

第 1 章 环境搭建和错误排查

都说万事开头难，对于刚刚步入学习编程的初学者来说，搭建开发环境往往是初学者们面临的最头疼问题。总会因为各种原因，不能顺利安装。本章将详细讲解搭建 Python 开发环境和 IDE 工具过程中的常见错误排查知识。

1.1 Windows 环境的安装问题

1.1.1 注意下载版本

(1) 登录 Python 官方网页，单击顶部导航中的

“Downloads” 链接，会显示如图 1-1 所示的页面。官网会自动检测当前的计算机系统，并自动提供适应于当前计算机系统的最新正式版本。如图 1-1 所示，适应于当前 Windows 系统的最新正式版本是 3.6.5。

(2) 对于初学者来说，最简单的做法是直接单击“Python 3.6.5”按钮下载 Python，然后按照书中介绍的方法安装即可。



图 1-1 单击顶部导航中的“Downloads”链接

(3) 有的读者不想安装最新正式版本，而是想安装其他版本。此时在图 1-1 所示的界面中单击左侧的“Windows”链接，在弹出的下载页面中提供了 Windows 系统可用的、多种 Python 版本的下载链接，如图 1-2 所示。

(3) 其中注意 x86-64 是指 64 位系统，x86 是指 32 位系统。我们必须根据自己计算机系统的实际情况选择正确

的 Python 下载并安装。查看计算机系统的方法非常简单，只需鼠标右键单击桌面中的“我的电脑”或“此电脑”，在弹出的快捷菜单中选择“属性”，在弹出的界面中会显示当前计算机系统的基本信息。例如笔者计算机的系统信息界面效果如图 1-3 所示，这说明笔者的计算机是 64 位系统。这就必须在图 1-2 所示的界面中选择带有“x86-64”标识的链接进行下载，只有这样才能下载 64 位的 Python。

1.1 Windows 环境的安装问题

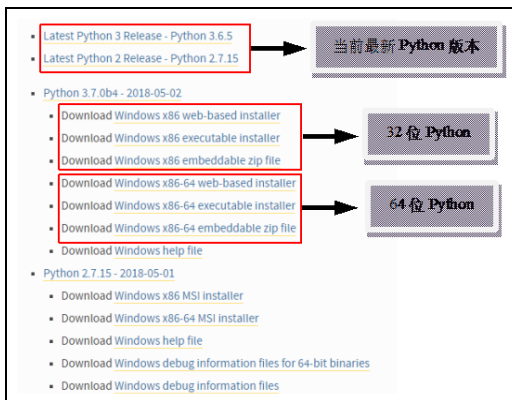


图 1-2 当前 Windows 系统可用的 Python 版本

[查看有关计算机的基本信息](#)

Windows 版本

Windows 10 专业版

© 2016 Microsoft Corporation。保留所有权利。

系统

处理器:	Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz 2.50 GHz
已安装的内存(RAM):	16.0 GB
系统类型:	64 位操作系统, 基于 x64 的处理器
笔和触摸:	没有可用于此显示器的笔或触控输入

图 1-3 当前电脑的 Windows 系统信息

(4) 如果笔者错误地下载了带有 “x86” 标识的链接的 Python, 则会下载一个 32 位的 Python, 这会导致 32 位 Python 和 64 位 Windows 不兼容, 造成 Python 安装失败。

1.1.2 一定要使用 “管理员权限” 安装

无论是 Windows 7, 还是 Windows 10 系统, 在打开 exe 安装程序时不要用双击鼠标左键的方式打开。正确的安装方式是单击鼠标右键, 然后在弹出的菜单命令中选择 “以管理员身份运行” 后进行安装, 如图 1-4 所示。

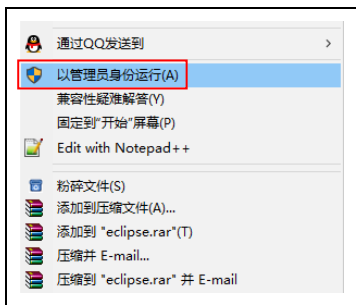


图 1-4 选择“以管理员身份运行”

1.1.3 不要忘记勾选两个复选框

在官网成功下载 Python 后，会得到一个“.exe”格式的可执行文件，双击此文件开始安装。在第一个安装界面中，一定要勾选如图 1-5 所示的下方的两个复选框。

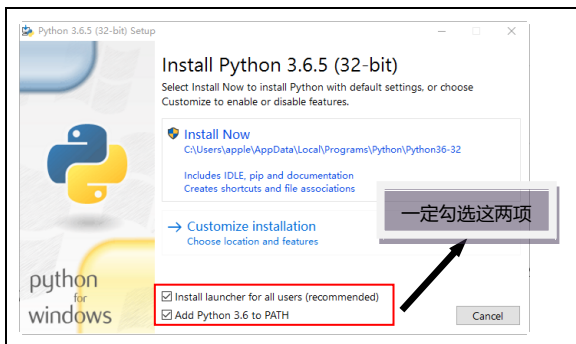


图 1-5 勾选下方的两个复选框

1.1.4 使用“以管理员身份运行”打开命令提示符

我们不但在安装 Python 时用“以管理员身份运行”的方式，在“开始”菜单中打开命令提示符（平常说的 cmd 命令）时也必须用“以管理员身份运行”方式。正确做法是依次单击“开始”→“命令提示符”→“更多”→“以

管理员身份运行”，如图 1-6 所示。

这一点非常重要，因为在 Windows 系统中经常需要进入到命令提示符界面安装 Python 第三方库，如果不使用“以管理员身份运行”打开命令提示符，在安装这些第三方库时会提示没有权限的错误。

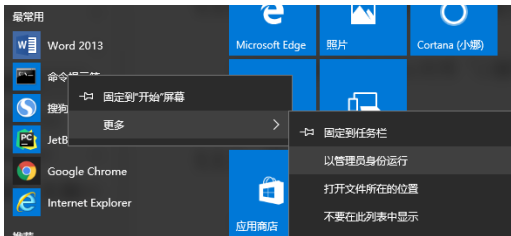


图 1-6 以管理员身份运行 cmd

1.1.5 解决“api-ms-win-crt-runtime-l1-1-0.dll 丢失”问题

安装 Python 成功后,有些读者在启动 Python 时可能会遇到“api-ms-win-crt-runtime-l1-1-0.dll 丢失”的错误提示,如图 1-7 所示。



图 1-7 错误提示

图 1-7 提示中的 api-ms-win-crt-runtime 是 MFC 的运行时环境的库,要想在 Windows 系统编译 Python,需要使用微软的 Visual Studio C++编译器,底层也会用到微软提供的

C++库和 runtime 库。我们只需安装 Visual C++ Redistributable for Visual Studio 2015 组件即可解决这个问题，这个组件可以在微软官网中下载，如图 1-8 所示。



图 1-8 下载 Visual C++ Redistributable for Visual Studio 2015

1.1.6 解决“2502/2503 错误”问题

有些读者在 Windows 10 中安装 Python 时会出现“2502/2503 错误”，如图 1-9 所示。

其原因是 “C:\Windows\Temp” 文件夹的 NTFS 权限错误，解决方法是将此文件夹的 user 权限改为完全控制，如图 1-10 所示。

1.1 Windows 环境的安装问题



图 1-9 “2502/2503 错误”

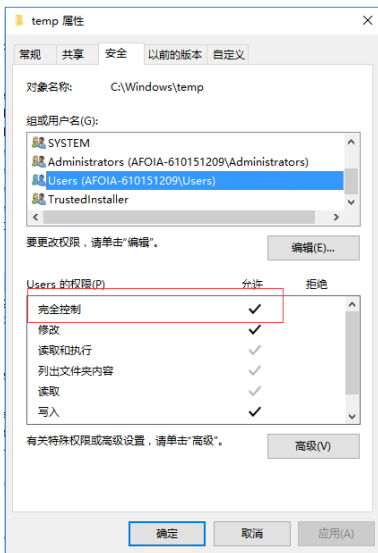


图 1-10 选中“完全控制”

1.2 苹果系统环境的安装问题

1.2.1 在苹果 macOS X 系统中需要注意版本问题

虽然在苹果的 macOS X 系统中内置安装了 Python，但是通常不一定是最新版本，其中有很多计算机内置安装的是 Python 2.7。苹果系统下可以使用 Homebrew 工具来安装 Python。Homebrew 是一款 mac OS 平台下的软件包管理工具，拥有安装、卸载、更新、查看、搜索等很多实用的功能。通过简单的一条指令，就可以实现包管理，而不用用户关心各种依赖和文件路径的情况，十分方便、快捷。

打开 Homebrew, 如果要安装 Python 2.7, 可以通过如下命令实现:

```
brew install python
```

如果想安装使用 Python 3, 只需要将上面命令中的 python 替换成 python3 即可:

```
brew install python3
```

如果想安装某一个具体版本的 Python, 首先需要查看我们可以安装哪些 Python, 通过下面的命令可以在 Homebrew 上搜索可以安装的 Python 版本:

```
brew search python
```

通过上述命令会列出可以安装的全部 Python 版本, 然

后通过如下命令可以安装指定版本的 Python：

```
brew install python版本
```

1.2.2 注意 64 位和 32 位操作系统

如果读者不使用 Homebrew 工具安装 Python，而是想从 Python 官网下载并安装。同前面的 Windows 系统一样，此时读者也需要注意自己使用的 macOS X 系统的位数。苹果系统是一个比较特殊的系统，默认为 64 位，但是可以使用 32 位启动。所以，Python 为 macOS X 提供了如下两类安装包。

- ❑ 64 位安装包：安装在 64 位 macOS X 系统中。
- ❑ 同时兼容 32 位和 64 位的安装包：可以安装在 32

位或 64 位 macOS X 系统中，主要考虑到极少数使用 32 位内核启动的用户。

如果笔者没有启动 32 位 macOS X 的需求，建议安装 64 位的 Python。在 Python 官方页面中提供了 macOS X 系统可用的、多种 Python 版本的下载链接，如图 1-11 所示。

Python Releases for macOS X



图 1-11 Python 官网 macOS X 下载页面

1.3 Linux 环境的安装问题

1.3.1 勿动系统自带的 Python 2.7

在 Linux 系统中内置安装了 Python 2.7，读者在安装

Python 3 时移动不要删除或修改原有的 Python 2.7。这是因为一些软件程序依赖目前的 Python 2.7 环境，例如比较重要的 yum。

1.3.2 安装成功的前提是安装依赖包

在 Linux 系统安装 Python 3 之前一定要安装依赖包，

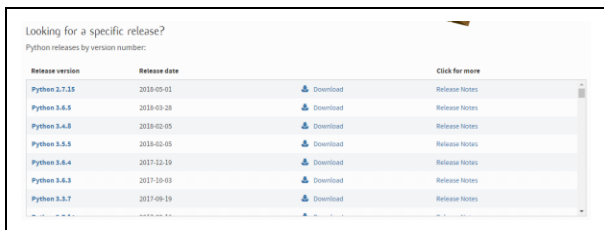
具体命令如下所示：

```
yum -y install zlib-devel bzip2-devel openssl-devel  
ncurses-devel sqlite-devel readline-devel tk-devel gdbm-  
devel db4-devel libpcap-devel xz-devel
```

1.3.3 安装指定版本的 Python

(1) 在 Python 官网页面中列出了各个版本的信息，如

图 1-12 所示。



Looking for a specific release?
Python releases by version number:

Release version	Release date	Click for more	
Python 3.7.15	2019-09-01	Download	Release Notes
Python 3.6.9	2018-09-28	Download	Release Notes
Python 3.4.8	2018-02-05	Download	Release Notes
Python 3.5.5	2018-02-05	Download	Release Notes
Python 3.6.4	2017-12-19	Download	Release Notes
Python 3.6.3	2017-10-03	Download	Release Notes
Python 3.3.7	2017-09-19	Download	Release Notes

图 1-12 各个版本的 Python

(2) 通过如下命令可以下载 Python 3.6.1 版本的安装包：

```
# wget https://www.python.org/ftp/python/3.6.1/  
Python-3.6.1.tgz
```

在安装时建议安装在一个比较好记的目录中，例如
`/usr/local/python3`：

```
# mkdir -p /usr/local/python3
```

(3) 解压下载好的 Python 包（具体包名因用户下载的

Python 具体版本的不同而不同，例如笔者下载的是 Python 3.6.1，那么这里就是 Python-3.6.1.tgz。

```
tar -zxvf Python-3.6.1.tgz
```

(4) 进入解压后的目录，编译安装。

```
# cd Python-3.6.1
# ./configure --prefix=/usr/local/python3
```

(5) 执行 make 命令：

```
# make
```

(6) 执行 make install 命令：

```
# make install
```

或者：

```
# make && make install
```

(7) 建立 Python 3 的软链接：

```
# ln -s/usr/local/python3/bin/python3/usr/bin/python3
```

(8) 将/usr/local/python3/bin 加入 PATH:

```
# vim ~/.bash_profile
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
# User specific environment and startup programs
PATH=$PATH:$HOME/bin:/usr/local/python3/bin
export PATH
```

(9) 按下 Esc 键, 输入:wq 并按下回车键退出。然后执行下面的命令, 让上一步的修改生效:

```
# source ~/.bash_profile
```

(10) 检查 Python 3 及 pip3 是否正常可用:

```
# python3 -V
Python 3.6.1
# pip3 -V
pip 9.0.1 from /usr/local/python3/lib/python3.6/site-packages (python 3.6)
```

(11) 如果不可用，再创建一下 pip3 的软链接：

```
# ln -s /usr/local/python3/bin/pip3 /usr/bin/pip3
```

1.3.4 解决 pip3 install paramiko 错误

在安装 pip 之前需要安装 setuptools 依赖包，在前面

1.3.2 中已经安装过了，接下来通如下命令安装 pip：

```
wget --no-check-certificate https://pypi.python.org/packages/source/p/pip/pip-8.0.2.tar.gz#md5=3a73c4188f8dbad6a1e6f6d44d117eeb
tar -zxvf pip-8.0.2.tar.gz
cd pip-8.0.2
python3 setup.py build
python3 setup.py install
```

