第3章

Python 3语言基础

3.1 Python 3 简介

Python 是一个高层次的结合了解释性、编译性、交互性和面向对象 4 个特点的脚本语言。

Python 的设计具有极强的可读性，相比其他语言，Python 经常使用英文关键字、标点符号，具有比其他语言更具特色的语法结构。

Python 具有以下显著特点。

1. 解释性

这意味着开发过程中没有了编译这个环节。类似于 PHP 和 JavaScript 语言。

2. 交互性

这意味着用户可以在一个 Python 控制台终端（提示符>>>后）直接执行代码，如图 3.1 所示。

图 3.1 Python 控制台终端编程

3. 面向对象

这意味着 Python 支持面向对象编程的风格或代码封装在对象中的编程技术。与其他主要语言(如 Java)相比,Python 以一种非常强大且简单的方式实现了面向对象编程。

4. 简单易学

Python 是一种对初学者十分友好的语言,一个风格良好的 Python 程序如同英文段落一样易懂。Python 的最大优点之一是具有伪代码的本质,它使用户在开发 Python 程序时专注于解决问题,而不是搞明白语言本身。

3.1.1 Python 的发展历史

20 世纪 80 年代末至 90 年代初,Python 由 Guido van Rossum 在荷兰数学和计算机科学研究学会设计出来。

Python 本身是由诸多其他语言发展而来的,包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、UNIX Shell 和其他脚本语言。

当前,Python 由一个核心开发团队进行系列维护工作,但 Guido van Rossum 仍然在其中发挥着至关重要的作用,指导 Python 的相关工作。

1989 年,Guido van Rossum 为消遣时间,决定为当时正构思的一个新脚本语言编写一个解释器,以 Python 命名该项目,使用 C 语言进行开发。

1991 年,Python 的第一个版本发布。此时 Python 已经具有了类、函数、异常处理、包含表和词典在内的核心数据类型以及以模块为基础的拓展系统。

1991—1994 年,Python 增加了 lambda、map、filter 和 reduce 模块。

1999 年,Python 的 Web 框架之祖——Zope 发布。

2000 年,Python 加入了内存回收机制,构成了现在 Python 语言框架的基础。

2004 年,Web 框架 Django 诞生。

2006 年,Python 2.5 发布。

……

2019 年,Python 3.8 发布。

2020 年,对 Python 2.0 版本的支持停止。

3.1.2 Python 的应用

Python 作为一种简单好用的编程语言,广泛应用于以下领域,如图 3.2 所示。

1. Web 开发

Python 经常被用于 Web 开发。例如,在 mod_wsgi 模块、Apache mod_wsgi 模块上可以运行用 Python 编写的 Web 程序。还有一些 Web 框架(如 Django、TurboGears、web2py 等)可以让程序员轻松地开发和管理复杂的 Web 程序。

图 3.2 Python 的应用领域

2. 系统运维

Python 在不断进化，它的功能已经延伸到 IT 运维的方方面面。Python 拥有强大的脚本处理功能，它在操作 Linux 系统方面具有先天的优势。许多云平台、运维监控管理工具均使用 Python 开发，Python 的自动化运维可以减少运维工程师的工作量，提高工作效率。

3. 科学计算

Python 第三方库提供的 NumPy、SciPy、Matplotlib 等模块能让 Python 程序员快速编写科学计算程序。

4. 桌面软件

PyQt、PySide、wxPython、PyGTK 是 Python 快速开发桌面应用程序的模块。

5. 游戏开发

除了在科学计算领域占据一席之地之外，Python 在游戏、后台等领域也大放异彩。pygame 是一组专门为编写游戏而设计的 Python 模块，可以使用户在 Python 语言中轻松地创建全功能的游戏和多媒体程序。

3.2 环境搭建

3.2.1 Python 3 环境的搭建

关于 Python 3 环境搭建的讲解视频可扫描二维码观看。

Python 3 可应用于多平台，包括 Windows、Linux 和 Mac OS X。本节将讲解如何在本地搭建 Python 3 开发环境。

1. 在 Windows 环境下安装 Python

1) Python 3 的下载

Python 3 最新源代码、二进制文档、新闻资讯等可以在 Python 的官方网站查看，

Python 的官方网站地址为 https://www.python.org/，如图 3.3 所示。

图 3.3　下载 Python 的界面

选择下载版本，下载对应系统的可执行安装文件，如图 3.4 所示。

图 3.4　选择下载版本

2）Python 的安装

双击下载的可执行文件 python-3.x.exe，按提示进行安装，直到安装完成。

安装 Python 的过程中会自动配置环境变量，如果没有自动生成，可以手动配置环境变量。右击“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，弹出“系统属性”对话框，在对话框中选择“高级”选项卡，单击“环境变量”按钮，如图 3.5 所示。

图 3.5　设置环境变量

在变量名为 Path 的变量值的尾部添加 Python 3 安装目录即可，如图 3.6 所示。

图 3.6 添加环境变量

3）验证

验证 Python 是否安装成功，可以通过在控制台终端输入 python -V 的方式进行验证，如果出现 Python 安装版本，则说明 Python 安装成功，如图 3.7 所示。

图 3.7 验证是否安装成功

2. 在 Linux 环境下安装 Python

在 Linux 系统中，一般情况下都预装有 Python 编译器，但是这个预装的 Python 版本一般比较低，许多特性都没有，如果要使用新版本的 Python 必须重新安装，如下所示：

```
[root@Master001 ~]# python -V
Python 2.6.6
[root@Master001 ~]#
```

注意，安装新的 Python 版本时，不需要删除旧的 Python 版本，因为 Linux 系统中有些命令仍需要旧的 Python 版本的支持，例如 yum。

下载 Python 安装包可以在 Python 官方网站下载，也可以使用 wget 工具下载。

（1）在 Python 官方网站下载，地址为 https://www.python.org/downloads/，如图 3.8 所示。

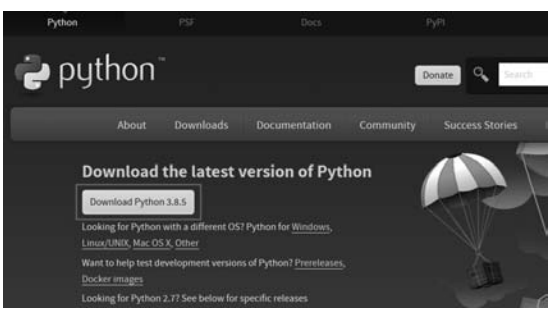

图 3.8 下载 Linux 版本的 Python

（2）使用 wget 工具下载资源在 Linux 系统中是最常用的下载方法之一。在使用 wget 命令之前需要先安装 wget 工具，wget 工具可以使用 yum 命令进行安装，操作方法如下：

```
[root@Master001 ~]# yum install -y wget
```

验证是否安装 wget：

```
[root@Master001 ~]# rpm -qa | grep wget
wget-1.12-10.el6.x86_64
```

使用 wget 命令获取安装包：

```
[root@Master001 ~]# wget https://www.python.org/ftp/python/3.6.2/Python-3.6.2.tgz
```

（3）安装 Python。

将 wget 下载的 Python 包解压：

```
[root@Master001 software]# pwd
    /root/software
[root@Master001 software]# tar -zxvf Python-3.6.2.tgz
```

配置 configure 文件，进入 Python 解压目录，执行 configure 命令。因为执行 configure 命令需要 gcc 工具的支持，所以需要先安装 gcc 工具。

```
[root@Master001 Python-3.6.2]# yum -y install gcc
```

执行 configure 命令：

```
[root@Master001 Python-3.6.2]# ./configure
```

configure 命令执行后会在当前目录生成 Makefile 文件，该文件将在 make 解析时使用。

编译完成后不要着急执行 make 命令，因为 pip 工具在下载安装时需要 SSL 的支持，默认的 SSL 与系统自带的 SSL 不一致，因此会导致 pip 工具在使用过程中不能正常下载。

配置 SSL 证书。使用 vim 命令打开 Pythonxx/Modules/Setup 文件，该文件必须要执行 configure 命令才能生成。打开 Setup 将 SSL 的地址修改为系统自带的 SSL 地址即可。

