.student:
            a_item = [
                self.class_name,
                a_stu.name,
                str(a_stu.chinese_score),
                str(a_stu.math_score)
            ]
            csv_writer.writerow(a_item)
```

```
...


if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    print('平均分:', cr1.get_average_score())
    cr1.student_manager()
```

13. 加载班级信息功能的实现

实现方法

- 学校相关模块：`school.py`

```
import csv
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

...

def load_from_csv_file(path_name='students.csv'):
    '''加载班级信息'''
    # 1. 清空之前班级的数据
    class_rooms.clear()
    # 2. 创建空集合，用来保存已经加载的班级名称，避免重复创建班级
    class_room_set = set()
    # 3. 打开文件并创建csv reader
    file = open(path_name)
    reader = csv.reader(file)
    # 4. 读取每一行数据并创建班级对象和学生对象
    first_line_flag = True  # 记录是否是第一行的标记
    for a_item in reader:
        if first_line_flag:
            first_line_flag = False
            continue
        if len(a_item) != 4:
            print('文件', path_name, '不是此程序保存的合法的csv文件,加载失败')
            break
        class_name = a_item[0]
        student_name = a_item[1]
        chinese_score = int(a_item[2])
        math_score = int(a_item[3])
        if class_name not in class_room_set:
```

```
            cur_class_room = ClassRoom(class_name)
            class_room_set.add(class_name)
            class_rooms.append(cur_class_room)
        cur_class_room.add_student_by_info(
            student_name, chinese_score, math_score)

    # 5. 关闭文件
    file.close()

...

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
            case '3':  # 3) 进入管理班级
                enter_class_manager()
            case '4':  # 4) 列出所用班级
                list_all_class_room()
            case '5':  # 5) 列出班级排名（平均成绩高 - 低）
                list_class_room_order_by_average_score()
            case '6':  # 6) 保存班级信息
                save_to_csv_file('students.csv')
            case '7':  # 7) 加载班级信息
                load_from_csv_file('students.csv')
            case '0':  # 0) 退出程序
...
```

- 班级相关模块：`class_room.py`

```
import student
from student import Student
import tools


class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

...

    def add_student_by_info(self, student_name, chinese_score, math_score):
        a_stu = Student(student_name, chinese_score, math_score)
        self.student.append(a_stu)

...
```

```
if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    print('平均分:', cr1.get_average_score())
    cr1.student_manager()
```

14. 项目大结局

添加项目主模块：`main.py`

```
import school

school.load_from_csv_file()
school.class_manager()
```

已经完成的主要模块

学校模块

school.py

主要数据结构和方法：

- class_rooms = []
- 操作班级的函数。

班级模块

class_room.py

主要数据结构和方法：

- 类：ClassRoom，班级数据和操作班级和管理学生的方法。

学生模块

student.py

主要数据结构和方法：

- 类：Student，用来保存学生数据。

主模块

main.py

作用

- 加载 csv 文件里面的数据，初始化管理系统。
- 调用 school 模块里的 class_manager 函数实现班级管理。

以下是项目完整代码（无删减）

- 工具模块：`tools.py`

```
def get_display_width(s):
    return sum([1 if ord(ch) < 128 else 2 for ch in s])


def center_to_display_width(s, width):
    '''将字符串s的左右两端添加空格，使其达到width的显示宽度!
    s = 'ABC中文', width = 10
    返回:' ABC中文  '
    '''
    s_width = get_display_width(s)  # 得到当前字符的显示宽度
    # 计算需要补充的空格数
    fill_blank_count = width - s_width
    return s.center(len(s) + fill_blank_count)
```

- 学校相关模块：`school.py`

```
import csv
from class_room import ClassRoom
import tools

class_rooms = []  # 用于存放班级对象

def add_class_room():
    '''添加班级'''
    class_name = input('请输入班级名称: ')
    # 判断 class_name 不能超过 10 个显示字符的宽度
    if tools.get_display_width(class_name) <= 10:
        cr = ClassRoom(class_name)
        class_rooms.append(cr)
        print('添加班级', class_name, '成功!')
    else:
        print('添加班级失败，班级名太长!')
    import time
    time.sleep(2)

def list_all_class_room():
    print('+------+------------+')
    print('| 序号 |  班级名称  |')
```