如果出现 pip3 install paramiko 错误，则需要安装 openssl

依赖包解决，具体命令如下所示：

```
yum install openssl
yum install openssl-devel
cd python3.6.1
```

```
make && make install
```


第 2 章 推荐使用第三方 IDE 开发工具：Pycharm

如果说编程是程序员的手艺，那么 IDE 就是程序员吃饭的必备工具。使用 IDE 可以提高程序员编写程序的效率。

IDE 的全称是 Integration Development Environment (集成开发环境)，而 PyCharm 是一款著名的 Python IDE 开发工具，是拥有一整套可以帮助用户在使用 Python 语言开发时提高其效率的工具，具备基本的调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。

此外，该 IDE 提供了一些高级功能，以用于支持 Django

框架下的专业 Web 开发。如果读者具有 Java 开发经验，会发现 PyCharm 和 IntelliJ IDEA 十分相似。如果读者拥有 Android 开发经验，就会发现 PyCharm 和 Android Studio 十分相似。事实也正是如此，PyCharm 不但跟 IntelliJ IDEA 和 Android Studio 外表相似，而且用法也相似。有 Java 和 Android 开发经验的读者可以迅速上手 PyCharm，几乎不用额外的学习工作。

2.1 下载、安装并设置 PyCharm



注意：在安装 PyCharm 之前需要先安装 Python。

PyCharm 是一款收费软件，建议读者尊重版权。

(1) 登录 PyCharm 官方页面，单击顶部中间的
“DOWNLOAD NOW” 按钮，如图 2-1 所示。



图 2-1 PyCharm 官方页面

(2) 在打开的新界面中显示了可以下载如下 PyCharm 的两个版本，如图 2-2 所示。

- ❑ Professional：专业版，可以使用 PyCharm 的全部功能，但是收费。
- ❑ Community：社区版，可以满足 Python 开发的大多数功能，完全免费。



图 2-2 专业版和社区版

并且在上方可以选择操作系统，PyCharm 分别提供了 Windows、macOS 和 Linux 三大主流操作系统的下载版本，并且每种操作系统都分为专业版和社区版两种。

(3) 笔者使用的 Windows 系统专业版，单击 Windows 选项中 Professional 下面的“DOWNLOAD”按钮，在弹出的“下载对话框”中单击“下载”按钮开始下载 PyCharm，如图 2-3 所示。

(4) 下载成功后将会得到一个形似“pycharm-professional-201x.x.x.exe”的可执行文件，鼠标双击打开这个可执行文

件，弹出如图 2-4 所示的欢迎安装界面。

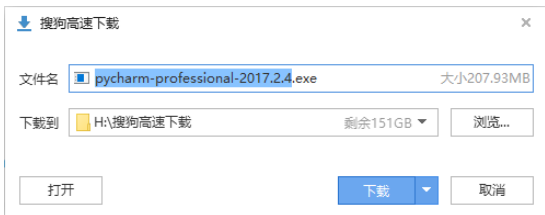


图 2-3 下载 PyCharm

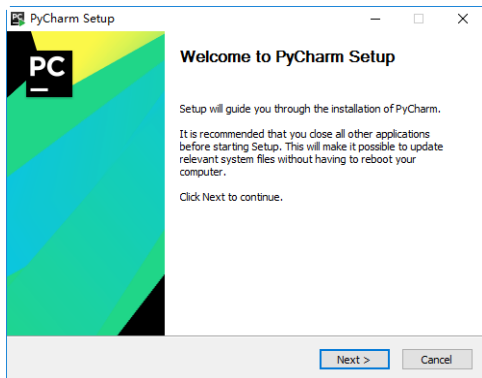


图 2-4 欢迎安装界面

(5) 单击 “Next” 按钮后弹出安装目录界面，在此设置 PyCharm 的安装位置，如图 2-5 所示。

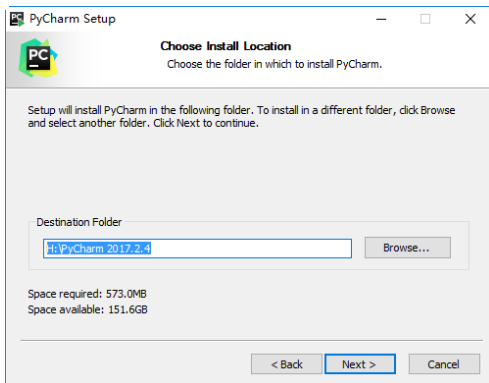


图 2-5 安装目录界面

(6) 单击 “Next” 按钮后弹出安装选项界面，在此根据自己计算机的配置勾选对应的选项。因为笔者使用的是

64 位系统，所以此处勾选 “64-bit launcher” 复选框。然后分别勾选 “create associations (创建关联 Python 源代码文件)” 和 “.py” 前面的复选框，如图 2-6 所示。

(7) 单击 “Next” 按钮后弹出创建启动菜单界面，如图 2-7 所示。

2.1 下载、安装并设置 PyCharm

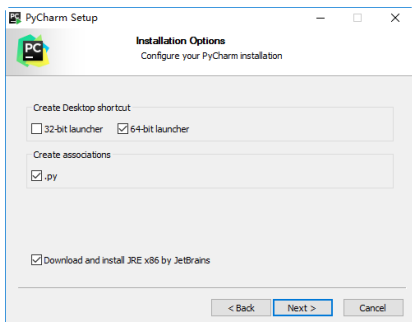


图 2-6 安装选项界面

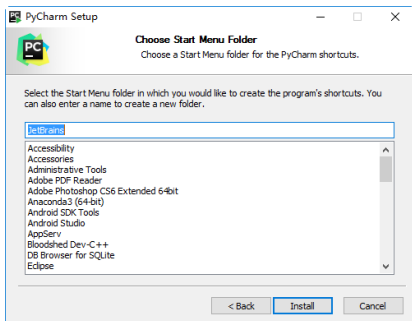


图 2-7 创建启动菜单界面

(8) 单击 “Install” 按钮后弹出安装进度界面，这一步

的过程需要读者耐心等待，如图 2-8 所示。

2.1 下载、安装并设置 PyCharm

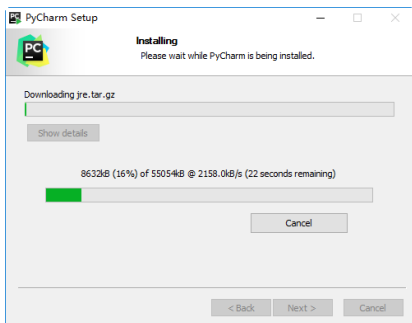


图 2-8 安装进度界面

(9) 安装进度条完成后弹出完成安装界面，如图 2-9 所示。单击“Finish”按钮完成 PyCharm 的全部安装工作。

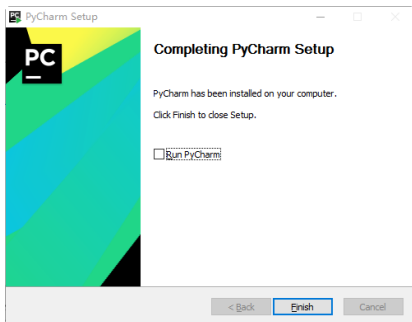


图 2-9 完成安装界面

(10) 单击桌面中快捷方式或“开始”菜单中的对应选项启动 PyCharm, 因为是第一次打开 PyCharm, 所以通常会询问我们是否要导入先前的设置 (默认为不导入)。因为我们是全新安装, 所以这里直接点击“OK”按钮即可。接着 PyCharm 会让我们设置主题和代码编辑器的样式, 读者可

以根据自己的喜好进行设置，例如有 Visual Studio.NET 开发经验的读者可以选择 Visual Studio 风格。完全启动 PyCharm 后的界面效果如图 2-10 所示。

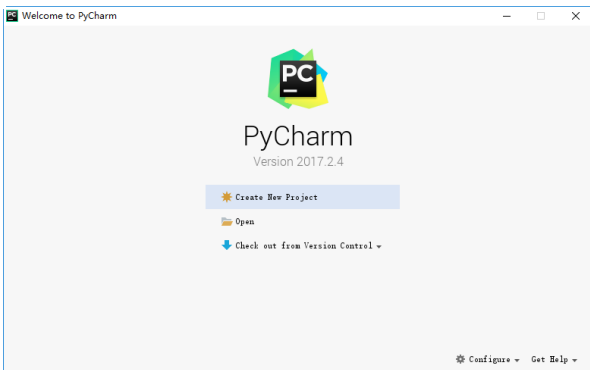


图 2-10 完全启动 PyCharm 后的界面效果

- ❑ 左侧区域面板：列表显示过去创建或使用过的项

目工程，因为我们是第一安装，所以暂时显示为空白。

- ❑ Create New Project 按钮：单击此按钮后将弹出“新建工程”对话框，开始新建项目。
- ❑ Open 按钮：单击此按钮后将弹出“打开”对话框，用于打开已经创建的工程项目。
- ❑ Check out from Version Control 下拉按钮：单击后弹出项目的地址来源列表，里面有 CVS、Github、Git 等常见的版本控制分支渠道。
- ❑ 右下角 Configure：单击后弹出和设置相关的列表，

可以实现基本的设置功能。

- ❑ 右下角 Get Help：单击后弹出和使用帮助相关的列表，可以帮助使用者快速入门。

2.2 使用 PyCharm 创建第一个 Hello Word 程序

(1) 打开 PyCharm，单击图 2-10 中的 “Create New Project” 按钮弹出 “New Project” 界面，单击左侧列表中的 Pure Python 选项，如图 2-11 所示。

- ❑ Location：Python 项目工程的保存路径。
- ❑ Interpreter：选择 Python 的版本，很多开发者在本机中安装了多个版本，例如 Python 2.7、Python 3.5

或 Python 3.6 等。这一功能十分人性化，因为不同版本切换十分方便。

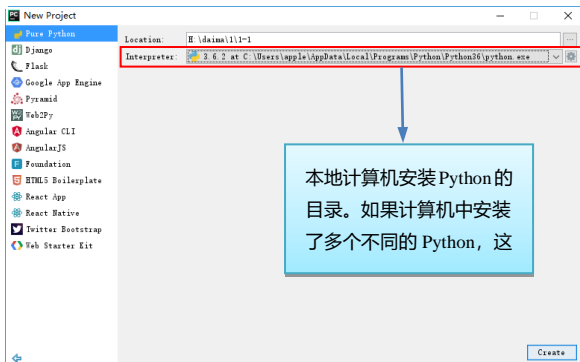


图 2-11 “New Project” 界面

(2) 单击 “Create” 按钮后将在创建一个 Python 工程，如图 2-12 所示。在图 2-12 所示的 PyCharm 工程界面中，依次单击顶部菜单中的 “File” → “New Project” 命令也

可以实现创建 Python 工程功能。

(3) 鼠标右键单击左侧工程名，在弹出的选项中依次选择 “New” → “Python File”，如图 2-13 所示。

第2章 推荐使用第三方 IDE 开发工具：Pycharm

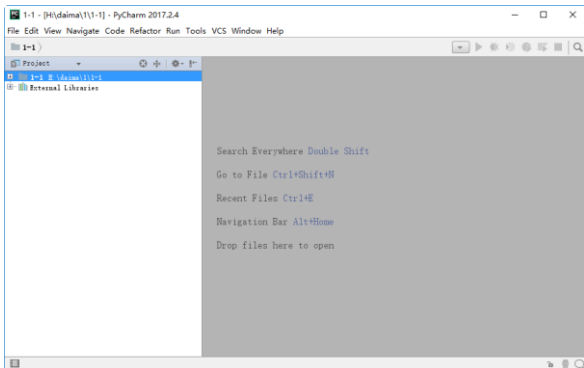


图 2-12 创建的 Python 工程

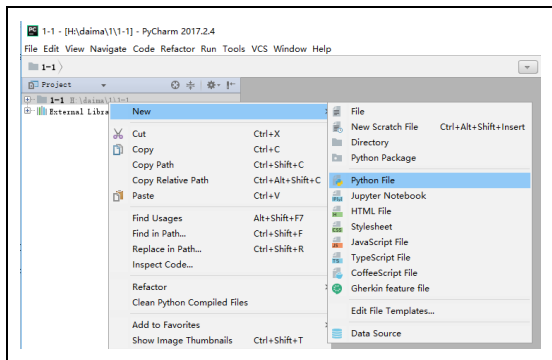


图 2-13 单击“Python File”

(4) 弹出 “New Python file” 对话框界面，在 “Name” 选项中给将要创建的 Python 文件起一个名字，例如 “first”，如图 2-14 所示。

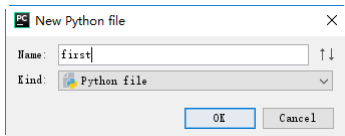


图 2-14 新建 Python 文件

(5) 单击 “OK” 按钮后将会创建一个名为 “first.py” 的 Python 文件，鼠标选择左侧列表中的 “first.py” 文件名，在 PyCharm 右侧代码编辑界面编写 Python 代码，例如编写如下所示的代码，如图 2-15 所示。

第2章 推荐使用第三方 IDE 开发工具：Pycharm

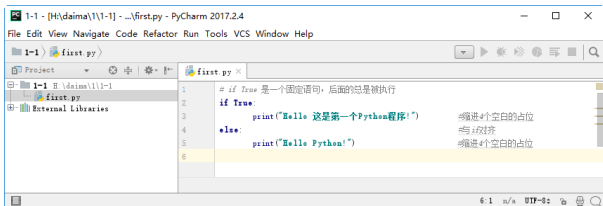


图 2-15 Python 文件 first.py

```
# if True 是一个固定语句, 后面的总是被执行
if True:
    print("Hello 这是第一个Python程序!")
    # 缩进4个空白的占位
    # 与if对齐
print("Hello Python!") # 缩进4个空白的占位
```