查询 SSL 地址，如图 3.9 所示。

```
[root@Master001 Python-3.6.2]# openssl version -a
```

```
[root@Master001 Python-3.6.2]# openssl version -a
OpenSSL 1.0.1e-fips 11 Feb 2013
built on: Wed Mar 22 21:43:28 UTC 2017
platform: linux-x86_64
options:  bn(64,64) md2(int) rc4(16x,int) des(idx,cisc
compiler: gcc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THRE
64 -DL_ENDIAN -DTERMIO -Wall -O2 -g -pipe -Wall -Wp,-D_
sp-buffer-size=4 -m64 -mtune=generic -Wa,--noexecstack
SSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA
AES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
OPENSSLDIR: "/etc/pki/tls"
```

图 3.9 查询 SSL 地址

修改 SSL，如图 3.10、图 3.11 所示。

```
[root@Master001 Python-3.6.2]# vim Modules/Setup
```

```
# Socket module helper for SSL support; you must comment out the other
# socket line above, and possibly edit the SSL variable:
#SSL=/usr/local/ssl
#_ssl _ssl.c \                                          没有修改前
#       -DUSE_SSL -I$(SSL)/include -I$(SSL)/include/openssl \
#       -L$(SSL)/lib -lssl -lcrypto
```

图 3.10 SSL 修改前

```
# Socket module helper for SSL support; you must comment out the other
# socket line above, and possibly edit the SSL variable:
SSL=/etc/pki/tls
_ssl _ssl.c \                                          修改之后
       -DUSE_SSL -I$(SSL)/include -I$(SSL)/include/openssl \
       -L$(SSL)/lib -lssl -lcrypto
```

图 3.11 SSL 修改后

使用 make 工具编译源代码，执行 make 命令需要 openssl 和 openssl-devel 的支持。

```
[root@Master001 Python-3.6.2]# yum -y install openssl.x86_64 openssl-devel.x86_64
[root@Master001 Python-3.6.2]# make
```

make install 命令用于安装 make 编译好的源代码。

```
[root@Master001 Python-3.6.2]# make install
```

使用 whereis 命令查询 Python 3 的安装路径，若查询到则说明 Python 3 安装成功，如图 3.12 所示。

```
[root@Master001 Python-3.6.2]# whereis python3
python3: /usr/bin/python3 /usr/local/bin/python3.6m-config /usr/local/bin/
cal/bin/python3.6 /usr/local/bin/python3.6m /usr/local/lib/python3.6
[root@Master001 Python-3.6.2]# whereis pip3
pip3: /usr/local/bin/pip3.6 /usr/local/bin/pip3
```

图 3.12 查询 Python 3 的安装路径

Python 3 编译安装完成后，默认的安装目录是在/usr/local/bin 和/usr/local/lib 目录下。如果想要卸载刚安装好的 Python 编译器，直接删除这两个目录即可，如图 3.13 所示。

(4) 配置 Python 3 环境。

Hadoop Streaming 工具提交 MapReduce 任务时需要 Python 3 编译器的支持，集群

图 3.13　Python 3 的安装位置

默认是在/usr/bin 下找指定的 Python 3 编译器，但是/usr/bin 中只有 Python 2 的编译器，所以在运行作业时可能会出现语法错误。为了解决这个问题，需要给 Python 3 的执行命令添加软链接到/usr/bin 目录。

将 Python 3 可执行文件同步到/usr/bin 目录中：

```
[root@Master001 Python-3.6.2]# ln -s /usr/local/bin/python3 /usr/bin/python3
```

将 pip3 可执行文件同步到/usr/bin 目录中：

```
[root@Master001 Python-3.6.2]# ln -s /usr/local/bin/pip3 /usr/bin/pip3
```

3.2.2　PyCharm

关于 PyCharm 安装与使用的讲解视频可扫描二维码观看。

1. 安装 PyCharm

PyCharm 是一种 Python IDE，带有一整套可以帮助用户在使用 Python 语言开发时提高效率的工具，如调试、语法高亮、目录管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。此外，该 IDE 还提供了一些高级功能，以用于支持 Django 框架下专业 Web 的开发。

官方网站下载地址为 https://pycharm.en.softonic.com/。

下载 PyCharm 时有两种选择：一种是专业版（Professional）；另一种是社区版（Community）。虽然社区版 PyCharm 比专业版 PyCharm 的功能少一些，但是社区版 PyCharm 是免费的。对于学习而言，社区版的功能完全能够满足用户实际需求，所以可以选择社区版进行下载，如图 3.14 所示。

图 3.14　选择 PyCharm 版本

(1) 运行 PyCharm 安装文件。双击可执行文件 pycharm-community-2019.3.2.exe 即可运行该文件。

(2) 单击 Browse 按钮,修改安装路径,单击 Next 按钮,如图 3.15 所示。

图 3.15 设置安装路径

(3) 在弹出的对话框中勾选 Create Desktop Shortcut 复选框用于创建桌面快捷方式,勾选 Create Associations 复选框用于创建关联文件,如图 3.16 所示。

图 3.16 选择安装方式

在图 3.16 中,配置完成后单击 Next 按钮进入下一项配置,在下一项配置中单击 Install 按钮进行软件安装。

2. 创建 PyCharm 项目

(1) 双击 PyCharm Community Edition 2019.3.2 x64 图标,打开 PyCharm 工具,如图 3.17 所示。

(2) 单击 Create New Project 按钮,创建新项目,如图 3.18 所示。

(3) 设置 Python 项目存放位置,并选择所需要使用的解释器,如图 3.19 所示。

(4) 选择已经安装的 Python 解释器和接口,如图 3.20～图 3.22 所示。

图 3.17　双击 PyCharm Community Edition 2019.3.2 x64 图标

图 3.18　创建新项目

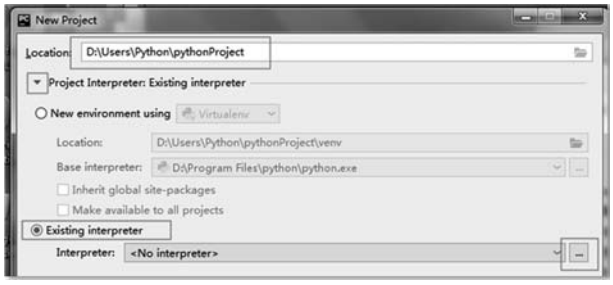

图 3.19　设置 Python 项目存储位置

图 3.20　Python 解释器

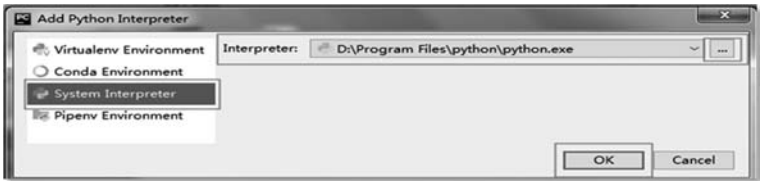

图 3.21　选择 Python 解释器

图 3.22　修改 Python 解释器后的效果

(5) 单击 Create 按钮,创建 Python 项目。

(6) 创建 Python 文件,编写第一个 Python 项目,如图 3.23 所示。

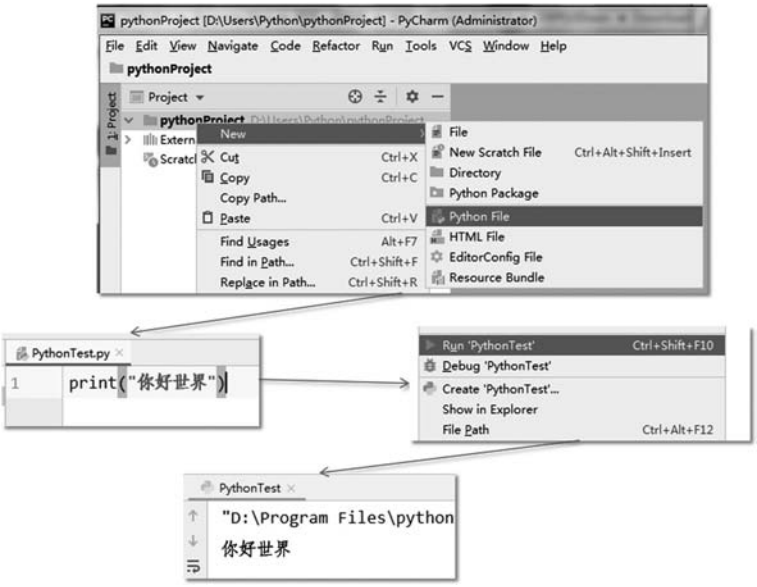

图 3.23 创建项目过程

3.3 基础语法

3.3.1 语法规范

关于语法规范的讲解视频可扫描二维码观看。

1. 注释

代码更多是用来“读”而不是用来“写”的。良好的编码规范可以使项目、模块或函数保持一致,从而提高 Python 代码的可读性。

注释分为单行注释与多行注释。示例代码如下:

```
# 这是单选注释
print("学到教育")

'''
    多行注释可以用 3 个单引号
'''
"""
    多行注释也可以使用 3 个双引号
    在 Python 中大多数情况单引号和双引号的作用一样,
    均代表字符串
"""
```