```
    print('+------+------------+')
    number = 1
    for cr in class_rooms:
        print('| %4d | %s |' % (number, tools.center_to_display_width(cr.class_n
ame, 10)))
        print('+------+------------+')
        number += 1

def del_class_room():
    '''删除班级'''
    list_all_class_room()
    number = int(input('请输入删除班级的序号: '))
    index = number - 1  # 对应列表的索引
    if 0 <= index < len(class_rooms):
        del class_rooms[index]
        print('删除成功!')
    else:
        print('您输入的序号有误，删除失败!')
    import time
    time.sleep(2)

def enter_class_manager():
    '''进入管理班级界面'''
    list_all_class_room()
    number = int(input('请输入一个要管理班级的序号: '))
    index = number - 1
    if index < 0 or index >= len(class_rooms):
        print('您输入的班级序号有误! ')
        return
    cr = class_rooms[index]
    cr.student_manager()

def list_class_room_order_by_average_score():
    '''列出班级排名（平均成绩高 - 低）'''
    def get_avg_score(class_r):
        return class_r.get_average_score()
    result = sorted(class_rooms, key=get_avg_score, reverse=True)
    print('+------+------------+--------+')
    print('| 序号 |  班级名称  | 平均分 |')
    print('+------+------------+--------+')
    number = 1
    for cr in result:
        print('| %4d | %s |  %5.1f |' % (
            number,
            tools.center_to_display_width(cr.class_name, 10),
            cr.get_average_score()
        ))
        number += 1
    if len(result) :
        print('+------+------------+--------+')

def save_to_csv_file(path_name='students.csv'):
    '''保存班级信息'''
    header = ['班级名称', '学生姓名', '语文成绩', '数学成绩']
    file = open(path_name, 'w')
    writer = csv.writer(file)
```

```
    writer.writerow(header)
    for cr in class_rooms:
        cr.writer_to_csv_writer(writer)
    file.close()

def load_from_csv_file(path_name='students.csv'):
    '''加载班级信息'''
    # 1. 清空之前班级的数据
    class_rooms.clear()
    # 2. 创建空集合，用来保存已经加载的班级名称，避免重复创建班级
    class_room_set = set()
    # 3. 打开文件并创建csv reader
    file = open(path_name)
    reader = csv.reader(file)
    # 4. 读取每一行数据并创建班级对象和学生对象
    first_line_flag = True  # 记录是否是第一行的标记
    for a_item in reader:
        if first_line_flag:
            first_line_flag = False
            continue
        if len(a_item) != 4:
            print('文件', path_name, '不是此程序保存的合法的csv文件,加载失败')
            break
        class_name = a_item[0]
        student_name = a_item[1]
        chinese_score = int(a_item[2])
        math_score = int(a_item[3])
        if class_name not in class_room_set:
            cur_class_room = ClassRoom(class_name)
            class_room_set.add(class_name)
            class_rooms.append(cur_class_room)
        cur_class_room.add_student_by_info(
            student_name, chinese_score, math_score)

    # 5. 关闭文件
    file.close()

def show_school_menu():
    print('        xxxx小学信息管理系统')
    print('+-----------------------------------+')
    print('| 1) 添加班级                        |')
    print('| 2) 删除班级                        |')
    print('| 3) 进入管理班级                    |')
    print('| 4) 列出所用班级                    |')
    print('| 5) 列出班级排名（平均成绩高 - 低）|')
    print('| 6) 保存班级信息                    |')
    print('| 7) 加载班级信息                    |')
    print('| 0) 退出程序                        |')
    print('+-----------------------------------+')

def class_manager():
    '''此函数用来管理班级数据'''
    while True:
        show_school_menu()
        sel = input('请选择: ')
```

```
        match sel:
            case '1':  # 1) 添加班级
                add_class_room()
            case '2':  # 2) 删除班级
                del_class_room()
            case '3':  # 3) 进入管理班级
                enter_class_manager()
            case '4':  # 4) 列出所用班级
                list_all_class_room()
            case '5':  # 5) 列出班级排名（平均成绩高 - 低）
                list_class_room_order_by_average_score()
            case '6':  # 6) 保存班级信息
                save_to_csv_file('students.csv')
            case '7':  # 7) 加载班级信息
                load_from_csv_file('students.csv')
            case '0':  # 0) 退出程序
                return
            case _:
                print('不存在的选项，请重新输入')
                import time
                time.sleep(2)  # 让程序睡眠2秒