(6) 开始运行文件 first.py, 在运行之前会发现 PyCharm

顶部菜单中的运行和调试按钮  都是灰色的, 处于不可用

状态。这时需要我们对控制台进行配置, 其方法是单击运行

旁边的黑色倒三角, 然后单击下面的 “Edit Configurations”

选项 (或者依次单击 PyCharm 顶部菜单中的 “Run” → “Edit

Configurations” 选项), 打开 “Run/Debug Configurations” 配置界面, 如图 2-16 所示。

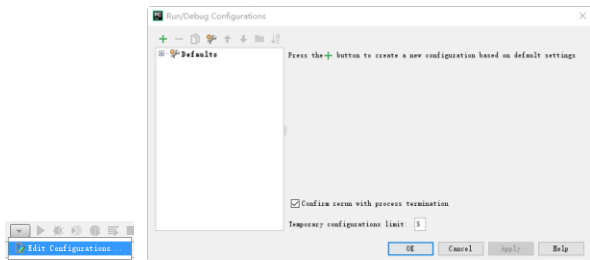


图 2-16 单击 “Edit Configurations” 选项, 打开 Run/Debug 配置界面

(7) 单击左上角的绿色加号, 在弹出的列表中选择 “Python” 选项, 设置右侧界面中的 “Scrip” 选项为我们前面刚刚编写的文件 `first.py` 的路径, 如图 2-17 所示。

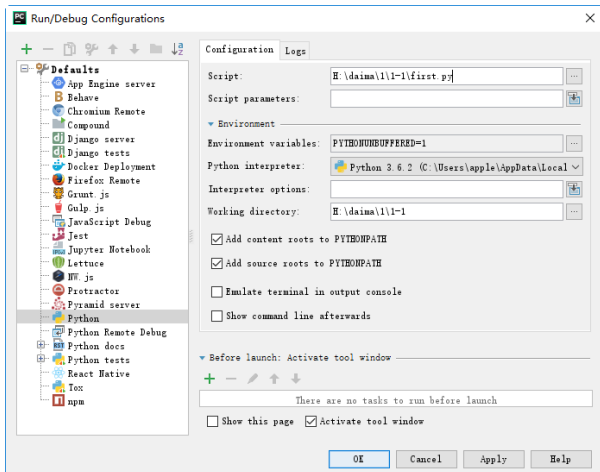


图 2-17 设置“Scrip”选项

(8) 单击“OK”按钮返回 PyCharm 代码编辑界面，此时会发现运行和调试按钮全部处于可用状态，单击后可以运行文件 first.py。也可以单击鼠标右键，选中左侧列表中的

文件名 `first.py`，在弹出的命令中选择 “Run ‘first’ ” 命令来运行文件 `first.py`，如图 2-18 所示。

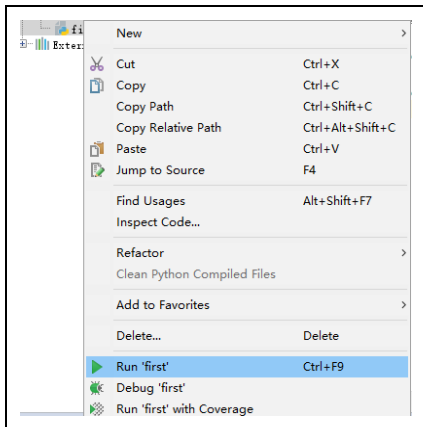


图 2-18 选择 “Run ‘first’ ” 命令运行文件 `first.py`

(9) 在 PyCharm 底部的调试面板中将会显示文件 `first.py` 的执行效果，如图 2-19 所示。

第 2 章 推荐使用第三方 IDE 开发工具：Pycharm

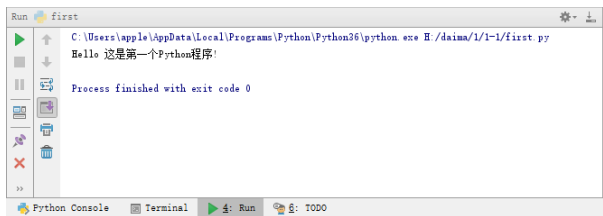


图 2-19 文件 first.py 的执行效果

2.3 常用功能介绍

2.3.1 设置行号

在安装 PyCharm 后，在代码编辑界面中是默认显示行号的。但是有的读者的 PyCharm 不显示行号，此时可以依次单击 “File” → “Settings” → “Editor” → “General” → “Appearance”，在弹出的 “Appearance” 界面中勾选 “Show Line Numbers” 复选框即可实现行号显示功能，如图 2-20 所示。

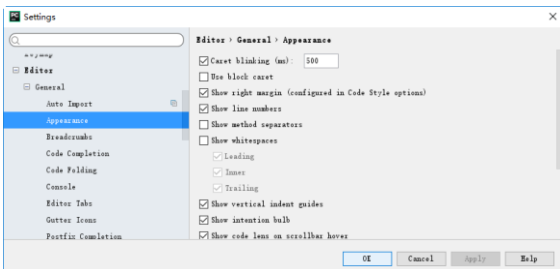




图 2-20 勾选“Show Line Numbers”复选框

2.3.2 断点调试

PyCharm 作为一款经典的 IDE 开发工具，断点调试是必须具备的功能。在 PyCharm 代码编辑界面设置断点的方法十分简单，在某行代码的前面、行号的后面，使用鼠标左键单击一下就可以设置断点。设置断点后会显示一个红色实心圆图像，如图 2-21 所示。



图 2-21 将文件 first.py 中第 3、4 行代码设置为断点

单击 PyCharm 顶部菜单中的图标可以启动断点调试，单击后会运行程序到第一个断点，并在 PyCharm 调试面板中显示该断点之前的变量信息，如图 2-22 所示。单击调试面板中的图标 (Step Over) 或按下 F10 快捷键后可以继续往下运行，将程序运行到下一个断点。

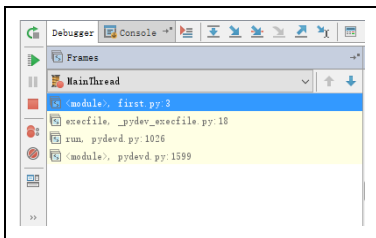


图 2-22 PyCharm 调试面板

第 3 章 Python 程序调试常见 错误排查

在编写并调试 Python 程序的过程中，总会遇到这样或那样的错误，其中绝大多数错误都是由于用户粗心或语法错误引起的。在本部分的内容中，将详细讲解排查 Python 程序错误的知识。

3.1 Python 2 升级 Python 3 发生的错误

在当今市面中，Python 有 2.7 和 Python 3.x 两个大的版本分支。在网络教程、教学文档和出版图书中有很多是

用 Python 2.7 实现的，而我们本书是用 Python 3.6 实现的。当我们直接将 Python 2.7 代码运行在 Python 3.6 环境中时，可能会发生一些语法错误。

3.1.1 print 变成了 print()

在 Python 2 版本中，print 是作为一个语句使用的，在 Python 3 版本中 print() 作为一个函数出现。下面通过两段代码来展示两个版本的区别。

Python 2.x 版本代码如下：

```
>>> i = 1
>>> print 'Python' 'is', 'number', i
Pythonis number 1
```

Python 3.x 版本代码如下：

```
>>> i = 1
>>> print('Python' 'is', 'number', i)
Pythonis number 1
```

也就是说，在 Python 3 版本中，所有的 print 内容必须用小括号括起来。

3.1.2 raw_input 变成了 input

在 Python 2 版本中，输入功能是通过 raw_input 实现的。而在 Python 3 版本中，是通过 input 实现的。下面来看两行代码的区别：

```
name = input('What is your name?\n') #python3版本的代码
name = raw_input('What is your name?\n') # python2版本的代码
```

3.1.3 整数及除法的问题

初学者在编写 Python 程序时，特别是将 Python 2 程序在

Python 3 环境下运行时，很可能会遇到 “TypeError: 'float' object cannot be interpreted as an integer” 错误。例如下面的代码是在 Python 2 运行成功的：

```
batch = 200
for x in range(len(order_nos) / batch + 1):
    # do something
```

其中，order_nos 是订单列表，而在 Python 3 环境下运行时，会提示 “TypeError: 'float' object cannot be interpreted as an integer” 错误，意思是 float 类型不能解释为 int 类型。这是因为在 Python 3 中，int 和 long 统一为 int 类型，int 表示任何精度的整数。在以前的 Python 2 版本中，如果参数是 int 或者是 long 的话，就会返回相除后结果的向

下取整 (floor)，而如果参数是 float 或者是 complex 的话，那么就会返回相除后结果的一个恰当的近似。当使用 int 超过本地整数大小时，不会再导致 OverflowError 异常。long 类型在 Python 3 中已经消失，并且后缀 L 也已经弃用。

下面是两个版本的除法对比：

```
>>>1/2    #Python 2版本中的结果是0
>>>1/2    #Python 3版本中结果是0.5，这样比较合理
```

与之相对应的是，除法也发生了变化，Python 3 中的 `"/"` 总是返回一个浮点数，`"//"` 永远表示向下除法。所以当涉及除法 `"/"` 操作遇到 `"TypeError: 'float' object cannot be`

interpreted as an integer” 错误时，只需将 “/” 修改为 “//” 即可。

3.1.4 异常处理大升级

在 Python 2 程序中，捕获异常的格式如下：

```
except Exception, identifier
```

在 Python 3 程序中，捕获异常的格式如下：

```
except Exception as identifier
```

例如，下面是 Python 2 捕获异常的演示代码：

```
except ValueError, e: # Python 2处理单个异常
except (ValueError, TypeError), e: # Python 2处理
多个异常
```

而下面是 Python 3 捕获异常的演示代码：

```
except ValueError as e: # Python3处理单个异常
except (ValueError, TypeError) as e: # Python3处理多个异常
```

在 Python 2 程序中，抛出异常的格式如下：

```
raise Exception, args
```

在 Python 3 程序中，抛出异常的格式如下：

```
raise Exception(args)
```

例如，下面两行代码演示了两种版本抛出异常的方法：

```
raise ValueError, e      # Python 2.x 的方法  
raise ValueError(e)      # Python 3.x 的方法
```

3.1.5 解决 “NameError: name ‘xrange’ is not defined” 错误提示

这个错误也是版本问题，Python2 使用的是 xrange() 函数，在 Python3 版本被 range() 函数代替。所以在 Python 3 程序中，只需将 xrange 修改为 range 即可解决这个问题。

3.1.6 解决 “name 'reload' is not defined 和 AttributeError: module 'sys' has no att” 错误提示

在 Python 3.6 程序中不能直接使用 reload，要想兼容

Python 2 中的 reload 功能，需要加入如下所示的代码：

```
import importlib
importlib.reload(sys)
```

3.1.7 解决 “python unicode is not defined” 错误提示

在 Python 3 程序中经常会遇到 “python unicode is not defined” 错误提示，这是因为在 Python 3 中已经没有了 unicode 类型，被全新的 str 类型所代替。而 Python 2 中原有的 str 类型，在 Python 3 中被 bytes 所代替。

3.1.8 解决“AttributeError: 'dict' object has no attribute 'has_key’” 错误提示

例如，下面的报错过程：

```
>>> d={}
>>> d.has_key('name')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    d.has_key('name')
AttributeError: 'dict' object has no attribute 'has_key'
```

这是因为在 Python 3 中已经舍弃了 `has_key`，修改方法

是用 `in` 来代替 `has_key`，修改为：

```
>>> d={}
>>> 'name' in d
True
```

3.1.9 解决 “ImportError: No module named urllib2” 错误提示

在 Python 3 中 `urllib2` 已经被 `urllib.request` 替代，所以

解决方法是将 `urllib2` 修改为 `urllib.request`。

3.2 常见错误

3.2.1 解决 “IndentationError:expected an indented bloc” 错误提示

这是一个初学者经常犯的错误，这个错误会让人欲哭无泪！这个错误并不是语法错误的问题，而是用户代码书写规范的问题。因为 Python 是一个对代码缩进非常敏感的语言，整个循环结构可能是依靠缩进的形式来表示的。

初学者最常见的错误就是混用 Tab 和 Space 键实现代码缩进，这是很容易报错的，而且肉眼很难分辨出来。虽然很多 IDE 编辑器可以选择显示空格，但是即便是这样，也很难找到到底哪里有问题。所以建议读者在程序中只使用 Tab 键

实现代码缩进，或者只使用 Space 键实现代码缩进。

另外，上面的报错还有一个原因经常遇到，就是无首行缩进，例如在编写 if 语句时在后面加冒号，如果直接换行，好的代码编辑器会自动首行缩进。但是有些代码编辑器可能没有这个功能，这时需要开发者手动缩进，这最好养成习惯。请大家不要连续敲几次空格键，建议直接按一下 Tab 键即可。

3.2.2 解决 “no module named XX” 错误提示

毫无疑问，这个错误将是读者在学习和开发过程中遇到的最多的错误，没有之一。随着读者开发水平的提高和程序复杂性的提升，将会在程序中用到越来越多的模块和

第三方库。那时候将会经常遇到 “no module named XX”

错误，这个错误的原因是没有安装库 “XX”。当遇到这个

错误的时候，需要使用如下命令安装库 XX：

```
pip install ww
```

3.2.3 解决 “TypeError: 'tuple' object cannot be interpreted as an integer” 错误提示

请看下面的代码：

```
t=('a','b','c')
for i in range(t):
    print(t[i])
```

上述代码会报错：TypeError: 'tuple' object cannot be interpreted as an integer

这是一个典型的类型错误问题，在上述代码中，range() 函数期望的传入参数是整型 (integer)，其但是却传入的入

参为元组 (tuple)。解决方法是将入参元组 t 改为元组个数整型 len(t)类型即可，例如将上述代码中的 range(t)改为 range(len(t))。

3.2.4 解决 “IOError: File not open for writing” 错误提示

这是一个典型的文件操作权限问题，例如下面的演示

代码会爆出这个错误：

```
>>> f=open("hello.py")
>>> f.write("test")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: File not open for writing
```