2. 缩进

每级缩进用 4 个空格，Python 不需要使用大括号({})来组织代码，完全依靠缩进。因此，缩进的格式非常重要，如果使用的缩进不是 4 个空格，则会报语法错误。在开发过程中，可以使用 Tab 键代替 4 个空格。

示例代码如下：

```
a = 5
b = 3
if a > b:
    print("大于")
else:
    print("小于")
```

输出结果：

```
大于
```

3. 分号

Python 不严格要求使用分号，理论上应该是每行一句代码。每行代码后面可以添加分号，也可以不添加分号，尽量不要多句代码放在一行。如果放在一行，则需要添加分号把它们隔开。

示例代码如图 3.24 所示。

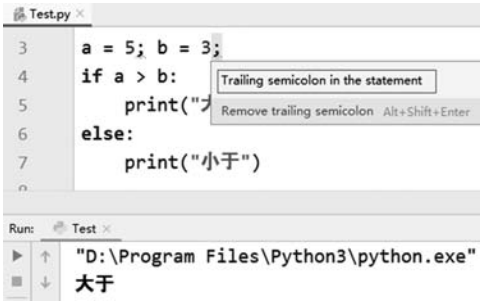

图 3.24　分号使用示例

通过图 3.24 可知，虽然在 Python 的代码中加上分号后仍能够正常运行，但是 Python 编译器建议去掉分号。

4. 标识符

标识符是在程序中对变量、常量、类、方法、参数等命名时使用的名字。命名时应该遵循以下规则：

(1) 标识符由字母、下画线、数字组成,且不能以数字开头。

(2) Python 对大小写敏感,a 和 A 是完全不同的。

(3) 不能使用 Python 关键字命名。

(4) 命名应该产生见名知义的效果。

(5) 命名推荐使用驼峰式命名(studentName)和下画线命名(student_name)。

5. 多行语句

Python 语句中一般以新行作为语句的结束符,也可以使用斜杠(\)将一行语句划分为多行显示。

示例代码如下:

```
itemOne = 1
itemTwo = 2
itemThree = 3
total = itemOne + \
    itemTwo + \
    itemThree
print(total)
```

输出结果:

```
6
```

语句中包含[]、{}、()符号的则不需要使用多行连接符。

```
days = ["星期一", "星期二",
"星期三", "星期四",
"星期五", "星期六", "星期日"]
print(days)
```

输出结果:

```
['星期一', '星期二', '星期三', '星期四', '星期五', '星期六', '星期日']
```

6. Python 引号

Python 可以使用单引号、双引号、三引号来表示字符串,开始与结束的引号必须是同一类型。

其中,三引号可以由多行组成,是编写多行文本的快捷语法,常用于文档字符串,也可以当成多行注释,如图 3.25 所示。

```
a = '单引号字符串'
b = "双引号字符串，与单引号字符串效果一样"
c = """这是一个段落字符串，它可以编写多行文本，
在Python中，每行不能超过80个字符，如果要
定义文本超过80字符的字符串时，可以使用三
引号来定义多行文本。
"""
print(c)
```

图 3.25　引号使用示例

7. 输出

```
a = "学到"
b = "教育"

print("换行输出：")
print(a)
print(b)

print("不换行输出：")
print(a, b)
```

输出结果：

```
换行输出：
学到
教育
不换行输出：
学到 教育
```

当 Python 使用加号进行连接时，只能连接同一类型的值，如果连接不同类型的值，将会出现错误，如图 3.26 所示。

```
TypeError: can only concatenate str (not "int") to str
```

图 3.26　输出结果

print()函数提供 sep 与 end 参数控制输出结果。

sep：分隔值与值，默认是一个空格。

end：为末尾传递一个指定的字符串。

示例代码如下：

```
print("AA", "BB", "CC", sep = ":")
print("DDD")

print("------------------------")

print("AA", "BB", "CC", end = ":")
print("DDD")
```

输出结果：

```
AA:BB:CC
DDD
------------------------
AA BB CC:DDD
```

3.3.2 数据类型

关于数据类型的讲解视频可扫描二维码观看。

Python 中的变量不需要声明，但每个变量在使用前都必须赋值，赋值以后该变量才会被创建。

在 Python 中，变量就是变量，变量没有类型，只有为它赋值后，才会根据值为它推导出类型。

等号(=)用来给变量赋值；等号左边是变量名，等号右边是存储在变量中的值。

示例代码如下：

```
a = 100                    # 整型变量
b = 100.0                  # 浮点型变量
c = "学到教育"             # 字符串变量
d = 'A'                    # 字符串变量,等同于"A"
f = True                   # 布尔类型变量
```

在 Python 中允许同时为多个变量赋值。

创建一个整型对象，值为 100，将 100 赋值给 c 变量后，又将 c 变量赋值给 b 变量，最后把 b 变量赋值给 a 变量。示例代码如下：

```
a = b = c = 100
print(a, b, c)
```

输出结果：

```
100 100 100
```

也可以为多个变量指定多个值。示例代码如下：

```
a, b, c = 1, 99.8, "学到"
print(a, b, c)

x, y = (20, 100)
print(x, y)

a1, b1, c1, d1 = {1, 2, 3, 4}
print(a1, b1, c1, d1)
```

输出结果：

```
1 99.8 学到
20 100
1 2 3 4
```

在 Python 中定义了 6 个标准类型，用于存储各种类型的数据，其中又分为不可变数据类型和可变数据类型。不可变数据类型有 Number(数字)、String(字符串)、Tuple(元组)；可变数据类型有 List(列表)、Dictionary(字典)、Set(集合)。

1. 条件语句

Python 条件语句是通过一条或多条语句的执行结果(true 或 false)来决定执行的代码块。

执行流程如图 3.27 所示。

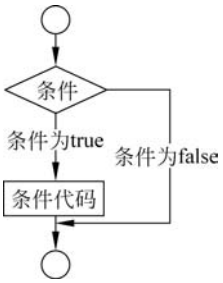

图 3.27　条件语句流程图

Python 编程中 if 语句用于控制程序的执行，基本形式为：

```
if 判断条件:
    语句块…
else:
    语句块…
```

示例代码如下：

```
flag = "A"
name = "张三"

if name == "李四":
    flag = "B"
else:
    print(name)

print(flag)
```

输出结果：

```
张三
A
```

因为 Python 不支持 switch 语句，所以多个条件判断时只能用 elif 来实现判断。当需要多个条件同时判断时，可以使用 or 或者 and。

当判断条件为多个值时，可以使用以下形式：

```
if 判断条件 1:
    语句块 1…
elif 判断条件 2:
    语句块 2…
elif 判断条件 3:
    语句块 3…
else:
    语句块…
```

示例代码如下：

```
inputAge = input("请输入您的年龄: ")
age = int(inputAge)
print("年龄: ", age)

if(age >= 0) and (age <= 12):
    print("儿童")
elif(age >= 12) and (age <= 18):
    print("青少年")
elif(age >= 18) and (age <= 150):
    print("成年人")
else:
    print("年龄不正确")
```