if __name__ == '__main__':
    from student import Student
    cr1 = ClassRoom('一年一班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
    class_rooms.append(cr1)
    cr2 = ClassRoom('一年2班')
    cr2.student.append(Student('张3', 7, 1))
    cr2.student.append(Student('李4', 8, 2))
    cr2.student.append(Student('王5', 9, 3))
    cr2.student.append(Student('赵7', 6, 4))
    class_rooms.append(cr2)
    cr3 = ClassRoom('一年三班')
    cr3.student.append(Student('aaa', 100, 99))
    cr3.student.append(Student('bbb', 98, 98))
    cr3.student.append(Student('cccc', 99, 97))
    class_rooms.append(cr3)
    cr4 = ClassRoom('一年4班')
    cr4.student.append(Student('xxx', 60, 80))
    cr4.student.append(Student('yyy', 70, 50))
    class_rooms.append(cr4)
    class_manager()
```

- 班级相关模块：`class_room.py`

```
from student import Student
import tools
```

```
class ClassRoom:
    '''班级类型'''
    def __init__(self, class_name):
        self.class_name = class_name  # 班级名称
        self.student = []  # 保存学生信息

    def add_student(self):
        student_name = input('请输入学生姓名: ')
        if tools.get_display_width(student_name) >= 20:
            print('学生的名字太长，添加失败!')
            return
        chinese_score = int(input('请输入学生的语文成绩: '))
        if chinese_score < 0 or chinese_score > 100:
            print('学生成绩不在合法范围内，添加失败!')
            return
        math_score = int(input('请输入学生的数学成绩: '))
        if math_score < 0 or math_score > 100:
            print('学生成绩不在合法范围内，添加失败!')
            return
        stu = Student(student_name, chinese_score, math_score)
        self.student.append(stu)
        print('添加学生', student_name, '成功!')

    def modify_chinese_score(self):
        '''修改语文成绩功能'''
        self.list_all_student_info(self.student)
        number = int(input('请输入要修改语文成绩的学生的序号: '))
        index = number - 1
        if index < 0 or index >= len(self.student):
            print('您输入的序号有误，修改失败!')
            return
        a_student = self.student[index]
        new_score = int(input('请输入' + a_student.name + '的新的语文成绩: '))
        a_student.chinese_score = new_score
        print('修改', a_student.name, '的语文成绩成功! ')

    def modify_math_score(self):
        '''修改数学成绩功能'''
        self.list_all_student_info(self.student)
        number = int(input('请输入要修改数学成绩的学生的序号: '))
        index = number - 1
        if index < 0 or index >= len(self.student):
            print('您输入的序号有误，修改失败!')
            return
        a_student = self.student[index]
        new_score = int(input('请输入' + a_student.name + '的新的数学成绩: '))
        a_student.math_score = new_score
        print('修改', a_student.name, '的数学成绩成功! ')

    def del_student(self):
        '''删除学生信息'''
        self.list_all_student_info(self.student)
        number = int(input('请选择要删除学生的序号: '))
        index = number - 1
```

```
        if index < 0 or index >= len(self.student):
            print('您输入的序号有错，删除失败!')
            return
        del self.student[index]
        print('删除学生成功!')

    def list_all_student_info(self, student_list):
        '''显示所有学生的信息'''
        print('+------+----------------------+--------+--------+')
        print('| 序号 |         姓名         |语文成绩|数学成绩|')
        print('+------+----------------------+--------+--------+')
        number = 1
        for stu in student_list:
            print('| %4d | %s |  %4d  |  %4d  |' % (
                number, tools.center_to_display_width(stu.name, 20),
            stu.chinese_score, stu.math_score))
            number += 1
        if len(student_list):
            print('+------+----------------------+--------+--------+')

    def list_student_info_order_by_chinese_score(self):
        '''按语文成绩从高到低显示学生成绩'''
        def get_chinese_score(a_student):
            return a_student.chinese_score
        result = sorted(self.student, key=get_chinese_score, reverse=True)
        print('按语文成绩排序的列表')
        self.list_all_student_info(result)

    def list_student_info_order_by_math_score(self):
        '''按数学成绩从高到低显示学生成绩'''
        result = sorted(
            self.student,
            key=lambda a_stu: a_stu.math_score,
            reverse=True)
        print('按数学成绩排序的列表')
        self.list_all_student_info(result)

    def get_average_score(self):
        if len(self.student) == 0:
            return 0
        total_score = 0
        for a_student in self.student:
            total_score += a_student.chinese_score + a_student.math_score
        return total_score / (len(self.student) * 2)

    def writer_to_csv_writer(self, csv_writer):
        '''写入一个班级的学生数据'''
        for a_stu in self.student:
            a_item = [
                self.class_name,
                a_stu.name,
                str(a_stu.chinese_score),
                str(a_stu.math_score)
            ]
            csv_writer.writerow(a_item)
```