出错原因是在没有在 open("hello.py")的传入参数中添加读写模式参数 mode，这说明默认打开文件的方式为只读

方式，而在上述代码中需要写入字符操作功能，所以出现权限受限问题，才会报错。解决方法是更改模式 mode，修改为写入模式权限 w+：

```
>>> f=open("hello.py", 'w+')  
>>> f.write("test")
```

3.2.5 解决 “SyntaxError: invalid syntax” 错误提示

这个错误通常是由于忘记在 if、elif、else、for、while、class 和 def 等语句末尾添加冒号 “:” 引起的，例如：

```
if spam == 42 print('Hello!')
```

解决方法是在最后添加冒号 “:”。

还有一种情况也会引发上述错误，错误的使用了=，而不是==。在 Python 程序中，=是赋值操作符，而==是等于

比较操作。

3.2.6 解决“TypeError: 'str' object does not support item assignment” 错误提示

这个错误通常是由于尝试修改 string 的值引起的，string 是一种不可变的数据类型。例如在如下代码中会发生该错误：

```
spam = 'I have a pet cat.'  
spam[13] = 'r' print(spam)
```

修改方法是：

```
spam = 'I have a pet cat.'  
spam = spam[:13] + 'r' + spam[14:] print(spam)
```

3.2.7 解决“TypeError: Can't convert 'int' object to str implicitly” 错误提示

这个错误通常是由于尝试连接非字符串值与字符串引起的，例如在如下代码中会发生该错误：

```
numEggs = 12
print('I have ' + numEggs + ' eggs.')
```

解决方法是修改为：

```
numEggs = 12
print('I have ' + str(numEggs) + ' eggs.')
```

也可以修改为：

```
numEggs = 12
print('I have %s eggs.' % (numEggs))
```

3.2.8 错误的使用类变量

考虑下面的演示过程：

```
>>> class A(object):
    x = 1
>>> class B(A):
    pass
>>> class C(A):
    pass
>>> print(A.x, B.x, C.x)
1 1 1
>>> B.x = 2
>>> print(A.x, B.x, C.x)
1 2 1
>>> A.x = 3
>>> print(A.x, B.x, C.x)
3 2 3
```

我们只修改了 A.x，为什么 C.x 也被改了？在 Python 程序中，类变量在内部当做字典来处理，其遵循常被引用的方法解析顺序 (MRO)。所以在上面的代码中，由于 class C 中的 x 属性没有找到，它会向上找它的基类（尽管 Python 支持多重继承，但上面的例子中只有 A）。换句话说，class C 中没有它自己的 x 属性，其独立于 A。因此，C.x 事实上

是 A.x 的引用。

3.2.9 错误地理解 Python 的作用域

Python 是基于 LEGB 来进行作用于解析的，LEGB 是 Local, Enclosing, Global, Built-in 的缩写。看起来“见文知意”，对吗？实际上，在 Python 中还有一些需要注意的地方，先看下面一段代码：

```
x = 10
def foo():
    x += 1
    print(x)
foo()

Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    foo()
  File "<pyshell#61>", line 2, in foo
    x += 1
UnboundLocalError: local variable 'x' referenced
before assignment
```

上述代码出错的原因是：局部变量 `x` 没有初始值，外部变量 `x` 不能引入到内部。

再看下面列表操作的情况：

```
>>> lst = [1,2,3] # 给列表lst赋值
>>> lst.append(4) # lst后边append一个元素4
>>> lst
[1, 2, 3, 4]
>>> lst += [5] # 两个列表合并
>>> lst
[1, 2, 3, 4, 5]
>>>
def foo1():
    lst.append(6) # 函数会查找外部的lst列表
>>> foo1()
>>> lst
[1, 2, 3, 4, 5, 6]
>>>
def foo2():
    lst += [6] # 合并列表时，不会查找外部列表，让人有
些不可思议吧
>>> foo2()
Traceback (most recent call last):
  File "<pyshell#82>", line 1, in <module>
    foo2()
  File "<pyshell#81>", line 2, in foo2
    lst += [6]
```

```
UnboundLocalError: local variable 'lst' referenced  
before assignment
```

上述代码的出错原因和前面的例子相同，不过更加令人难以捉摸。foo1 没有对 lst 进行赋值操作，而 foo2 做了。要知道，`lst += [5]`是 `lst = lst + [5]`的缩写，我们试图对 lst 进行赋值操作（Python 把他当成了局部变量）。此外，我们对 lst 进行的赋值操作是基于 lst 自身（这再一次被 Python 当成了局部变量），但此时还未定义，因此出错！

第 4 章 Python 常用函数速查

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。函数能提高应用的模块性，和代码的重复利用率。你已经知道 Python 提供了许多内建函数，比如 `print()`。本章将传授 Python 语言中的常用内置模块库函数，这些库函数都是 Python 语言的功能精髓。

4.1 标准内置函数

在当今市面中，Python 有 2.7 和 Python 3.x 两个版本分支。在网络教程、教学文档和出版图书中有很多是用 Python

2.7 实现的，而我们本书是用 Python 3.6 实现的。当我们直接将 Python 2.7 代码运行在 Python 3.6 环境中时，可能会发生一些语法错误。

4.1.1 函数 `abs(x)`

在 Python 程序中，函数 `abs(x)` 的功能是返回参数 “x” 的绝对值，参数 “x” 可以是一个整数或浮点数。如果参数 “x” 是一个复数，则返回其大小。

4.1.2 函数 all(iterable)

在 Python 程序中，函数 all(iterable)的功能是如果可迭代的对象的所有元素全部非空（或者空迭代对象），就返回 True。当参数 "iterable" 的元素都不为 false、"、0 或者 iterable 为空时，则 all(iterable)的值为 True，也就是说只要 iterable 元素有一个为"假"，则为 False，"全 '真' 为 True，有 '假' 为 False"。

函数 all(iterable)主要用于判断列表、元组、字典等对象是否有空元素，比如有 10000 个元素的列表，如果没有提供此函数，需要使用循环来实现，那么计算速度会

比较慢。

4.1.3 函数 any(iterable)

在 Python 程序中，函数 any(iterable)的功能是如果参数 “iterable” 中的任何一个元素为 True，则返回 True。如果参数 “iterable” 为空 (empty)，则返回 False。也就是说，当所有的 iterable 都为假时 any(iterable)为 False。参数 “iterable” 是一个元组或列表。

4.1.4 函数 ascii(object)

在 Python 程序中，函数 ascii(object)的功能是返回一个可打印的对象字符串。当遇到非 ASCII 码时，就会输出

\x、\u 或\U 等字符。与 Python 2 版本里的 repr()函数是等效的。例如在下面的实例文件 as.py 中，演示了使用函数 ascii(object)处理不同参数对象的过程。

4.1.5 函数 bin()

在 Python 程序中，函数 bin()的功能是返回一个整数 int 或者长整数 long int 的二进制表示。使用函数 bin()的语法格式如下所示：

```
bin(x)
```

参数“x”是一个 int 类型或者 long int 类型的数字。返回值将是一个合法的 Python 表达式。如果 x 不是 int 类型的对象，那么就定义一个__index__()方法返回一个整数。

4.1.6 函数 bool()

在 Python 程序中，函数 bool()的功能是将给定的参数转换为布尔类型 True 或 False。如果没有参数，则返回 False。使用函数 bool()的语法格式如下所示。

```
bool([x])
```

参数 “x” 表示要进行转换的参数，将会使用真值测试对 x 进行转换。如果 x 的值为 false 或被省略时返回 False，否则会返回 True。bool 不能进一步进行子类化，其唯一的实例是 False 和 True。

4.1.7 函数 bytearray()

在 Python 程序中，函数 bytearray()的功能是返回一个

新字节数组。这个数组中的元素是可变的，并且每个元素的值范围是： $0 \leq x < 256$ 。函数 `bytearray()` 的语法格式如下所示：

```
class bytearray([source[, encoding[, errors]]])
```

- ❑ 如果 `source` 为整数，则返回一个长度为 `source` 的初始化数组；
- ❑ 如果 `source` 为字符串，则按照指定的 `encoding` 将字符串转换为字节序列；
- ❑ 如果 `source` 为可迭代类型，则元素必须为 `[0, 255]` 中的整数；
- ❑ 如果 `source` 为与 `buffer` 接口一致的对象，则此对

象也可以被用于初始化 bytearray;

- 如果没有输入任何参数, 默认就是初始化数组为 0 个元素。

4.1.8 函数 bytes()

在 Python 程序中, 函数 bytes()的功能是返回一个新的 bytes 对象, 该对象是一个 $0 \leq x < 256$ 区间内的整数不可变序列。函数 bytes()的语法格式如下所示:

```
class bytes([source[, encoding[, errors]])
```

- 如果 source 为整数, 则返回一个长度为 source 的初始化数组;
- 如果 source 为字符串, 则按照指定的 encoding 将

字符串转换为字节序列;

- ❑ 如果 `source` 为可迭代类型, 则元素必须为`[0, 255]`中的整数;
- ❑ 如果 `source` 为与 `buffer` 接口一致的对象, 则此对象也可以被用于初始化 `bytearray`;
- ❑ 如果没有输入任何参数, 默认就是初始化数组为 0 个元素。

4.1.9 函数 `callable()`

在 Python 程序中, 函数 `callable(object)`的功能是检查一个对象是否是可调用的。如果返回 `True`, 参数 `object`

仍然可能调用失败；但如果返回 `False`，调用参数对象 `object` 绝对不会成功。使用函数 `callable(object)` 的语法格式如下所示：

```
callable(object)
```

如果参数 `object` 是函数、方法、`lambda` 表达式、类，以及实现了 `__call__` 方法的类实例，则函数 `callable(object)` 都返回 `True`。在此需要注意，类是可调用的（调用类将返回一个新的实例）。如果实例中的类有 `__call__()` 方法，则它是可调用，否则它是不可调用的。

4.1.10 函数 `chr()`

在 Python 程序中，函数 `chr(i)` 的功能是用一个范围在

0~255 的整数作参数，返回一个对应的字符。函数 `chr()` 的语法格式如下所示：

```
chr(i)
```

参数 “i” 可以是十进制或十六进制形式的数字，返回一个表示 Unicode 码点为整数 i 的字符的字符串。例如，`chr(97)` 返回字符串 'a'，而 `chr(8364)` 返回字符串 '€'，它是 `ord()` 的逆操作。参数 “i” 的有效范围为 0~1114111 (基址 16 中的 0x10FFFF)。如果参数 “i” 超出该范围，则会引发 `ValueError`。

4.1.11 函数 `classmethod()`

在 Python 程序中，函数 `classmethod()` 的功能是将函数

包装成类方法。其语法格式如下所示：

```
classmethod(function)
```

在 Python 程序中，经常使用 `@classmethod` 修饰符的用法。在声明一个类方法时，通常使用如下所示的用法：

```
class C:
    @classmethod
    def f(cls, arg1, arg2, ...): ...
```

修饰符 `@classmethod` 对应的函数不需要实例化，不需要 `self` 参数，但第一个参数需要是表示自身类的 `cls` 参数，可以用来调用类的属性、类的方法、实例化对象等。`@classmethod` 形式的用法是一个函数装饰器，用于查看函数定义中关于函数定义的详细说明。`@classmethod` 既可以在类上调用（如 `C.f()`），也可以在实例上调用（如 `C().f()`）。除了实例的类，

实例本身被忽略。如果一个类方法在子类上调用，那么子类对象被传递为隐式的第一个参数。类方法不同于 C++ 或 Java 中的静态方法，如果需要静态方法，请参考本章后面介绍的 `staticmethod()` 函数。

4.1.12 函数 `compile()`

在 Python 程序中，函数 `compile()` 的功能是将一个字符串编译为字节代码。使用函数 `compile()` 的语法格式如下所示：

```
compile(source, filename, mode, flags=0,
dont_inherit=False, optimize=-1)
```

❑ `source`：字符串或者 AST (Abstract Syntax Trees) 对

象，函数 `compile()` 能够将 `source` 编译成代码对象，或者 AST (Abstract Syntax Tree, 抽象语法树) 对象。代码对象可以由 `exec()` 或 `eval()` 执行。源可以是普通字符串、字节字符串或 AST 对象。

- ❑ `filename`: 代码文件名称，如果不是从文件读取代码，则传递一些可辨认的值。
- ❑ `mode`: 指定编译代码的模式，可以指定为 `exec`、`eval`、`single`。如果 `source` 由语句序列组成，则它可以是 `'exec'`；如果它是单个语句，则可以使用 `'eval'`；如果它由单个交互式语句组成，则可以使

用'single'。

- ❑ flags: 变量作用域, 局部命名空间, 如果被提供, 可以是任何映射对象。
- ❑ flags 和 dont_inherit: 是用来控制编译源码时的标志。
- ❑ optimize: 指定编译器的优化级别; 默认值为-1, 选择由-O 选项给出的解释器的优化级别。显式级别为 0 (无优化; __debug__为真), 1 (声明被删除, __debug__为假) 或 2 (docstrings 也被删除)。

在上述参数中, 可选参数 flags 和 dont_inherit 控制哪

些未来版本的语句会应用于源编译。如果两者都不存在 (或两者都为 0), 则使用在调用 `compile()` 的代码中生效的未来语句来编译代码。如果给出了 `flags` 参数且没有给出 `dont_inherit` 参数 (或者为 0), 除了本该使用的 `future` 语句之外, 由 `flags` 参数指明的 `future` 语句也会影响编译。如果 `dont_inherit` 是非 0 整数, `flags` 参数被忽略 (调用 `compile` 周围的有效的 `future` 语句被忽略)。