输出结果：

```
请输入您的年龄: 10
年龄: 10
儿童
```

Python 中支持 if 语句的嵌套，可以在对条件进行判断之后，再次对条件进行判断。示例代码如下：

```
strA = ["A", "张三", 22]

if strA[0] == "A":
    if strA[1] == "张三":
```

```
            strA[1] = "李四"
            print(strA)
        else:
            print(strA)
    else:
        print(strA[1])
```

输出结果：

```
['A', '李四', 22]
```

2. 循环语句

在 Python 中只提供了 for 循环和 while 循环两种循环语句。

循环语句允许多次执行一个语句或语句组。循环语句流程如图 3.28 所示。

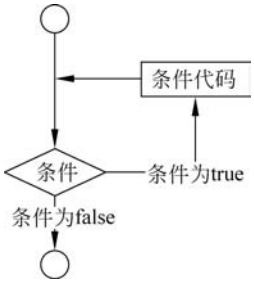

图 3.28　循环语句流程图

while 循环在给定的判断条件为 true 时执行循环体，否则退出循环体。示例代码如下：

```
a = 0
while a <= 5:
    print("每次输出的值：", a)
    a += 1
```

输出结果：

```
每次输出的值：0
每次输出的值：1
每次输出的值：2
每次输出的值：3
每次输出的值：4
每次输出的值：5
```

“:=”为赋值运算符，因形似海象，所以被称为海象运算符。海象运算符常在 while、if 等语句中使用。示例代码如下：

```
a = 0
while (a := a + 1) <= 5:
    print("每次输出的值: ", a)
```

输出结果：

```
每次输出的值: 0
每次输出的值: 1
每次输出的值: 2
每次输出的值: 3
每次输出的值: 4
每次输出的值: 5
```

在 Python 中，while…else 在循环条件为 false 时执行 else 语句块。示例代码如下：

```
count = 0
while count < 5:
    print("输出的值: ", count)
    count = count + 1
else:
    print("最后输出结果: ", count)
```

输出结果：

```
输出的值: 0
输出的值: 1
输出的值: 2
输出的值: 3
输出的值: 4
最后输出结果: 5
```

for 循环可以遍历任何序列的项目，如一个列表或者一个字符串。for 循环的语法格式如下：

```
for 迭代变量  in 列表
    语句块…
```

for 循环的示例代码如下：

```
sVal1 = "www.baidu.com"
for a in sVal1:
    print(a, end = " * ")

print()
sVal2 = ["Java", "Python", "Scala", "Spark", "Hadoop"]
for b in sVal2:
    print("遍历集合的值: ", b)
```

输出结果：

```
w*w*w*.*b*a*i*d*u*.*c*o*m*
遍历集合的值: Java
遍历集合的值: Python
遍历集合的值: Scala
遍历集合的值: Spark
遍历集合的值: Hadoop
```

for 循环还可以通过索引迭代方式遍历集合元素。其中 len()函数用于返回列表长度，range()函数用于返回一个序列的数。示例代码如下：

```
sVal1 = "www.baidu.com"
for a in range(len(sVal1)):
    print(sVal1[a], end = "*")

print()
sVal2 = ["Java", "Python", "Scala", "Spark", "Hadoop"]
for b in range(len(sVal2)):
    print("遍历集合的值: ", sVal2[b])
```

输出结果：

```
w*w*w*.*b*a*i*d*u*.*c*o*m*
遍历集合的值: Java
遍历集合的值: Python
遍历集合的值: Scala
遍历集合的值: Spark
遍历集合的值: Hadoop
```

range()函数用于返回整数序列的对象，语法格式如下：

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

参数如下。

start：计数从 start 开始，默认从 0 开始。

stop：计数到 stop 结束，但不包括 stop。

step：步长，默认为 1。

示例代码如下：

```
print(range(20))
print(range(10, 20))
print(range(10, 20, 2))
print(list(range(10, 20, 2)))
```

输出结果：

```
range(0, 20)
range(10, 20)
range(10, 20, 2)
[10, 12, 14, 16, 18]
```

在 Python 中，for…else 中的 else 表示 else 语句在 for 语句正常执行完的前提下再执行。示例代码如下：

```
for a in range(5):
    print("值: ", a)
else:
    print("for 循环退出")
```

输出结果：

```
值: 0
值: 1
值: 2
值: 3
值: 4
for 循环退出
```

Python 语言允许在一个循环体内嵌入另一个循环。示例代码如下：

```
# for 循环嵌套
strVal = [["张三", "女", 20],
          ["李四", "男", 18],
          ["王五", "男", 22]]
for a in strVal:
    for b in range(len(a)):
        print(a[b], end = "\t")

print("----------------------")
# while 循环嵌套
i = 0
while i < len(strVal):
    print(strVal[i])
    j = 0
    while j < len(strVal[i]):
        print(strVal[i][j])
        j = j + 1
    i = i + 1
```

输出结果：

```
张三  女  20  李四  男  18  王五  男  22  --------------------
['张三', '女', 20]
张三
女
20
['李四', '男', 18]
李四
男
18
['王五', '男', 22]
王五
男
```

3. 循环控制语句

循环控制语句可以更改语句的执行顺序，Python 支持的循环控制语句如表 3.1 所示。

表 3.1　循环控制语句

控 制 语 句	描　　述
break	在语句块执行过程中终止循环，并且跳出整个循环
continue	在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环
pass	空语句，为了保持程序结构的完整性

Python 中的 break 语句就像是在 Java 语言中打破了最小封闭的 for 或 while 循环。

break 语句用来终止循环语句，即循环条件中没有 false 条件或者序列还没被完全递归完，也会被停止执行循环语句。

break 语句用在 while 和 for 循环中，如果用户使用嵌套循环，那么 break 语句将停止执行最深层的循环，并开始执行下一行代码。

示例代码如下：

```
sVal = "baidu"
for a in sVal:
    if a == "d":
        print("遇到字母 d,退出系统")
        break
    print(a)
```

输出结果：

```
b
a
i
遇到字母 d,退出系统
```

Python 中的 continue 语句是指跳出本次循环。continue 语句用来告诉 Python 跳过当前循环，然后继续进行下一轮循环。continue 语句可以用在 while 和 for 循环中。

示例代码如下：

```
for a in range(5):
    for b in range(5):
        if b == 2:
            continue
        print(b, end = "")
    print("外层循环: ", a)
```

输出结果：

```
0 1 3 4 外层循环: 0
0 1 3 4 外层循环: 1
0 1 3 4 外层循环: 2
0 1 3 4 外层循环: 3
0 1 3 4 外层循环: 4
```

pass 是空语句，其作用是保护程序结构的完整性。pass 不做任何事情，一般用作占位语句。

示例代码如下：

```
for a in range(5):
    if a == 3:
        pass
    print("外层循环: ", a)
```

输出结果：

```
外层循环: 0
外层循环: 1
外层循环: 2
外层循环: 3
外层循环: 4
```

3.3.3 Number 数据类型

关于 Number 数据类型的讲解视频可扫描二维码观看。

Python 中的 Number 数据类型用于存储数值。

数据类型是不允许被改变的，这就意味着如果改变 Number 数据类型的值，将需要重新分配内存空间。

以下示例表示在变量赋值时 Number 对象将被创建。

```
var1 = 10
var2 = 20
```

1. del 语句

del 语句用于删除 Number 对象引用。示例代码如下：

```
var1 = 10
var2 = 20
var3 = 30

print(var1, var2, var3)
del var1
del var2, var3
print(var1, var2, var3)
```

输出结果：

```
10 20 30
Traceback (most recent call last):
    File "D:/Users/Python/PythonSpark/Test.py", line 8, in <module>
    print(var1, var2, var3)
NameError: name 'var1' is not defined
```

2. math 模块

Python 中数学运算常用的函数基本都在 math 模块中，math 模块为浮点数提供了许多数学运算函数。因此，在使用 math 模块前必须先导入 math 模块。

示例代码如下：

```
import math

print(max(1, 2, 30, 4, 5), "找出一组数据中的最大值")
print(min(1, 2, 30, 4, 5), "找出一组数据中的最小值")

print(math.floor(4.9), "截取整数值")
print(math.ceil(4.9), "向上取整")
print(round(3.1456546, 2), "四舍五入,2 代表保留两位小数,默认取整")

print(abs(-10), "返回数字的绝对值")
print(math.fabs(-10), "返回数字的绝对值")

print(math.sqrt(9), "返回指定数字的平方根")
print(math.modf(3.14), "分离指定值的小数与整数")
```