```
    def add_student_by_info(self, student_name, chinese_score, math_score):
        a_stu = Student(student_name, chinese_score, math_score)
        self.student.append(a_stu)

    def show_class_menu(self):
        '此函数用来显示操作菜单'
        print(f'        {self.class_name}-班级管理')
        print('+-------------------------------------+')
        print('| 1) 添加学生                          |')
        print('| 2) 修改学生的语文成绩                |')
        print('| 3) 修改学生的数学成绩                |')
        print('| 4) 删除学生                          |')
        print('| 5) 列出所有学生的成绩                |')
        print('| 6) 按语文成绩从高到低显示学生成绩 |')
        print('| 7) 按数学成绩从高到低显示学生成绩 |')
        print('| 0) 退出班级                          |')
        print('+-------------------------------------+')

    def student_manager(self):
        '''此函数用来学生数据'''
        while True:
            self.show_class_menu()
            sel = input('请选择: ')
            match sel:
                case '1':  # 1) 添加学生
                    self.add_student()
                case '2':  # 2) 修改学生的语文成绩
                    self.modify_chinese_score()
                case '3':  # 3) 修改学生的数学成绩
                    self.modify_math_score()
                case '4':  # 4) 删除学生
                    self.del_student()
                case '5':  # 5) 列出所有学生的成绩
                    self.list_all_student_info(self.student)
                case '6':  # 6) 按语文成绩从高到低显示学生成绩
                    self.list_student_info_order_by_chinese_score()
                case '7':  # 7) 按数学成绩从高到低显示学生成绩
                    self.list_student_info_order_by_math_score()
                case '0':  # 0) 退出班级
                    return
                case _:
                    print('不存在的选项，请重新输入')
                    import time
                    time.sleep(2)  # 让程序睡眠2秒


if __name__ == '__main__':
    cr1 = ClassRoom('一年1班')
    cr1.student.append(Student('张三', 100, 61))
    cr1.student.append(Student('李四', 70, 81))
    cr1.student.append(Student('王五', 90, 71))
    cr1.student.append(Student('赵六', 80, 91))
```

```
    print('平均分:', cr1.get_average_score())
    cr1.student_manager()
```

- 学生相关模块：`student.py`