4.1.13 函数 `complex()`

在 Python 程序中, 函数 `complex()` 的功能是创建一个值为 “`real + imag * j`” 的复数或者转化一个字符串或数为复

数。如果第一个参数为字符串，则不需要指定第二个参数。

使用函数 `complex()` 的语法格式如下所示：

```
class complex([real[, imag]])
```

❑ `real`: `int`、`long`、`float` 或字符串。

❑ `imag`: `int`、`long` 或 `float` 类型。

函数 `complex()` 的返回值形式为 “`real + imag * j`” 的复数，或将字符串或数字转换为复数。如果第一个参数是一个字符串，它将被解释成复数，同时函数不能有第二个参数。第二个参数不能是字符串。每个参数必须是数值类型（包括复数）。如果省略参数 `imag`，则默认为 0，构造函数会像 `int` 和 `float` 一样进行转换。如果省略这两个参数，则

返回 0j。

4.1.14 函数 delattr()

在 Python 程序中，函数 delattr()的功能是删除指定对象的某个属性。使用函数 delattr()的语法格式如下所示：

```
delattr(object, name)
```

- ❑ object：对象。
- ❑ name：是一个字符串，这个字符串必须是对象的某个属性的名字。

例如，代码 “delattr(x, 'foobar’)” 等同于 “del x.foobar”，表示删除对象 x 中的属性 foobar。

4.1.15 函数 dict()

在 Python 程序中，函数 dict()的功能是创建一个字典。

在现实中有如下 3 种使用函数 dict()的语法格式：

```
class dict(**kwarg)
class dict(mapping, **kwarg)
class dict(iterable, **kwarg)
```

- ❑ ****kwargs**：关键字。
- ❑ **Mapping**：元素的容器。
- ❑ **Iterable**：可迭代对象。

4.1.16 函数 dir()

在 Python 程序中，如果函数 `dir()` 没有参数，则返回当前本地作用域内的名字列表。如果有参数，则尝试返回参数所指明对象的合法属性的列表。使用函数 `dir()` 的语法格式如下所示：

```
dir([object])
```

参数 `object` 是对象、变量或类型。如果参数 `object` 具有名为 `__dir__()` 的方法，那么将调用此方法，并且必须返回属性列表。这允许实现自定义 `__getattr__()` 或 `__getattribute__()` 函数的对象自定义 `dir()` 报告其属性的方式。如果参数 `object` 不提供 `__dir__()`，则函数会尽量从对象的 `__dict__` 属性（如

果已定义) 和其类型对象中收集信息。结果列表不一定是完整的, 并且当对象具有自定义`__getattr__()`时可能会不准确。具体来说, 在使用函数 `dir()` 时, 根据参数的不同会返回不同的结果, 具体说明如下所示:

- ❑ 如果参数为空, 则返回当前作用域内的变量、方法和定义的类型列表。
- ❑ 当参数是一个模块时, 则会返回模块的属性、方法列表。
- ❑ 当参数是一个类时, 则会返回类及其子类的属性、方法列表。

- 当在参数对象中定义了__dir__方法时，则返回__dir__方法的结果。

4.1.17 函数 divmod()

在 Python 程序中，函数 divmod()的功能是把除数和余数运算结果结合起来，返回一个包含商和余数的元组 (a // b, a % b)。在 Python 2.3 版本之前，函数 divmod()不允许处理复数。使用函数 divmod()的语法格式如下所示：

```
divmod(a, b)
```

参数 a 和参数 b 都是数字，函数 divmod()能够取两个(非复数)数字作为参数，并在使用整数除法时返回由商和余数组成的一对数字。对于混合的操作数类型参数，采用

二元算术运算符的规则。对于整数参数来说，结果与 $(a // b, a \% b)$ 相同。对于浮点数参数来说，结果为 $(q, a \% b)$ ，其中 q 通常为 $\text{math.floor}(a / b)$ ，也有可能比这个结果小 1。不管怎样， $q * b + a \% b$ 非常接近于 a ，如果 $a \% b$ 非 0，它和 b 符号相同且 $0 \leq \text{abs}(a \% b) < \text{abs}(b)$ 。

4.1.18 函数 `enumerate()`

在 Python 程序中，函数 `enumerate()` 的功能是将一个可遍历的数据对象（如列表、元组或字符串）组合为一个索引序列，同时列出数据和数据下标，一般被用在 `for` 循环中。使用函数 `enumerate()` 的语法格式如下所示：

```
enumerate(sequence, [start=0])
```

- ❑ `sequence`: 必须是一个序列、一个迭代器, 或者其他某种支持迭代的对象。
- ❑ `start`: 下标起始位置。

4.1.19 函数 `eval()`

在 Python 程序中, 函数 `eval()` 的功能是执行一个字符串表达式, 并返回表达式的值。使用函数 `eval()` 的语法格式如下所示:

```
eval(expression, globals=None, locals=None)
```

- ❑ `expression`: 表达式。
- ❑ `globals`: 变量作用域, 表示全局命名空间。如果被提供, 则必须是一个字典对象。

- ❑ `locals`: 变量作用域, 表示局部命名空间。如果有局部变量, 则 `locals` 可以是任何映射类型对象。

4.1.20 函数 `exec()`

在 Python 程序中, 函数 `exec()` 的功能是执行储存在字符串或文件中的 Python 语句, 和函数 `eval()` 相比, 函数 `exec()` 可以执行更复杂的 Python 代码。使用函数 `exec()` 的语法规则如下所示:

```
exec(object[, globals[, locals]])
```

- ❑ `object`: 必选参数, 表示需要被指定的 Python 代码, 必须是字符串或 `code` 对象。如果 `object` 是一个字符串, 则该字符串会先被解析为一组 Python 语句,

然后在执行（除非发生语法错误）。如果 object 是一个 code 对象，那么它只是被简单地执行。

- ❑ `globals`：可选参数，表示全局命名空间（存放全局变量），如果被提供，则必须是一个字典对象。
- ❑ `locals`：可选参数，表示当前局部命名空间（存放局部变量）。如果被提供，可以是任何映射对象。如果该参数被忽略，那么它将会取与 `globals` 相同的值。

4.1.21 函数 `filter()`

在 Python 程序中，函数 `filter()` 的功能是过滤序列，过

滤掉不符合条件的元素，返回一个 filter 类。filter 类实现了 `__iter__` 和 `__next__` 方法，可以看成是一个迭代器，有惰性运算的特性。使用函数 `filter()` 的语法格式如下所示：

```
filter(function, iterable)
```

- ❑ function：判断函数。
- ❑ iterable：可迭代对象。

4.1.22 函数 float()

在 Python 程序中，函数 `float()` 的功能是将整数和字符串转换成浮点数。使用函数 `float()` 的语法格式如下所示：

```
class float([x])
```

参数 “x” 是一个整数或字符串，如果参数 “x” 是一

个字符串，它应该包含一个十进制数，可选地前面有一个符号，并且可选地嵌入在空格中。可选的 `sign` 可以是 '+' 或 '-'； '+' 符号对生成的值没有影响。参数 “x” 还可以是表示 NaN（非数字）或正或负无穷大的字符串。更确切地说，输入必须符合如下所示的语法，前导和尾随空白字符被删除：

```
sign          ::= "+" | "-"
infinity      ::= "Infinity" | "inf"
nan           ::= "nan"
numeric_value ::= floatnumber | infinity | nan
numeric_string ::= [sign] numeric_value
```

在上述格式中，`floatnumber` 是在浮点字面值中描述的 Python 浮点字面值的形式。例如 “inf” “Inf” “INFINITY”

和 “iNfINity” 都是正无穷大的可接受拼写。如果参数 “x” 是整数或浮点数，则返回具有相同值（在 Python 的浮点精度内）的浮点数。如果参数 “x” 在 Python 浮点数的范围之外，则引发一个 OverflowError 错误。对于一般的 Python 对象 x，float(x)委托给 x .__float__()。如果没有给出参数 “x”，则返回 0.0。

4.1.23 函数 format()

在 Python 程序中，函数 format()是一种格式化字符串的函数，它增强了字符串格式化的功能。使用函数 format() 的语法格式如下所示：

```
format(value[, format_spec])
```

函数 `format()` 的功能是将 `value` 转化成“格式化”的表现形式，格式由 `format_spec` 控制。对 `format_spec` 的解释依赖于参数 `value` 的类型，大多数内置类型有标准的格式化语法。`format_spec` 是一个格式化参数，默认是一个空字符串，通常给出与调用 `str (value)` 相同的效果。对参数 `format_spec` 的调用将被转换为 `type (value) .__ format __ (value, t4> format_spec)` 其在搜索值的 `__format__()` 方法时绕过实例字典。如果方法搜索到达 `object` 并且 `format_spec` 不为空，或者如果 `format_spec`，则会引发 `TypeError t7>` 或返回 `返回值不是字符串`。

4.1.24 函数 frozenset()

在 Python 程序中，函数 `frozenset()` 的功能是返回一个冻结的集合，冻结后的集合不能再添加或删除任何元素。使用函数 `frozenset()` 的语法格式如下所示：

```
class frozenset([iterable])
```

函数 `frozenset()` 能够返回一个新的 `frozenset` 对象，参数 `iterable` 是一个可迭代的对象，例如列表、字典、元组等。如果可选参数 `iterable` 存在，则 `frozenset` 的元素来自于 `iterable`。

4.1.25 函数 getattr()

在 Python 程序中，函数 `getattr()` 的功能是返回一个对

象属性值。使用函数 `getattr()` 的语法格式如下所示：

```
getattr(object, name[, default])
```

- ❑ `object`：是一个对象，函数 `getattr()` 的功能是从对象 `object` 中获取名为 `name` 的属性，等效与调用 `object.name`。
- ❑ `name`：对象属性，`name` 必须是一个字符串。如果字符串是对象某个属性的名字，则返回该属性的值。例如，`getattr(x, 'foobar')` 等同于 `x.foobar`。如果这个名字的属性不存在，提供 `default` 则返回它，否则引发 `AttributeError`。
- ❑ `default`：默认返回值，如果不提供该参数，在没有

对应属性时，将触发 `AttributeError`。

4.1.26 函数 `globals()`

在 Python 程序中，函数 `globals()` 的功能是以字典类型返回当前位置的全部全局变量，也就是返回表示当前全局符号表的字典。函数 `globals()` 总是当前模块的字典，在函数或者方法中，它是指定义的模块，而不是调用的模块。

4.1.27 函数 `hasattr()`

在 Python 程序中，函数 `hasattr()` 的功能是判断在指定对象中是否包含对应的属性。使用函数 `hasattr(object, name)` 的语法格式如下所示：

```
hasattr(object, name)
```

- ❑ object: 对象。
- ❑ name: 字符串, 表示属性名。

4.1.28 函数 hash()

在 Python 程序中, 函数 hash()的功能是获取取一个对象 (字符串或者数值等) 的哈希值。使用函数 hash()的语法格式如下所示:

```
hash(object)
```

参数 object 是一个对象, 返回的哈希值是一个整数。哈希值用于在查找字典时快速地比较字典的键, 相等数值的哈希值相同 (即使它们的类型不同, 比如 1 和 1.0)。

4.1.29 函数 help()

在 Python 程序中，使用函数 help() 的语法格式如下所示：

```
help([object])
```

函数 help() 的功能是调用内置的帮助系统，查看 object 函数或模块用途的详细说明。如果没有参数 object，在解释器的控制台启动交互式帮助系统。如果参数 object 是一个字符串，该字符串被当作模块名、函数名、类名、方法名、关键字或者文档主题而被查询，在控制台上打印帮助页面。如果参数 object 是其他类型的某种对象，则会生成关于对象的帮助页面。

4.1.30 函数 hex()

在 Python 程序中，使用函数 hex() 的语法格式如下所示：

```
hex(x)
```

函数 hex() 的功能是将十进制整数转换成十六进制，以字符串形式表示。参数 x 是一个十进制整数。如果参数 x 不是 Python int 对象，则必须定义一个 __index__() 方法，返回一个整数。读者需要注意的是，如果要获取浮点型的十六进制字符串表示形式，需要使用 float.hex() 方法实现。

4.1.31 函数 id()

在 Python 程序中，使用函数 id() 的语法格式如下所示：

```
id(object)
```

函数 `id()` 的功能是获取对象 `object` 的内存地址，这个内存地址是一个整数，能够保证在该对象的生命周期内是唯一的和恒定的。在 Python 程序中，具有不重叠寿命的两个对象可以具有相同的 `id()` 值。

4.1.32 函数 `input()`

在 Python 程序中，使用函数 `input()` 的语法格式如下所示：

```
input([prompt])
```

函数 `input()` 的功能是获取用户在控制台中输入的信息。如果有 `prompt` 参数，则将它输出到标准输出且不带换行。然后函数 `input()` 从标准输入读取一行，将它转换成一

个字符串（去掉一个末尾的换行符），然后返回它。当读取到 EOF 时，会产生 EOFError 错误。

4.1.33 函数 int()

在 Python 程序中，使用函数 int() 的语法格式如下所示：

```
class int(x=0)
class int(x, base=10)
```

函数 int() 的功能是将一个字符串或数字转换为整型。参数 x 是一个字符串或数字，参数 base 表示进制数，默认为十进制。函数 int() 能够从数字或字符串 (x) 构造并返回一个整数对象，如果没有给出参数 x，则返回 0。如果 x 是一个数字，则返回 x.__int__()。如果 x 是一个浮点数，将截断到零只取整。如果参数 x 不是数字，或者如果给定参数 base，则 x 必须是字符串 bytes bytearray 实例代表基数 base 中的

integer literal。字面量的前面可以有+或者-（中间不能有空格），周围可以有空白。以 n 为基数的字面量包含数字 0~n-1，用 a~z（或者 A~Z）来表示 10~35。Base 的默认值是 10。如果 base 为 0，则意味着完全解释为代码字面值，使得实际基数为 2,8,10 或 16，并且使得 int('010', 0) 是不合法的，而 int('010')是以及 int('010', 8)。

4.1.34 函数 isinstance()

在 Python 程序中，使用函数 isinstance()的语法格式如下所示：

```
isinstance(object, classinfo)
```

□ object：实例对象。

- ❑ `classinfo`: 可以是直接或间接类名、基本类型或者有它们组成的元组。

函数 `isinstance()` 的功能是判断一个对象是否是一个已知的类型, 其功能类似于函数 `type()`。如果 `object` 对象的类型与参数 `classinfo` 的类型相同, 则返回 `True`, 否则返回 `False`。