输出结果：

```
30 找出一组数据中的最大值
1 找出一组数据中的最小值
4 截取整数值
5 向上取整
3.15 四舍五入,2 代表保留两位小数,默认取整
10 返回数字的绝对值
10.0 返回数字的绝对值
3.0 返回指定数字的平方根
(0.14000000000000012, 3.0) 分离指定值的小数与整数
```

3. random 模块

随机数可以用于数学、游戏、安全等领域中,也经常被嵌入算法中,用以提高算法效率,并提高程序的安全性。

示例代码如下:

```
import random

# 在[0, 9)范围内随机生成一个整数
print(random.choice(range(0, 10)))

# 在[100, 200)范围内随机生成一个整数,步长为 10
print(random.randrange(100, 200, 10))

# 在[0, 1)范围内随机生成一个浮点数
print(random.random())

# 在[10, 100)范围内随机生成一个浮点数
print(random.uniform(10, 100))

# 将列表中的所有元素随机排序
a = [1, 7, 4, 5, 6, 7]
print("随机排序前:", a)
random.shuffle(a)
print("随机排序后:", a)
```

输出结果(每次结果都随机):

```
1
130
0.3317805641157606
66.44410419643317
随机排序前:[1, 7, 4, 5, 6, 7]
随机排序后:[6, 1, 4, 7, 5, 7]
```

3.3.4 字符串

关于字符串的讲解视频可扫描二维码观看。

1. 定义访问字符串

字符串是 Python 中最常用的数据类型。创建字符串十分简单，只需为变量分配一个值即可，可以使用单引号或双引号来创建字符串。在 Python 中不支持单字符类型，单字符在 Python 中是作为一个字符串使用的。示例代码如下：

```
a = '学到教育'
b = "www.baidu.com"

# 获取指定字符串的第一个字符
print(a[0])

# 获取指定字符串的最后一个字符
print(a[len(a): 1])

# 获取[4, 10)范围内的字符
print(b[4:10])

# 拼接字符串。Python 中只允许同类型拼接
print(a + b)
print(a, b, sep = "")
```

输出结果：

```
学
育
baidu
学到教育 www.baidu.com
学到教育 www.baidu.com
```

2. 转义符

需要使用特殊字符时，可以在字符串中用转义符(\)进行特殊字符转换，这样 Python 就会将特殊字符当成普通字符使用。

示例代码如下：

```
print("转义反斜杠\\")
print("转义单引号\'")
print("转义双引号\"")
print("转义制表符\\t")
print("转义换行符\\n")
```

输出结果：

```
转义反斜杠\
转义单引号'
转义双引号"
转义制表符\t
转义换行符\n
```

3. 字符串内建函数

以下是 Python 字符串常见内建函数，可以使用 help(str)查看 str 函数库中的函数。

1) find()函数

find()函数用于查询指定字符的索引，包含返回开始的索引值，可指定范围查询。

示例代码如下：

```
d = "www.baidu.com"
# 返回指定字符的索引，如果找到多个字符，则返回第一个字符的索引
print(d.find("w"))
# 指定范围查询
print(d.find("w", 1, len(d): 1))
```

输出结果：

```
0
1
```

2) join()函数

join()函数用于将列中的元素以指定的字符连接生成一个新的字符串。

示例代码如下：

```
a = "-"
b = ("张三", "20", "成都")
print(a.join(b))
```

输出结果：

```
张三-20-成都
```

3) split()函数

split()函数指定分隔符对字符串进行切片，返回集合。

示例代码如下：

```
a = "张三 李四 王五 赵六 孙七 周八"
# 对字符串按空格分隔返回数据集
print(a.split(""))
# 对字符串按空格限次数分隔，返回数据集
print(a.split("", 2))
```

输出结果：

```
['张三', '李四', '王五', '赵六', '孙七', '周八']
['张三', '李四', '王五 赵六 孙七 周八']
```

4）其他常用函数

示例代码如下：

```
a = " Xue Dao "
# 转换为全小写
print(a.lower())
# 转换为全大写
print(a.upper())
# 获取字符串长度
print(len(a))
# 去除左边空格
print(a.lstrip())
# 去除右边空格
print(a.rstrip())
# 替换所有空格为 -
print(a.replace("", " - "))
# 替换 1 次空格为 -
print(a.replace("", " - ", 1))
```

输出结果：

```
 xue dao
 XUE DAO
9
Xue Dao
Xue Dao
 - Xue - Dao -
 - Xue Dao
```

3.3.5 列表

关于列表的讲解视频可扫描二维码观看。

1. 定义列表

序列是 Python 中最基本的数据结构，序列中的每个元素都分配一个索引，第一个索引是 0，第二个索引是 1，以此类推。列表的数据项不需要具有相同的类型，创建一个列表，只要把用逗号分隔的不同的数据项使用方括号括起来即可。

示例代码如下：

```
# 定义字符串列表
a = ["Python", "Java", "C#", "C 语句"]

# 定义 Number 类型列表
b = [1, 2, 3, 4, 5, 6]

# 定义混合类型列表
c = ["张三", 20, "成都"]
print(a, b, c, sep = "\n")

# list 可以定义一个空列表,也可以将其他可迭代的值转换为列表
d = list()
print("空列表", d)
```

输出结果：

```
['Python', 'Java', 'C#', 'C 语句']
[1, 2, 3, 4, 5, 6]
['张三', 20, '成都']
空列表 []
```

2. 访问列表中的值

使用下标索引可以访问列表中的元素值,也可以使用取值范围访问指定范围内的元素值。

示例代码如下：

```
# 定义字符串列表
a = ["Python", "Java", "C#", "C 语句"]

# 获取下标为 1 的元素
print(a[1])

# 获取下标为 2 到下标为 4 范围内的元素
print(a[2:4])

# 利用 for 循环遍历所有元素
for b in a:
    print("for 循环遍历结果: ", b)
```

输出结果：

```
Java
['C#', 'C 语句']
for 循环遍历结果: Python
for 循环遍历结果: Java
for 循环遍历结果: C#
for 循环遍历结果: C 语句
```

3. 更新、删除列表

列表是可变数据类型，可以通过列表对列表数据进行修改或更新。可以使用 append() 方法将元素添加到列表项中，也可以使用 del 语句删除列表中的元素。

示例代码如下：

```
# 定义字符串列表
a = ["Python", "Java", "C#", "C 语句"]

# 在 a 列表尾部追加元素
a.append("PHP")
print(a)

# 更新下标为 1 的元素值
a[1] = "Java 语言"
print(a)

# 删除指定下标中的元素
del a[2]
print(a)
```

输出结果：

```
['Python', 'Java', 'C#', 'C 语句', 'PHP']
['Python', 'Java 语言', 'C#', 'C 语句', 'PHP']
['Python', 'Java 语言', 'C 语句', 'PHP']
```

4. 列表操作符

列表中＋和＊操作符与字符串操作符相似。＋用于组合列表，＊用于重复列表。

示例代码如下：

```
# 定义字符串列表
a = ["Python", "Java", "C#", "C 语句"]
b = ["Scala", "PHP"]

# 获取 a 列表中元素的个数
print(len(a))

# 组合两个列表，生成新的列表
print(a + b)

# 重复列表
print(a * 3)

# 判断元素是否在集合中
print("Java" in a)
```

输出结果：

```
4
['Python', 'Java', 'C#', 'C 语句', 'Scala', 'PHP']
['Python', 'Java', 'C#', 'C 语句', 'Python', 'Java', 'C#', 'C 语句', 'Python', 'Java', 'C#',
'C 语句']
True
```

5. 嵌套列表

嵌套列表即在列表中创建其他列表，类似 Java 中的二维数组。
示例代码如下：

```
a = [["张三", 20, "成都"],
     ["李四", 22, "北京"],
     ["王五", 18, "上海"]]

# 打印列表中所有数据
print(a)

# 打印索引为 0 的列表的所有数据
print(a[0])

# 打印具体元素
print(a[0][0])
```