```
class Student:
    '''学生类型'''
    def __init__(self, name, chinese, math):
        self.name = name
        self.chinese_score = chinese
        self.math_score = math
```

优缺点

优点

- 模块化设计，设计思路清晰。
- 代码量少，功能齐全。
- 使用类和对象的设计，结构简单。
- 数据能够长期存储。
- 数据用excel可读。

缺点

- 学生只有两门成绩，难于扩展。
- 只适合一个学校的单机使用。
- 如果班级里面没有学生，则保存是会丢失班级信息，导致加载时缺少班级信息。
- 班级重名，保存后加载会合并班级。
- 没有异常处理机制，程序崩溃，数据丢失。
- 模块太多，应当封装成为包。
- 对象的属性没有封装，不安全。

缺点的改进型存储方法

班级表:

classes.csv

格式

班级ID	班级名称
1	一年一班
2	一年2班

学生表

student.csv

格式

班级ID	学生姓名	语文成绩	数学成绩
1	张三	100	99
1	李四	98	97
2	王五	96	95
2	赵六	94	93

改进为关系型数据库存储

班级表:

classes.csv

班级ID	班级名称
1	一年一班
2	一年2班

学生表

student.csv

学生ID	学生姓名	班级ID
1	张三	1
2	李四	1
3	王五	2
4	赵六	2

课程表

subject.csv

课程ID	课程
1	语文
2	数学

成绩表

score.csv

学生ID	课程ID	成绩
1	1	100
1	2	99
2	1	98
2	2	97
...	...	...

总结

Python编程语言（基础篇）总结

共计 19 章

章节	内容	章节	内容
1	python开发环境搭建	11	集合
2	python语法基础	12	函数
3	字符串	13	类和对象
4	数字	14	面向对象编程
5	布尔类型	15	模块
6	分支语句	16	包
7	循环语句	17	文件
8	列表	18	内置函数
9	元组	19	校园信息管理系统-项目
10	字典		

以下是上述章节学过的内容。

表达式（7/12）

名称	符号
幂运算	**
一元算术运算	-（负号）、+（正号）
二元算术运算	+（加法）、-（减法）、*、/、//、%
比较运算	<、>、==、<=、>=、!=、is、is not、in、not in
布尔运算	or、and、not
条件表达式	if -- else
lambda 表达式	lambda

简单语句（9/14）

语句	示例
表达式语句	print("xxxx")
赋值语句	z = 100、x,y=a,b、 a=b=c=x
pass 语句	pass
del 语句	del x,dict[key],list[index],obj.name
return 语句	return 表达式
break 语句	break
continue 语句	continue
import 语句	import xxx、from yyy import zzz、from yyy import *
global 语句	global

复合语句（6/10）

语句	示例
if 语句	if expression: pass; elif expression: pass; else: pass
while 语句	while expression: pass
for 语句	for variable in iterator: pass
match 语句	match expression: case value1: pass;case value2:pass
函数定义	def function_name(...):pass
类定义	class class_name(fathor_class1, fathor_class2):pass

标准库：内置函数(37/71)

类型	函数
基本输入输出函数	print()、input()
类的构造函数	int()、bool()、float()、complex()、str()、list()、tuple()、dict()、set()、frozenset()
容器统计函数	len()、sum()、max()、min()
布尔运算函数	any()、all()
整数生成器函数	range()
类和对象函数	id()、dir()、isinstance()、issubclass()、type()、super()
文件打开函数	open()
帮助函数	help()
数学运算函数	abs()、pow()、round()、divmod()
整数进制转换函数	bin()、oct()、hex()
字符编码函数	chr()、ord()
排序函数	sorted()

标准库：内置常量（3/6）

- False
- True
- None

标准库：内置类型

名称	类型
数字类型	int, float, complex
布尔类型	bool
序列类型	list, tuple, range
文本序列类型	str
集合类型	set, frozenset
映射类型	dict
其他内置类型	模块、类、函数、方法

标准库：内置模块

名称	模块
数学模块	math
随机模块	random

高级篇-预告

Python编程语言（高级篇）

内容

- 异常
- 字节串和字节数组
- 文件（高级)
- 生成器和迭代器
- 迭代工具函数
- 函数式编程
- 函数（高级）
- 函数参数标注
- 闭包
- 装饰器
- 类和对象（高级）
- 面向对象（高级）
- 特性属性
- 内置函数重载
- 运算符重载
- 元类

适应人群

- 高等计算机爱好者
- 计算机专业的大学生
- Web开发者
- 人工智能算法工程师
- Python高级软件工程师

以下是高级篇要学习的内容。

简单语句（高级篇）

语句	示例
assert 语句	assert expression
yield 语句	yield expression
raise 语句	raise expression
nonlocal 语句	nonlocal identifier

复合语句（高级篇）

语句	示例
try 语句	try--except、try-finally
with 语句	with expression as target
函数定义(高级)	装饰器 def function_name(arg_list) -> return_type: pass
类定义(高级)	装饰器 class classname(inheritance): pass

标准库：内置函数(24/71)-（高级篇）

类型	函数
迭代器函数	iter()、next()、reversed()
类的构造函数	bytes()、bytearray()、object()、type()
类型判断函数	callable()
类成员相关函数	classmethod()、staticmethed()、super()
对象属性函数	setattr()、getattr()、hasattr()、delattr()
作用域函数函数	globals()、locals()
编译函数函数	eval()、exec()、compile()
高阶函数函数	enumerate()、filter、map()、
特性属性	property()
特殊对象函数	slice()

标准库：内置常量（2/6）-（高级篇）

- Ellipsis 省略号字面值 "..."
- `__debug__`

标准库：内置类型（高级篇）

类型	说明
二进制序列类型	bytes, bytearray
迭代器类型	`__iter__`, `__next__`
上下文管理器类型	`__enter__`, `__exit__`