4.1.35 函数 `issubclass()`

在 Python 程序中, 使用函数 `issubclass()` 的语法格式如下所示:

```
issubclass(class, classinfo)
```

- ❑ `class`: 类, 表示需要检查的类型对象。

❑ `classinfo`: 类, 表示需要对比类型对象。

函数 `issubclass()` 的功能是判断参数 `class` 是否是类型参数 `classinfo` 的子类。如果 `class` 参数是参数 `classinfo` 的类型对象 (或者 `classinfo` 类对象的直接、间接、虚拟子类) 的实例, 则返回 `True`。

4.1.36 函数 `iter()`

在 Python 程序中, 使用函数 `iter()` 的语法格式如下所示:

```
iter(object[, sentinel])
```

❑ `object`: 支持迭代的集合对象。

❑ `sentinel`: 如果传递了第二个参数, 则参数 `object` 必须是一个可调用的对象 (如函数), 此时, `iter`

创建了一个迭代器对象，每次调用这个迭代器对象的`__next__()`方法时，都会调用 `object`。

4.1.37 函数 `len()`

在 Python 程序中,使用函数 `len()`的语法格式如下所示:

```
len( s )
```

函数 `len()`的功能是返回参数 `s` 的长度或项目个数，参数 `s` 表示对象，可以是序列（如字符串、字节、元组、列表或者范围）或者集合（如字典、集合或者固定集合）等。

4.1.38 函数 `list()`

在 Python 程序中,使用函数 `list()`的语法格式如下所示:

```
list( seq )
```

函数 `list()` 实际是上列表类型的构造函数的功能是将元组 `seq` 转换为列表。元组与列表是非常类似的，区别在于元组的元素值不能修改，元组是放在括号中，列表是放于方括号中。

4.1.39 函数 locals()

在 Python 程序中，使用函数 `locals()` 的语法格式如下所示：

```
locals()
```

函数 `locals()` 的功能是以字典类型返回当前位置的全部局部变量。对于函数、方法、`lambda`、类，以及实现了 `__call__` 方法的类实例，都会返回 `True`。当 `locals()` 在函数代码块中调用时会返回自由变量，但是在类代码块中不会。

4.1.40 函数 map()

在 Python 程序中，使用函数 `map()` 的语法格式如下所示：

```
map(function, iterable, ...)
```

❑ `function`: 函数, 有两个参数。

❑ `iterable`: 一个或多个序列。

函数 `map()` 的功能是根据提供的函数对指定序列做映射。第一个参数 `function` 以参数序列中的每一个元素调用 `function` 函数, 返回包含每次 `function` 函数返回值的新列表。

函数 `map()` 最终会返回一个迭代器, 对 `iterable` 的每个项应用 `function`, 并 `yield` 结果。如果传递多个 `iterable` 参数, `function` 必须接受这么多参数, 并应用到从 `iterables` 并行提取的项中。如果有多个 `iterable`, 迭代器则在最短的 `iterable` 耗尽时停止。

4.1.41 函数 max()

在 Python 程序中，使用函数 max()的语法格式如下

所示：

```
max(iterable, *[, key, default])  
max(arg1, arg2, *args[, key])
```

参数 arg1、arg2 和 args 可以都是数值表达式，函数 max() 的功能是返回给定参数的最大值。因为参数可以为序列，所以函数 max()也能返回传入的多个参数中的最大值，或者传入的可迭代对象元素中的最大值。默认数值型参数取值大者，字符型参数取字母表排序靠后者。还可以传入命名参数 key，其为一个函数，用来指定取最大值的方法。命名参

数 default 用来指定最大值不存在时返回的默认值。函数 max() 至少传入两个参数，但是只有传入一个参数的例外，此时参数必须为可迭代对象，返回的是可迭代对象中的最大元素。

4.1.42 函数 memoryview()

在 Python 程序中，使用函数 memoryview() 的语法格式如下所示：

```
memoryview(obj)
```

参数 “obj” 是一个对象，函数 memoryview() 的功能是返回给定参数的内存查看对象 (Memory view)。所谓内存查看对象，是指对支持缓冲区协议的数据进行包装，在不需要复制对象基础上允许 Python 代码访问。在 Python 内置

对象中，支持缓冲区协议的对象有 `bytes` 和 `bytearray`。

4.1.43 函数 `min()`

在 Python 程序中，使用函数 `min()` 的语法格式如下所示：

```
min(iterable, *[, key, default])  
min(arg1, arg2, *args[, key])
```

函数 `min()` 的功能是返回可迭代的对象中的最小的元素，或者返回两个或多个参数中最小的参数。函数 `min()` 的参数默认为数值型，取值小者；如果是字符型参数，则取字母表排序靠前者。还可以传入命名参数 `key`，其为一个函数，用来指定取最小值的方法。命名参数 `default` 用来指定最小值不存在时返回的默认值。其功能与 `max` 函数相反。

4.1.44 函数 next()

在 Python 程序中，使用函数 next() 的语法格式如下所示：

```
next(iterator[, default])
```

- ❑ iterator：可迭代对象。
- ❑ default：可选参数，用于设置在没有下一个元素时返回该默认值。如果不设置这个参数，又没有下一个元素，则会触发 StopIteration 异常。

函数 next() 的功能是返回迭代器的下一个项目，通过调用 __next__() 方法从迭代器中检索下一个项目。如果有参数 default，在迭代器迭代完所有元素之后返回该参数；否则会抛出 StopIteration。

4.1.45 函数 oct(x)

在 Python 程序中，使用函数 oct(x)的语法格式如下所示：

```
oct(x)
```

参数 x 是一个整数，函数 oct(x)的功能是将一个整数转换成八进制字符串。如果 x 不是 Python int 对象，则必须定义一个返回整数的__index__()方法。

4.1.46 函数 open()

在读取一个文件的内容之前，需要先打开这个文件。

在 Python 程序中，可以通过内置函数 open()来打开一个文件，并用相关的方法读或写文件中的内容供程序处理和使

用，而且也可以将文件看作是 Python 中的一种数据类型。

使用函数 `open()` 的语法格式如下所示：

```
open(file, mode='r', buffering=-1, encoding=None,
      errors=None, newline=None, closefd=True, opener=None)
```

- ❑ `file`：表示要打开的文件名。
- ❑ `mode`：可选参数，文件打开模式。这个参数是非强制的，默认文件访问模式为只读(`r`)。不同模式打开文件的完全列表如表 4-1 所示。

表 4-1 不同模式打开文件的完全列表

模式	描述
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式

4.1 标准内置函数

r+	打开一个文件用于读写。文件指针将会放在文件的开头
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头
w	打开一个文件只用于写入。如果该文件已存，在则将其覆盖。如果该文件不存在，创建新文件
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在，则将其覆盖。如果该文件不存在，则创建新文件
w+	打开一个文件用于读写。如果该文件已存在，则将其覆盖。如果该文件不存在，则创建新文件
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在，则将其覆盖。如果该文件不存在，则创建新文件

续表

模式	描述
a	打开一个文件用于追加。如果该文件已存在，则文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，则创建新文件进行写入
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，则文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不

	存在，则创建新文件进行写入
a+	打开一个文件用于读写。如果该文件已存在，则文件指针将会放在文件的结尾。文件打开时会追加模式。如果该文件不存在，则创建新文件用于读写
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，则文件指针将会放在文件的结尾。如果该文件不存在，则创建新文件用于读写

4.1.47 函数 ord()

在 Python 程序中，使用函数 ord() 的语法格式如下所示：

```
ord(c)
```

参数 *c* 是一个字符，函数 ord() 的功能和 chr() 函数刚好相反，功能是返回参数 *c* 对应的整数数值。函数 ord() 以一个字符（长度为 1 的字符串）作为参数，返回对应的 ASCII 数值，或者 Unicode 数值。例如，ord('a') 返回整数 97 和 ord('€')

(欧元符号) 返回 8364。如果所给的 Unicode 字符超出了你的 Python 定义范围，则会引发一个 `TypeError` 异常。

4.1.48 函数 `pow()`

在 Python 程序中，使用函数 `pow()` 的语法格式如下所示：

```
pow(x, y[, z])
```

函数 `pow()` 的功能是计算 x 的 y 次方值，如果参数 z 存在，则再对结果进行取模运算，其结果等效于 `pow(x,y) %z`。函数 `pow()` 的参数必须是数字类型，由于操作数是混合类型的，二进制计算的原因需要一些强制的规定。对于 `int` 操作数，结果具有与操作数相同的类型（强制后），除非第二个参数为负；在这种情况下，所有参数都转

换为 float，并传递 float 结果。例如， $10^{**}2$ 返回 100，但 $10^{**}-2$ 返回 0.01。如果第二个参数为负数，那么第三个参数必须省略。

4.1.49 函数 print()

在 Python 程序中，使用函数 print() 的语法格式如下所示：

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

- ❑ objects：复数，表示可以一次输出多个对象。当输出多个对象时，需要用逗号分隔。
- ❑ sep：用来间隔多个对象，默认值是一个空格。
- ❑ end：用来设定以什么结尾，默认值是换行符 \n，

可以将其换成其他字符串。

- ❑ file: 要写入的文件对象。
- ❑ flush: 设置输出是否使用缓存, 默认 False。如果设置为 true, 那么输出流将会被强制刷新。

4.1.50 函数 property()

在 Python 程序中, 使用函数 property() 的语法格式如下

所示:

```
class property([fget[, fset[, fdel[, doc]]]])
```

- ❑ fget: 获取属性值的函数。
- ❑ fset: 设置属性值的函数。
- ❑ fdel: 删除属性值函数。

❑ doc: 属性描述信息。

函数 `property()` 的功能是返回一个 `property` 属性。`property` 是一个类，其作用是用来包装类的属性，这个属性可以根据实际需要，控制是否可读（设置 `fget` 参数）、可写（设置 `fset` 参数）、可删除（设置 `fdel` 参数）。

4.1.51 函数 `range()`

在 Python 程序中，使用函数 `range()` 的语法格式如下所示：

```
range(stop)
range(start, stop[, step])
```

❑ `start`: 计数从 `start` 开始。默认是从 0 开始。例如

`range (5)` 等价于 `range (0, 5)`。

□ `end`: 计数到 `end` 结束, 但不包括 `end`。例如 `range`

`(0, 5)` 是 `[0, 1, 2, 3, 4]` 没有 5。

□ `step`: 步长, 默认为 1。例如: `range (0, 5)` 等价于

`range(0, 5, 1)`。

函数 `range()` 的功能是创建一个整数列表, 一般被用在 `for` 循环中。

4.1.52 函数 `repr()`

在 Python 程序中, 使用函数 `repr()` 的语法格式如下所示:

```
repr(object)
```

函数 `repr()` 的功能是返回参数对象 `object` 可打印形式的

字符串，其功能和 `str()` 函数比较类似，但是两者也有差异：

函数 `str()` 用于将值转化为适于人们阅读的形式，而函数 `repr()` 转化为供解释器读取的形式。

4.1.53 函数 `reversed()`

在 Python 程序中，使用函数 `reversed()` 的语法格式如下所示：

```
reversed(seq)
```

参数 “seq” 表示要转换的序列，可以是 `tuple`、`string`、`list` 或 `range` 类型。函数 `reversed()` 的功能是返回一个反转的 `iterator` 迭代器。seq 必须是一个具有 `__reversed__()` 方法或支持序列协议的对象（整数参数从 0 开始的 `__len__()` 方法

和`__getitem__()`方法)。

4.1.54 函数 `round()`

在 Python 程序中，使用函数 `round()` 的语法格式如下所示：

```
round(number[, ndigits])
```

函数 `round()` 的功能是返回浮点数 `x` 的四舍五入值，也就是返回参数 `number` 舍入到小数点后 `ndigits` 位的浮点值。如果省略参数 `ndigits`，将返回最接近输入的整数。底层调用的是 `number.__round__(ndigits)`。对于支持 `round()` 的内建类型来说，值舍入到 10 的最接近的负 `ndigits` 次幂的倍数；如果离两个倍数的距离相等，则舍入选择偶数（因

此, `round(0.5)`和 `round(-0.5)`都是 0, 而 `round(1.5)`是 2)。

如果使用一个参数调用, 返回值是一个整数, 否则类型与 `number` 相同。

4.1.55 函数 set()

在 Python 程序中，使用函数 set() 的语法格式如下所示。

```
class set([iterable])
```

函数 set() 的功能是创建一个无序不重复元素集，可以进行关系测试，删除重复数据，还可以计算交集、差集和并集等。执行函数 set() 后会返回一个新的 set 对象，其元素可以从可选的参数 iterable 中获得。

4.1.56 函数 setattr()

在 Python 程序中，使用函数 setattr() 的语法格式如下所示：

```
setattr(object, name, value)
```

- ❑ object: 对象。
- ❑ name: 字符串, 表示对象属性。字符串既可以是一个已存在属性的名字, 也可以是一个新属性的名字。
- ❑ value: 属性值。

4.1.57 函数 slice()

在 Python 程序中, 使用函数 slice() 的语法格式如下

所示:

```
class slice(stop)
class slice(start, stop[, step])
```

- ❑ start: 起始位置。

❑ stop: 结束位置。

❑ step: 间距。

函数 `slice()` 的功能是实现切片对象，主要用在切片操作函数里的参数传递操作。函数 `slice()` 能够返回一个 `slice` 对象，表示由索引 `range(start, stop, step)` 指出的集合。参数 `start` 和参数 `step` 默认为 `None`。切片对象具有只读属性 `start`、`stop` 和 `step`，它们仅仅返回参数的值（或者它们的默认值）。他们没有其他明确的功能；但是它们被数字 Python 和其他第三方扩展使用。在使用扩展的索引语法时同样会生成切片对象。例如：`a[start:stop:step]` 或者 `a[start:stop, i]`。