输出结果：

```
[['张三', 20, '成都'], ['李四', 22, '北京'], ['王五', 18, '上海']]
['张三', 20, '成都']
张三
```

6. 列表常用函数

列表常用函数示例如下：

```
a = [2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2]
b = [100, 200]
c = (300, 600)

# 统计 5 在列表中出现的次数
print(a.count(5))

# 将 b 列表追加到 a 列表中
a.extend(b)
print(a)
```

```
# 将 c 元组追加到 a 列表中
a.extend(c)
print(a)

# 在 a 列表中找到第一个元素为 5 的下标值
print(a.index(5))

# 将 999 元素值插入下标为 0 的位置
a.insert(0, 999)
print(a)

# 移除最后一个元素
a.pop()
print(a)

# 移除指定位置元素
a.pop(0)
print(a)

# 删除第一个匹配的元素
a.remove(5)
print(a)

# 反转列表中的所有元素
a.reverse()
print(a)

# 从 a 列表中复制一个新列表
d = a.copy()
print(d)

# 清空 a 列表中的所有元素
a.clear()
print(a)
```

输出结果：

```
3
[2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2, 100, 200]
[2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2, 100, 200, 300, 600]
3
[999, 2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2, 100, 200, 300, 600]
[999, 2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2, 100, 200, 300]
[2, 3, 4, 5, 6, 5, 4, 4, 5, 2, 2, 100, 200, 300]
[2, 3, 4, 6, 5, 4, 4, 5, 2, 2, 100, 200, 300]
[300, 200, 100, 2, 2, 5, 4, 4, 5, 6, 4, 3, 2]
[300, 200, 100, 2, 2, 5, 4, 4, 5, 6, 4, 3, 2]
[]
```

3.3.6 元组

关于元组的讲解视频可扫描二维码观看。

Python 的元组与列表基本类似，但也有不同，不同之处在于元组的元素不能修改。元组使用小括号，而列表使用方括号。元组的创建十分简单，只需在括号中添加元素，并用逗号隔开即可。

1. 定义元组

示例代码如下：

```
# 定义同一类型的元素
a = ("张三", "女", "20")

# 定义不同类型的元素
b = ("张三", "女", 20)

# 元组可以不需要括号
c = "张三", "女", 20

print(a, b, c)
```

输出结果：

```
('张三','女','20') ('张三','女',20) ('张三','女',20)
```

2. 访问元组

可以使用下标索引来访问元组中的值。

示例代码如下：

```
a = ("张三", "女", "20", "成都", "软件开发工程师")

# 获取下标为 0 的值
print(a[0])

# 获取[1, 3)范围内的元组
print(a[1:3])

# 遍历元组数据
for b in a:
    print(b, end = "\t")
```

输出结果：

```
张三
('女', '20')
张三  女  20  成都  软件开发工程师
```

3. 修改元组

元组为不可变类型，不允许修改。但是，可以使用＋运算符连接两个元组，从而生成另一个新的元组。

示例代码如下：

```
a = ("张三", "女", "20")
b = ("成都", "软件开发工程师")

c = a + b
print(c)
```

输出结果：

```
('张三','女','20','成都','软件开发工程师')
```

4. 删除元组

由于元组是不可变类型，因此元组中的元素值是不允许被删除的，但可以使用 del 语句来删除整个元组。

示例代码如下：

```
a = ("张三", "女", "20")
b = ("成都", "软件开发工程师")

del a
c = a + b
print(c)
```

输出结果：

```
Traceback (most recent call last):
    File "D:/Users/Python/PythonSpark/Test.py", line 5, in <module>
    c = a + b
NameError: name 'a' is not defined
```

5. 元组常用内置函数

元组常用内置函数示例如下：

```
a = ("张三", "女", "20", "成都", "软件开发工程师")
b = [1, 2, 4]

# 获取 a 元组中的元素个数
print(len(a))

# 获取元组中的最大值
print(max(a))

# 获取元组中的最小值
print(min(a))

# 将列表转换为元组
print(tuple(b))
```

输出结果：

```
5
软件开发工程师
20
(1, 2, 4)
```

3.3.7 字典

关于字典的讲解视频可扫描二维码观看。

字典是另一种可变容器，可存储任意类型的对象。字典的每个元素都是键值对，每个元素之间都用逗号分隔，其中键必须是唯一的，但值则不必须唯一。值可以取任何数据类型，但键必须是不可变的，如字符串、数字或元组。

1. 访问字典

把相应的键放入方括号中，可以获取相应的值。

示例代码如下：

```
a = {"name": "张三", "age": 20, "地址": "成都"}
print(a["name"], a["地址"])

b = {1: "成都", 2: "上海", 3: "北京"}
print(b[1], b[3])

c = {(1, 2, 3): "成都", (4, 5, 6): "北京"}
print(c[(1, 2, 3)])
```

输出结果：

```
张三 成都
成都 北京
成都
```

2. 更新字典

更新字典包括对字典中元素的增、删、改操作。示例代码如下：

```
a = {"name": "张三", "age": 20, "地址": "成都"}

# 向字典中添加新值
a["tel"] = "15008208000"
print(a)

# 修改字典中某个元素的值
a["age"] = 30
print(a)

# 删除指定 key 的值
del a["地址"]
print(a)
```

输出结果：

```
{'name': '张三', 'age': 20, '地址': '成都', 'tel': '15008208000'}
{'name': '张三', 'age': 30, '地址': '成都', 'tel': '15008208000'}
{'name': '张三', 'age': 30, 'tel': '15008208000'}
```

3. 字典常用内置函数

以下为字典常用内置函数的示例：

```
a = {"name": "张三", "age": 20, "地址": "成都"}

# 获取字典中元素的个数
print(len(a))

# 复制字典
b = a.copy()
print(b)

# Python 字典中 fromkeys()方法用于创建一个新字典,以序列 seq 中元素作为字典的键,value
# 为字典所有键对应的初始值
c = ("name", "sex", "tel")
d = "初始值"
f = a.fromkeys(c, d)
print(f)
```

```
# get()方法用于返回指定键的值,如果值不在字典中则返回默认值
print(a.get("name"))
print(a.get("name2", "默认值"))

# items()方法用于返回可遍历的元素数组
print(a.items())

# keys()方法用于返回 key 值的可迭代对象,可以使用 list()转换为列表
a2 = a.keys()
print(list(a2))

# values()方法用于返回 values 值的可迭代对象
a3 = a.values()
print(list(a3))

# update()方法用于将一个字典追加到另一个字典中
str2 = {"a":"A", "b":"b"}
a.update(str2)
print(a)

# pop()方法用于删除字典中给定 key 所对应的值
a.pop("name")
print(a)

# 清除字典中所有元素
a.clear()
print(a)
```

输出结果：

```
3
{'name': '张三', 'age': 20, '地址': '成都'}
{'name': '初始值', 'sex': '初始值', 'tel': '初始值'}
张三
默认值
dict_items([('name', '张三'), ('age', 20), ('地址', '成都')])
['name', 'age', '地址']
['张三', 20, '成都']
{'name': '张三', 'age': 20, '地址': '成都', 'a': 'A', 'b': 'b'}
{'age': 20, '地址': '成都', 'a': 'A', 'b': 'b'}
{}
```

3.3.8 集合

关于集合的讲解视频可扫描二维码观看。

集合是一个无序、不重复的元素序列,可以使用花括号创建集合,集合元素只能为不可变类型。

1. 更新集合

可以通过对集合进行增、删、改操作来更新集合。
示例代码如下：

```
a = {"张三", "李四", "张三", "王五"}

# 集合内元素不能重复
print(a)

# 往集合中添加元素
a.add("赵六")
print(a)

# 往指定集合中添加列表
a.update({"孙七", "周八"})
print(a)

# 往指定集合中添加元组
a.update(("孙一一", "周杰"))
print(a)

# 往指定集合中添加字典
a.update({"name": "王伦"})
print(a)

# 不能添加可变类型,如 Number 类型
a.update(1)
print(a)
```