4.1.58 函数 sorted()

在 Python 程序中，使用函数 sorted() 的语法格式如下

所示：

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

- ❑ **iterable**：可迭代对象。
- ❑ **key**：一个带有一个参数的函数，用于从列表的每个元

素中提取比较的关键字：key=str.lower。默认值是 None

(直接比较元素)。该参数用于指定可迭代对象中的一个

元素来进行排序；

- ❑ **reverse**：排序规则，是一个布尔值。如果设置为 True，那么列表中元素反过来比较来排序。reverse = True

时表示降序，`reverse = False`（默认）时表示升序。

函数 `sorted()` 的功能是对所有可迭代的对象进行排序操作，与函数 `sort()` 的区别如下所示：

- ❑ 函数 `sort()` 是应用在 `list` 上的方法，函数 `sorted()` 可以对所有可迭代的对象进行排序操作。
- ❑ `list` 的 `sort()` 函数返回的是对已经存在的列表进行操作，而内建函数 `sorted()` 返回的是一个新的 `list`，而不是在原来的基础上进行的操作。

4.1.59 函数 `staticmethod()`

在 Python 程序中，使用函数 `staticmethod()` 的语法格式

如下所示:

```
staticmethod(function)
```

函数 `staticmethod()` 的功能是返回 `function` 的一个静态方法, 静态方法不接受隐式的第一个参数 (也就是实例名称 `self`)。要想声明静态方法, 请使用下面的习惯方式:

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

`@staticmethod` 形式是一个函数装饰器, 可以在类 (如 `C.f()`) 或实例 (如 `C().f()`) 中调用。除了它的类型, 实例其他的内容都被忽略。

4.1.60 函数 str()

在 Python 程序中，使用函数 str() 的语法格式如下所示：

```
class str(object='')
class str(object=b'', encoding='utf-8', errors=
'strict')
```

函数 str() 的功能是返回 object 的 str 版本，参数 str 是一个内置字符串类。

4.1.61 函数 sum()

在 Python 程序中，使用函数 sum() 的语法格式如下所示：

```
sum(iterable[, start])
```

- ❑ iterable：可迭代对象，例如列表。
- ❑ start：指定相加的参数，如果没有设置这个值则默认为 0。

函数 `rsum()` 的功能是将 `start` 以及 `iterable` 的元素从左向右相加并, `iterable` 的元素通常是数字, `start` 值不允许是一个字符串。对于某些使用情况, 有很好的替代 `sum()` 的方法。连接字符串序列的首选快速方法是调用 `".join(sequence)`。要以扩展精度添加浮点值, 请使用 `math.fsum()`。要连接一系列可迭代对象, 请使用 `itertools.chain()`。

4.1.62 函数 `super()`

在 Python 程序中, 使用函数 `super()` 的语法格式如下所示:

```
super([type[, object-or-type]])
```

□ `type`: 类。

❑ object-or-type: 类，一般是 self。

函数 `super()` 是调用父类（超类）的一个方法，能够返回一个代理对象，它委托方法给父类或者 `type` 的同级类，这对于访问类中被覆盖的继承方法很有用。除了跳过 `type` 本身之外，搜索顺序与 `getattr()` 所使用的顺序相同。

4.1.63 函数 `type()`

在 Python 程序中，使用函数 `type()` 的语法格式如下所示：

```
class type(object)
class type(name, bases, dict)
```

❑ name: 类的名称。

- ❑ `bases`: 基类的元组。
- ❑ `dict`: 字典, 类内定义的命名空间变量。

当函数 `type()` 只有一个参数 `object` 时, 返回 `object` 的类型。返回值是一个类型对象, 通常与 `object.__class__` 返回的对象相同。

4.1.64 函数 `vars()`

在 Python 程序中, 使用函数 `vars()` 的语法格式如下所示:

```
vars([object])
```

函数 `vars()` 的功能是返回参数对象 `object` 的属性和属性值的字典对象。

4.1.65 函数 zip()

在 Python 程序中，使用函数 zip() 的语法格式如下所示：

```
zip([iterable, ...])
```

参数 `iterabl` 表示一个或多个迭代器，函数 `zip()` 的功能是将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用 `*` 号操作符，可以将元组解压为列表。

4.1.66 函数__import__()

在 Python 程序中，使用函数__import__()的语法格式如下所示：

```
__import__(name[, globals[, locals[, fromlist[,  
level]]]])
```

参数 name 表示模块名，函数__import__()的功能是动态加载指定的类和函数。如果一个模块经常变化，就可以使用函数__import__()实现动态载入功能。函数__import__(module) 相当于 import module 语句，例如在下面的实例中演示了函数__import__()的这一用法。

4.2 文 本 处 理

在 Python 语言中，文本数据由 `str` 对象或 `strings` 进行处理。字符串是不可变的 Unicode 码点序列。

4.2.1 方法 `str.capitalize()`

在 Python 语言中，方法 `str.capitalize()` 的功能是返回字符串的副本，该副本第一个字符大写，其余字符小写。

4.2.2 方法 `str.casefold()`

在 Python 语言中，`str.casefold()` 的功能是返回字符串的小写形式，其功能和小写方法 `lower()` 相同，但 `casefold()` 的功能更强大，因为它旨在删除字符串中的所有 case 区别。

例如，德国小写字母'ß'等效于"ss"。由于它已经是小写的，所以 `lower()` 对'ß'不起作用，而函数 `casefold()` 能够将其转换为"ss"。

4.2.3 方法 `str.count()`

在 Python 语言中，方法 `str.count(sub[, start[, end]])` 的功能是返回在 `[start, end]` 范围内的子串 `sub` 非重叠出现的次数。可选参数 `start` 和 `end` 都以切片表示法解释，分别表示字符串的开始和结束限定范围。

- ❑ `sub`：搜索的子字符串。
- ❑ `start`：字符串开始搜索的位置。默认为第一个字符，

第一个字符索引值为 0。

- ❑ `end`: 字符串中结束搜索的位置。字符串中第一个字符的索引为 0。默认为字符串的最后一个位置。

4.2.4 方法 `str.encode()`

在 Python 语言中, 方法 `str.encode(encoding="utf-8", errors="strict")` 的功能是将字符串的编码版本作为字节对象返回, 默认编码为 'utf-8'。errors 的默认值是 'strict', 意思编码错误引发一个 `UnicodeError`。

- ❑ `encoding`: 要使用的编码, 如 "UTF-8"。
- ❑ `errors`: 设置不同错误的处理方案, 默认为 'strict',

意为编码错误引起一个 `UnicodeError`。其他可能得值

有 `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'`

以及通过 `codecs.register_error()` 注册的任何值。

4.2.5 方法 `str.endswith()`

在 Python 语言中，方法 `str.endswith(suffix[, start[, end]])` 的功能是如果字符串以指定的 `suffix` 结尾，则返回 `True`，否则返回 `False`。`suffix` 也可以是一个元组。可选的 `start` 表示从该位置开始测试，可选的 `end` 表示在该位置停止比较。

□ `suffix`: 该参数可以是一个字符串或者是一个元素。

- ❑ start: 字符串中的开始位置。
- ❑ end: 字符串中的结束位置。

4.2.6 方法 str.expandtabs()

在 Python 语言中, 方法 `str.expandtabs(tabsize=8)` 的功能是把字符串中的 tab 符号(`\t`)转为空格, tab 符号(`\t`)默认的空格数是 8。参数 “tabsize” 用于指定转换字符串中的 tab 符号(`\t`)转为空格的字符数。例如下面的演示过程:

```
>>> '01\t012\t0123\t01234'.expandtabs()
'01          012          0123          01234 '
>>> '01\t012\t0123\t01234'.expandtabs(4)
'01  012 0123      01234 '
```

4.2.7 方法 str.find()

在 Python 语言中, 方法 `str.find(sub[, start[, end]])` 的功能

是 `str.find (sub[, start[, end]])` 检测字符串中是否包含子字符串 `str`，如果指定 `beg`（开始）和 `end`（结束）范围，则检查是否包含在指定范围内，如果包含子字符串返回开始的索引值，否则返回-1。

- ❑ `str`：指定检索的字符串。
- ❑ `beg`：开始索引，默认为 0；
- ❑ `end`：结束索引，默认为字符串的长度。

例如下面的演示过程：

```
True  
str.format(*args, **kwargs)
```


4.2.8 方法 str.format()

在 Python 语言中,方法 `str.format(*args, **kwargs)` 的功能是执行字符串格式化操作,调用此方法的字符串可以包含文本字面值或由花括号`{}`分隔的替换字段,每个替换字段包含位置参数的数字索引或关键字参数的名称。返回字符串的一个副本,其中每个替换字段使用对应参数的字符串值替换。例如下面的演示过程:

```
>>> "The sum of 1 + 2 is {}".format(1+2)
'The sum of 1 + 2 is 3'
```

4.2.9 方法 str.format_map()

在 Python 语言中,方法 `str.format_map(mapping)` 的功

能类似于 `str.format(**mapping)`，区别在于 `format_map` 直接用字典，而不是复制一个。

4.2.10 方法 `str.index()`

在 Python 语言中，方法 `str.index(str, beg=0, end=len(string))` 的功能是检测字符串中是否包含子字符串 `str`，如果指定 `beg`（开始）和 `end`（结束）范围，则检查是否包含在指定范围内，该方法与 Python `find()` 方法一样，只不过如果 `str` 不在 `string` 中会报一个异常。方法 `str.index()` 的功能与 `str.find()` 方法类似，区别在于如果 `index` 找不到要寻到的字符，则会得到 `ValueError`，而 `find` 则返回 -1。

- ❑ str: 指定检索的字符串。
- ❑ beg: 开始索引, 默认为 0。
- ❑ end: 结束索引, 默认为字符串的长度。

4.2.11 方法 `str.isalnum()`

在 Python 语言中, 方法 `str.isalnum()` 的功能是如果字符串中的所有字符都是字母数字且至少有一个字符, 则返回 `true`, 否则返回 `false`。

4.2.12 方法 `str.isdecimal()`

在 Python 语言中, 方法 `str.isdecimal()` 的功能是如果字符串中的所有字符都是十进制字符并且至少有一个字符, 则返回 `true`, 否则返回 `false`。十进制字符是来自通用类别 “Nd” 的字符。此类别包括数字字符, 以及可用于形成十进制数字的所有字符。要想定义一个十进制字符串, 只需要在字

字符串前添加 'u' 前缀即可。

4.2.13 方法 str.isdigit()

在 Python 语言中，方法 str.isdigit()的功能是如果字符串中的所有字符都是数字，并且至少有一个字符则返回真，否则返回假。数字包括十进制字符和需要特殊处理的数字，例如兼容性上标数字。在形式上，数字是具有属性值 Numeric_Type = Digit 或 Numeric_Type = Decimal 的字符。

4.2.14 方法 str.format()

在 Python 语言中，方法 str.isidentifier()的功能是检测字符串是否是字母开头，如果是，则返回 True。例如下面的演示过程：

```
'asdfghjkl'.isidentifier()  
Out[33]: True  
  
'2asdfghjkl'.isidentifier()  
Out[34]: False
```

4.2.15 方法 str.islower()

在 Python 语言中，方法 str.islower()的功能是如果字符串中的所有字符都是小写，并且至少有一个字符，则返回 True，否则返回 False。

4.2.16 方法 `str.isnumeric()`

在 Python 语言中，方法 `str.isnumeric()` 的功能是如果字符串中的所有字符都是数字字符，并且至少有一个字符，则返回 `true`，否则返回 `false`。数字字符包括数字字符和具有 Unicode 数字值属性的所有字符。

4.2.17 方法 `str.isprintable()`

在 Python 语言中，方法 `str.isprintable()` 的功能是如果字符串中的所有字符都可打印或字符串为空，则返回 `true`，否则返回 `false`。不可打印字符是在 Unicode 字符数据库中定义为“其他”或“分隔符”的字符，除了被认为是可打

印的 ASCII 空间 (0x20)。请注意, 在此上下文中的可打印字符是在字符串上调用 `repr()` 时不应转义的字符, 对处理写入 `sys.stdout` 或 `sys.stderr` 的字符串没有影响。

4.2.18 方法 `str.isspace()`

在 Python 语言中, 方法 `str.isspace()` 的功能是如果在字符串中只有空格字符, 并且至少有一个字符, 则返回 `True`, 否则返回 `False`。空格字符是在 Unicode 字符数据库中定义为 “其他” 或 “分隔符”, 并且具有双向属性为 “WS” “B” 或 “S” 之一的那些字符。

4.2.19 方法 `str.istitle()`

在 Python 语言中，方法 `str.istitle()`的功能是如果字符串是标题类型的字符串且至少包含一个字符，则返回 `true`。

例如：大写字符可能只能跟着非标题类（数字、符号和转义字符）的字符和小写字符，否则返回 `false`。

4.2.20 方法 `str.ljust()`

在 Python 语言中，方法 `str.ljust(width,fillchar)`的功能是得到一个原始字符串左对齐，并使用 `fillchar` 填充至指定长度的新字符串。如果指定的长度小于原字符串的长度，则返回原始字符串，与 `format` 的填充用法相似。

- ❑ width: 指定长度。
- ❑ fillchar: 填充字符串, 默认为空格字符。

4.2.21 方法 str.lower()

在 Python 语言中, 方法 str.lower()的功能是把所有字母转化为小写, 功能与 str.upper()相反。

4.2.22 方法 str.lstrip()

在 Python 语言中, 方法 str.lstrip(chars)的功能是删除 str 左边所有出现在 chars 子字符串, chars 为空时默认空格字符。

参数 “chars” 用于指定截取的字符。例如下面的演示过程:

```
' Wo Shi Hao ren '.lstrip()  
Out[67]: 'Wo Shi Hao ren '  
  
'Wo Shi Hao ren'.lstrip('fdsfsfW')
```

```
Out[68]: 'o Shi Hao ren'

'Wo Shi Hao ren'.rstrip('fdsfsfw')
Out[69]: 'Wo Shi Hao ren'
```

4.2.23 方法 str.maketrans()

在 Python 语言中，方法 `str.maketrans(x[, y[, z]])` 的功能是得到一个用于 `str.translate()` 的映射，其实就是一个字典。

如果只有一个参数，它必须是将 Unicode ordinals (整数) 或字符 (长度为 1 的字符串) 映射到 Unicode ordinal, 字符串 (任意长度) 或 None 的字典。字符键将被转换为序数。

如果有两个参数，它们必须是相等长度的字符串，并且在结果字典中，x 中的每个字符将被映射到 y 中相同位置的字符。如果有第三个参数，它必须是一个字符串，在结果

中这些字符将被映射到 “None”。

4.2.24 方法 str.partition()

在 Python 语言中，方法 str.partition(sep)的功能是在分隔符首次出现位置拆分字符串，并返回包含分隔符之前部分、分隔符本身和分隔符之后部分的三元组。如果找不到分隔符，返回包含字符串本身，跟着两个空字符串的三元组。

4.2.25 方法 str.replace()

在 Python 语言中，方法 str.replace(old,new,count)的功能是把字符串中的 old (旧字符串) 替换成 new (新字符串)，替换不超过 count 次，count 为空时不限次数。

- ❑ old: 将被替换的子字符串。
- ❑ new: 新字符串, 用于替换 old 子字符串。
- ❑ max: 可选字符串, 替换不超过 max 次。

4.2.26 方法 str.rfind()

在 Python 语言中, 方法 `str.rfind(sub[, start[, end]])` 的功能是返回被搜索子串最后一次出现在字符串的索引位置, 失败则返回-1。

- ❑ `str`: 查找的字符串。
- ❑ `beg`: 开始查找的位置, 默认为 0。
- ❑ `end`: 结束查找的位置, 默认为字符串的长度。

4.2.27 方法 str.rindex()

在 Python 语言中, 方法 `str.rindex(str, beg=0, end=len(string))` 的功能是返回子字符串 `str` 在字符串中最后出现的位置, 如

果没有匹配的字符串会报异常，用户可以指定可选参数 [beg:end] 设置查找的区间。返回子字符串 `str` 在字符串中最后出现的位置，如果没有匹配的字符串会报异常。

- ❑ `str`: 查找的字符串。
- ❑ `beg`: 开始查找的位置，默认为 0。
- ❑ `end`: 结束查找的位置，默认为字符串的长度。

4.2.28 方法 `str.rjust()`

在 Python 语言中，方法 `str.rjust(width[, fillchar])` 的功能是得到一个原始字符串右对齐，并使用 `fillchar` 填充至指定长度的新字符串。若指定的长度小于原字符串的长度，则返

回原始字符串。与 `format` 的填充用法相似。

- ❑ `width`: 指定填充指定字符后中字符串的总长度。
- ❑ `fillchar`: 填充的字符，默认为空格。

4.2.29 方法 `str.rpartition(char)`

在 Python 语言中，方法 `str.rpartition(char)` 的功能是根据字符串 `char` 分割 `str` 得到一个三元素元组（只识别最后一次出现的字符串）。参数 `char` 不能为空，表示指定的分隔符。

4.2.30 方法 `str.rsplit()`

在 Python 语言中，方法 `str.rsplit(sep=None, maxsplit=-1)` 的功能是在字符串中，使用 `sep` 作为分隔符字符串返回一个单

词列表。如果给出了 `maxsplit`，则最多分裂为 `maxsplit+1` 个元素，从最右边开始。如果未指定 `sep` 或 `None` 任何空格的字符串是一个分隔符。

4.2.31 方法 `str.rstrip()`

在 Python 语言中，方法 `str.rstrip()` 的功能是删除 string 字符串末尾的指定字符（默认为空格）。

4.2.32 方法 `str.rstrip([chars])`

在 Python 语言中，方法 `str.rstrip([chars])` 的功能是返回一个移去尾部字符后的字符串的副本。参数 `chars` 是一个字符串，指定要移除的字符集。如果省略 `chars` 参数，则默认

为删除空格。

4.2.33 方法 `str.split()`

在 Python 语言中, 方法 `str.split(sep=None, maxsplit=-1)` 的功能是在字符串中, 使用参数 `sep` 作为分隔符分割字符串, 返回分割后的列表。如果给出了 `maxsplit` 参数, 则至多拆分 `maxsplit` 次 (因此, 列表中将最多有 `maxsplit+1` 个元素)。如果没有指定 `maxsplit` 或为 -1, 那么分割的数量没有限制 (进行所有可能的分割)。

如果给定了 `sep` 参数, 连续的分隔符不分组在一起, 并被视为空字符串进行分隔 (例如, `'1,,2'.split(',')` 返回 `['1', '', '2']`)。

参数 `sep` 可以由多个字符组成 (例如, `'1<>2<>3'.split('<>')` 返回 `['1', '2', '3']`)。用指定的分隔符分隔空字符串返回 `['']`。

4.2.34 方法 `str.splitlines()`

在 Python 语言中, 方法 `str.splitlines(keepends)` 的功能是按照行 (`'\r'`, `'\r\n'`, `'\n'`) 进行分隔, 得到各行元素的列表。如果 `keepends` 为 `False`, 不包含换行符。如果为 `True`, 则保留换行符。默认为 `False`。参数 “`keepends`” 表示在输出结果里是否去掉换行符 (`'\r'`, `'\r\n'`, `'\n'`), 默认为 `False`, 不包含换行符。如果为 `True`, 则保留换行符。

4.2.35 方法 str.startswith()

在 Python 语言中, 方法 `str.startswith(prefix[, start[, end]])` 的功能是如果字符串以 `prefix` 开头, 则返回 `True`, 否则返回 `False`。

- ❑ `prefix`: 检测的字符串, 也可以是一个需要查找的前缀元组。
- ❑ `start`: 可选参数用于设置字符串检测的起始位置。
- ❑ `end`: 可选参数用于设置字符串检测的结束位置。

4.2.36 方法 str.strip()

在 Python 语言中, 方法 `str.strip([chars])` 的功能是返回字符串的一个副本, 删除前导和尾随字符。参数 `chars` 是一

个字符串，指定要移除的字符集。如果要省略或 `chars` 参数，则默认为删除空格。参数 `chars` 不是前缀或后缀；相反，它的值的所有组合被去除，例如下面的演示过程：

```
>>>
>>> '   spacious '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

4.2.37 方法 `str.translate()`

在 Python 语言中，方法 `str.translate(table[, deletechars])`；的功能是返回通过给定的翻译表映射每个字符的字符串的副本，该 `table` 必须是通过 `__getitem__()` (通常为 `mapping` 或 `sequence`) 实现索引的对象。当使用 Unicode 序号（整数）索引时，`table`

对象可以执行以下任何操作：返回 Unicode 序号或字符串，将字符映射到一个或多个其他字符；return None，从返回字符串中删除字符；或引发 LookupError 异常，将字符映射到自身。

❑ table：翻译表，翻译表是通过 maketrans()方法转换而来。

❑ deletechars：字符串中要过滤的字符列表。

4.2.38 方法 str.upper()

在 Python 语言中，方法 str.upper()的功能是把所有字母转化为大写形式。

4.2.39 方法 str.zfill()

在 Python 语言中，方法 `str.zfill(width)` 的功能是在字符串 `str` 前面填充字符串 `'0'`，使长度为指定长度 `width`。参数 `width` 表示指定长度，原字符串右对齐，前面填充 `0`。

4.3 列表处理

4.3.1 方法 list.append(obj)

此方法的功能是在列表末尾添加新的对象，`append()`

方法的语法格式如下：

```
list.append(obj)
```

参数 `obj` 表示添加到列表末尾的对象。

4.3.2 方法 list.count(obj)

此方法的功能是统计某个元素在列表中出现的次数，

count()方法的语法格式如下：

```
list.count(obj)
```

参数 obj 表示列表中统计的对象。

4.3.3 方法 list.extend(seq)

此方法的功能是在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表），extend()方法的语法格式如下：

```
list.extend(seq)
```

参数 seq 表示元素列表。

4.3.4 方法 list.index(obj)

此方法的功能是从列表中找出某个值第一个匹配项的索引位置，index()方法的语法格式如下：

```
list.index(obj)
```

参数 obj 表示查找的对象。

4.3.5 方法 list.insert(index, obj)

此方法的功能是将对象插入列表，insert()方法的语法格式如下：

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

- ❑ index：对象 obj 需要插入的索引位置。
- ❑ obj：要插入列表中的对象。

4.3.6 方法 `list.pop([index=-1])`

此方法的功能是移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。`pop()`方法的语法格式如下：

```
list.pop([index=-1])
```

参数 `index` 是一个可选参数，表示要移除列表元素的索引值，不能超过列表总长度，默认为 `index=-1`，删除最后一个列表值。

4.3.7 方法 `list.remove(obj)`

此方法的功能是移除列表中某个值的第一个匹配项，`remove()`方法的语法格式如下：

```
list.remove(obj)
```

参数 obj 表示列表中要移除的对象。

4.3.8 方法 list.reverse()

此方法的功能是反向列表中元素，reverse()方法的语格式如下：

```
list.reverse()
```

4.3.9 方法 list.sort(cmp=None, key=None, reverse=False)

此方法的功能是对原列表进行排序，sort()方法的语格式如下：

```
list.sort(cmp=None, key=None, reverse=False)
```

- ❑ `cmp`: 可选参数, 如果指定了该参数会使用该参数的方法进行排序。
- ❑ `key`: 主要是用来进行比较的元素, 只有一个参数, 具体的函数的参数就是取自于可迭代对象中, 指定可迭代对象中的一个元素来进行排序。
- ❑ `reverse`: 排序规则, `reverse = True` 降序, `reverse = False` 升序 (默认)。

4.3.10 方法 `list.clear()`

此方法的功能是清空列表, `clear()`方法的语法格式

如下:

```
list.clear()
```

4.3.11 方法 list.copy()

此方法的功能是复制列表，copy()方法的语法格式如下：

```
list.copy()
```

4.4 元组处理

4.4.1 方法 len(tuple)

此方法的功能是计算元组元素个数，例如：

```
>>> tuple1 = ('Google', 'Runoob', 'Taobao')
>>> len(tuple1)
3
>>>
```

4.4.2 方法 max(tuple)

此方法的功能是返回元组中元素最大值，例如：

```
>>> tuple2 = ('5', '4', '8')
>>> max(tuple2)
'8'
```

```
>>>
```

4.4.3 方法 min(tuple)

此方法的功能是返回元组中元素最小

值, 例如:

```
>>> tuple2 = ('5', '4', '8')
>>> min(tuple2)
'4'
>>>
```

4.4.4 方法 tuple(seq)

此方法的功能是将列表转换为元组, 例如:

```
>>> list1= ['Google', 'Taobao', 'Runoob', 'Baidu']
>>> tuple1=tuple(list1)
>>> tuple1
('Google', 'Taobao', 'Runoob', 'Baidu')
```

4.5 字典处理

4.5.1 方法 `radiansdict.clear()`

此方法的功能是删除字典内所有元素，`clear()`方法的

语法格式如下：

```
dict.clear()
```

4.5.2 方法 `radiansdict.copy()`

此方法的功能是返回一个字典的浅复制，`copy()`方法的

语法格式如下：

```
dict.copy()
```

4.5.3 方法 `radiansdict.fromkeys()`

此方法的功能是创建一个新字典，以序列 `seq` 中元素

做字典的键，val 为字典所有键对应的初始值。fromkeys()

方法的语法格式如下：

```
dict.fromkeys(seq[, value])
```

- ❑ seq，字典键值列表。
- ❑ value，可选参数，设置键序列 (seq) 的值。

4.5.4 方法 radiansdict.get(key, default=None)

此方法的功能是返回指定键的值，如果值不在字典中，返回 default 值。get()方法的语法格式如下：

```
dict.get(key, default=None)
```

- ❑ key：字典中要查找的键。
- ❑ default：如果指定键的值不存在时，返回该默

认值。

4.5.5 方法 `key in dict`

Python 字典 `in` 操作符用于判断键是否存在于字典中，如果键在字典 `dict` 里返回 `true`，否则返回 `false`。`in` 操作符的语法：

```
key in dict
```

参数 `key` 表示要在字典中查找的键。

4.5.6 方法 `dict.items()`

此方法的功能是以列表返回可遍历的（键，值）元组数组。`items()`方法的语法：

```
dict.items()
```

4.5.7 方法 `radiansdict.keys()`

此方法的功能是以列表返回一个字典所有的键。 `keys()`

方法的语法：

```
dict.keys()
```

4.5.8 方法 `radiansdict.setdefault(key, default=None)`

此方法的功能是和 `get()`类似，但如果键不存在于字典中，将会添加键并将值设为 `default`。 `setdefault()`方法的

语法：

```
dict.setdefault(key, default=None)
```

- ❑ `key`：查找的键值。
- ❑ `default`：键不存在时，设置的默认键值。

- ❑ 返回值：如果 key 在字典中，返回对应的值。如果不在字典中，则插入 key 及设置的默认值 default，并返回 default，default 默认值为 None。

4.5.9 方法 `radiansdict.update(dict2)`

此方法的功能是把字典 dict2 的键/值对更新到 dict 里。

update()方法的语法：

```
dict.update(dict2)
```

参数 dict2 表示添加到指定字典 dict 里的字典。

4.5.10 方法 `radiansdict.values()`

此方法的功能是以列表返回字典中的所有值。values()

方法的语法：

```
dict.values()
```

4.5.11 方法 pop(key[,default])

此方法的功能是删除字典给定键 key 所对应的值，返回值为被删除的值。key 值必须给出。否则，返回 default 值。pop()方法的语法：

```
pop(key[,default])
```

- ❑ key：要删除的键值。
- ❑ default：如果没有 key，则返回 default 值。

4.5.12 方法 popitem()

此方法的功能是随机返回并删除字典中的一对键和值（一般删除末尾对）。popitem()方法的语法：

```
popitem()
```