输出结果：

```
TypeError: 'int' object is not iterable
{'李四', '张三', '王五'}
{'李四', '张三', '赵六', '王五'}
{'张三', '赵六', '王五', '孙七', '李四', '周八'}
{'张三', '赵六', '孙一一', '王五', '孙七', '周杰', '李四', '周八'}
{'张三', '王五', '周八', '周杰', '李四', '赵六', '孙一一', '孙七', 'name'}

Traceback (most recent call last):
    File "D:/Users/Python/PythonSpark/Test.py", line 23, in <module>
    a.update(1)
Process finished with exit code 1
```

2. 移除元素

示例代码如下：

```
a = {"张三", "李四", "张三", "王五"}

# remove()用于从集合中移除指定的元素,如果元素不存在,则会发生错误
a.remove("张三")
print(a)

# discard()用于从指定集合中移除指定的元素,如果元素不存在,则不会发生错误
a.discard("张三三")
print(a)

# pop()用于设置随机删除集合中的一个元素,集合的 pop()方法会对集合进行无序的排序,然后
# 将这个无序排序的第一个元素删除
a.pop()
print(a)
```

输出结果：

```
{'王五', '李四'}
{'王五', '李四'}
{'李四'}
```

3. 集合常用内置函数

以下为集合常用内置函数的示例：

```
a = {"张三", "李四", "张三", "王五"}
b = {"李四", "王一一"}

# union()用于返回两个集合的并集
print(a.union(b))

# intersection()用于返回两个集合的交集
print(a.intersection(b))

# copy()用于复制一个集合
c = a.copy()
print(c)

# clear()用于清空指定集合
a.clear()
print(a)
```

输出结果：

```
{'王一一', '王五', '张三', '李四'}
{'李四'}
{'李四', '王五', '张三'}
set()
```

3.3.9 函数

关于函数的讲解视频可扫描二维码观看。

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。函数能提高模块的应用和代码的重复利用率。Python 提供了许多内建函数，如 print()。用户也可以自己创建函数，这样的函数被称为用户自定义函数。

用户定义一个自己想要功能的函数时，需要遵循下列规则。

(1) 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号。

(2) 任何传入的参数和自变量必须放在圆括号中间，圆括号之间可以定义参数。

(3) 函数内容以冒号起始，并且缩进。

(4) return[表达式]结束函数，选择性返回一个值给调用方。不带表达式的 return 语句相当于返回 None。

(5) 方法与方法要间隔 2 行。

1. 函数的语法

一个函数由名字、参数和函数体 3 个部分组成。

Python 定义函数时使用 def 关键字，格式如下：

```
def 函数(参数列表):
    函数体
```

PyCharm 编译工具要求 Python 中的方法与方法的间隔为 2 行。示例代码如下：

```
def method1():
    print("无参函数")


def method2(a):
    print("有一个参数函数,参数值为: ", a)


def method3(a, b, c):
    print("多个参数函数,参数值为: ", a, b, c, end = "\t")


# 调用函数
method1()
method2("Python")
method3("Python", 88, "基础语言")
```

输出结果：

```
无参函数
有一个参数函数,参数值为: Python
多个参数函数,参数值为: Python 88 基础语言
```

2. 函数调用

定义一个函数时需要给函数一个名称，可指定函数中的接收参数和代码块结构。这个函数的基本结构完成以后，用户可以通过另一个函数调用执行，也可以直接通过Python的命令提示符执行。

示例代码如下：

```
def method1():
    print("无参函数")
    # 在函数中调用另一个函数
    method2("测试")


def method2(a):
    print("有一个参数函数,参数值为: ", a)


# 调用函数
method1()
```

输出结果：

```
无参函数
有一个参数函数,参数值为: 测试
```

3. 变量

函数中的变量不能跨区域使用。如图3.29所示，在函数外调用函数内的变量时，将出现该变量的作用域错误的问题，这时a变量在方法内，称为局部变量。局部变量的作用域在该方法内。

变量定义在方法外的变量称为全局变量。全局变量在整个.py文件中均有效，如图3.30所示。

```
def method1():
    a = 10
    b = 20
    return a * b


print("被调用的函数返回值: ", method1())
print("调用method1()中的a值: ", a)
```

图3.29 局部变量示例

```
def method1():
    b = 20
    return a * b


a = 10
print("被调用的函数返回值: ", method1())
print("调用method1()函数中的a值: ", a)
```

图3.30 全局变量示例

4. 设置默认值

函数除了定义标准的参数外还有一些特殊的传参方式，例如设置参数默认值、设置不

定长参数。

可以通过设置参数默认值从而设置指定传参的默认值，没有传入该参数时，将使用它的默认值。

示例代码如下：

```
def method(name, age = 20):
    print(name, age)


method("张三", 33)
method("李四")
```

输出结果：

```
张三 33
李四 20
```

5. 设置不定长参数

用户如果需要使用一个函数处理更多的参数时，可将其定义为不定长参数。不定长参数允许传入多个参数，这些参数将以元组形式存在。

示例代码如下：

```
def method(name, age = 20, * score):
    print(name, age, score)


method("张三", 33)
method("李四")
method("张三", 30, "A", 98.5, 10)
```

输出结果：

```
张三 33 ()
李四 20 ()
张三 30 ('A', 98.5, 10)
```

6. return 语句

return 语句用于结束方法或返回值。

示例代码如下：

```
def method():
    for val in range(5):
        if val == 3:
```

```
            print("结束整个方法")
            return
        else:
            print("打印值: ", val)


method()


def method2(a, b):
    return a + b


print("返回结果值: ", method2(10, 20))
```

输出结果：

```
打印值: 0
打印值: 1
打印值: 2
结束整个方法
返回结果值: 30
```

3.3.10 模块

关于模块的讲解视频可扫描二维码观看。

Python 的流行主要得益于其有众多功能强大的库，Python 自带的标准库(standard library)可以满足大多数用户的基本需求。除了函数库以外，模块(module)和包(package)也常被提及。

Python 可以把定义的所有方法和变量存放在文件中，为一些脚本或者交互式的解释器实例所使用，这个文件被称为模块。

模块包含所有被定义的函数和变量的文件，其扩展名是.py。

模块可以被其他程序引入，以使用该模块中的函数等功能。引入模块有以下两种方法：

(1) import 包.模块。

(2) form 包 import 模块。

1. 使用模块

以下例子使用 Python 标准库中的 sys 模块，结果如图 3.31 所示。

```
test.py
import sys

print("输入的结果: ")
for i in sys.argv:
    print(i)
```

图 3.31　sys 模块

上述代码中 import sys 引入 Python 标准库中的 sys 模块。sys.argv 用于查询当前文件在本地的位置。

2. import 语句

要使用 Python 源文件，只需要在另一个源文件中执行 import 语句即可。语法如下所示。

```
import module1[,module2[,… moduleN]
```

其中，import 关键字表示导入指定模块文件，当解释器遇到 import 语句后，解释器会首先执行 import 后面指定的模块文件，并为该模块文件分配一个命名空间用于进一步使用。

3. 搜索路径

搜索路径是在 Python 编译或安装时确定的，搜索路径被存储在 sys 模块中的 path 变量中。

导入模块时，搜索引擎将在 sys.path 路径中进行搜索。如果在指定路径中搜索到要导入的包便将包导入，如果搜索不到则将会报错，如图 3.32 所示。

```
test.py
import sys

listVal = sys.path
for val in listVal:
    print("搜索路径：", val)
```

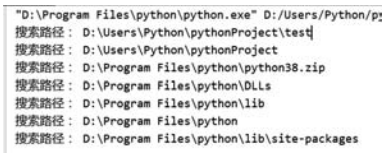

图 3.32　搜索路径

4. form…import 语句

from…import 语句除了导入指定模块中的单个函数外，还可以导入多个函数或全部函数。

导入多个函数的示例代码如下：

```
test.py
from db.student import study, sleep

print(study("李四"))
print(sleep("张三"))
```

导入全部函数的示例代码如下：

```
test.py
from db.student import *

print(student("李四"))
print(student("张三"))
```

5. __name__属性

__name__是一个变量，前后加两个下画线是因为这是系统定义的名字（普通变量不要使用该方式命名变量）。

__name__系统变量分以下两种情况。

第一种：假如当前模块是主模块，那么此模块名字就是__name__，可通过 if 判断来执行 if 中的语句块，类似于 Java 中的 main()方法。

第二种：假如如此模块是被 import 导入的模块，那么此模块名字为文件名字，通过 if 判断就会跳过__main__后面的内容。

通过上面的方式，Python 可以分清楚哪些是主函数，进行函数执行，并且可以调用其他模块的各个函数等。

一个模块被另一个程序第一次引入时，其主程序将运行。如果想在模块被引入时模块中的某一程序块不执行，可以用__name__属性使该程序块仅在该模块自身运行时执行。

```
test.py
from db.student import *

print(study("李四"))

student.py
def study(name):
    val = name + "在学习"
    return val


def sleep(name):
    return name + "在睡觉"
```

```
if __name__ == "__main__":
    print(sleep("王一一"))
```

加上__name__属性后，在test.py运行时，sleep("王一一")将不会运行，只在student.py中，它才会运行。

每个模块都有一个__name__属性，当其值是__main__时，表明该模块自身在运行，否则被引入。

6. 包

包是一种管理Python模块命名空间的形式，采用".模块名称"的形式。

在导入一个包时，Python会根据sys.path中的目录来寻找这个包中所包含的子目录。

目录只包含一个__init__.py的文件时才会被认作是一个包，该文件为一个空文件即可，不需要进一步操作。

用户可以每次只导入一个其中的特定模块。示例代码如下：

```
import db.student
print(db.student.study("李四"))


import db.student as st
print(st.study("李四"))
```

还有一种导入子模块的方法。示例代码如下：

```
from db.student import study

print(study("李四"))


from db import student

print(student.study("李四"))
```

3.3.11 类和对象

关于类和对象的讲解视频可扫描二维码观看。

Python设计之初就是一种面向对象的语言。正因如此，在Python中创建一个类和对象十分容易。

在计算机的世界中，对象是一个十分重要的概念。对象是程序组织代码的方法，它可以将复杂的想法拆分开，使其变得容易被理解。

在Python中，对象是由类定义的，可以把类当成一种把对象分组归类的方法。

Python 中的类提供了面向对象编程的所有基本功能：类的继承、封装和多态。

案例：通过一个人要开车从 A 点到 B 点比较面向对象与面向过程的区别。

面向过程方式如图 3.33 所示。

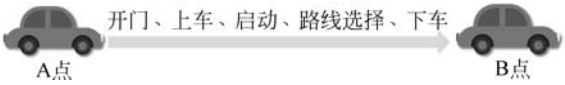

图 3.33　面向过程方式

面向对象方式如图 3.34 所示。

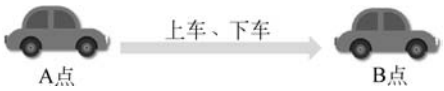

图 3.34　面向对象方式

1. 类与对象

在 Java 的世界中，万物皆对象，对象是类的一个实例，它有状态和行为。

类是一个模板，它描述一类对象的行为和状态，对象是具体的事物；类是对于对象的抽象；类可以看成一类对象的模板，对象可以看成该类的一个具体实例；类是用于描述同一类型对象的一个抽象概念，类则是定义了这一类对象所应具有的共同属性和方法。

例如，一条狗是一个对象。它的属性为颜色、名字、品种，行为为叫、吃、跑、摇尾巴。

类与对象的关系示例如图 3.35 所示。

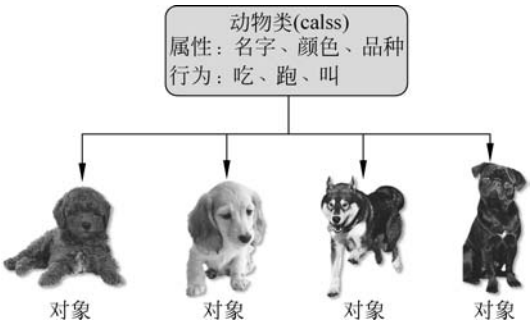

图 3.35　类与对象的关系示例图

2. 面向对象的特性

对象是模拟真实世界，对数据和程序进行封装。

```
对象 = 属性 + 方法
```

需要用类来创建一个对象，就像要用图纸来造房子一样。

面向对象编程的3大特性是封装、继承、多态。

(1) 封装：把自己的属性和行为封装起来，并提供公共接口给调用者使用。

(2) 继承：子类对父类的继承，子类拥有父类的属性和行为。

(3) 多态：父类在子类中的多种形态。

3. 类的定义

类是一个独立存储属性和方法的空间，可使用运算符“.”来调用类的属性和方法。示例代码如下：

```
class Person:
    name = "张三"
    sex = "女"
    age = 20

    def eat(self):
        print(self.name, "在吃饭")


# 调用类
a = Person()
print(a.name)
a.eta()
```

输出结果：

```
张三
张三  在吃饭
```

3.3.12 封装

关于封装的讲解视频可扫描二维码观看。

面向对象编程的一个重要特征就是数据封装。封装是把内部细节私有化，通过公共接口把入口和出口暴露出去来实现封装效果。

示例代码如下：

```
class Person:
    __name = "张三"
    __age = 18

    # 封装细节的方法
    def __privateMethod(self):
        return [self.__name, self.__age]

    # 暴露给调用者
    def method(self):
```

```
        return self.__privateMethod()


p = Person()
print(p.method())
```

输出结果：

```
['张三',18]
```

1. 私有属性

Python 默认的方法和成员变量都是公开的，Python 的私有属性和方法并没有像其他语言一样用 public、private 等关键词来修饰。

不加任何下画线表示公开变量，它的作用域在整个项目中均有效，例如，x ="变量"。

双下画线(__)表示私有变量，只有内部可以访问，外部不可以访问，例如，__xx ="私有变量"。

示例代码如下：

```
class Person:
    name = "张三"                    # 公开属性
    __age = 18                       # 私有属性


p = Person()
print("公开属性类外可以调用: ", p.name)
print("私有属性类外不可以调用: ", p.__age)
```

输出结果：

```
公开属性类外可以调用: 张三
Traceback (most recent call last):
    File "D:/Users/Python/PythonSpark/Test.py", line 8, in <module>
    print("私有属性类外不可以调用: ", p.__age)
AttributeError: 'Person' object has no attribute '__age'
```

注意，当使用 PyCharm 进行 Python 开发时，如果需要在类中定义方法，若该方法不涉及对属性的操作，那么 PyCharm 会提示 Method xxx may be 'static'。因为 PyCharm 会认为该方法是一个静态方法，而不是类方法，所以提示在该方法前添加@staticmethod 装饰器进行装饰。

2. self

self 代表类的实例。类的方法与普通的函数只有一个特殊的区别，它们必须有一个

额外的第一个参数名称，按照惯例它的名称是 self。

示例代码如下：

```
class Test:
    def prt(self):
        print(self)


t = Test()
t.prt()

t2 = Test()
t2.prt()
```

输入结果：

```
<__main__.Test object at 0x0000000001DD9400>
<__main__.Test object at 0x0000000001DE4550>
```

从执行结果可以很明显地看出，self 代表的是类的实例，代表当前对象的地址。

3.3.13 构造函数

关于构造函数的讲解视频可扫描二维码观看。

构造函数也称为构造器，是创建对象时会被自动调用的。系统默认将提供一个无参的构造函数。

因为类可以起到模板的作用，所以在创建实例时，可以把一些自认为必须绑定的属性强制写进去。通过定义一个特殊的__init__()方法，在创建实例时，对属性进行赋值，称为初始化。

构造函数有以下特点：

(1) 在一个类中，只能定义一个结构函数。

(2) 重新定义结构函数时可以使用__init__(self,arg1,arg2,…)函数。

(3) arg1,arg2,…是传入的形参，用于初始化成员变量。

(4) 在构造函数中定义的变量为成员变量(全局变量)，可以在类中任意一处使用。例如，self.name = "小猫咪"。

(5) 在对象被创建时自动调用构造函数。

1. 默认构造函数

示例代码如下：

```
class Person:
    def __init__(self):
        print("无参构造函数是默认方式")
```

```
# 创建对象时调用无参构造函数
Person()
```

输出结果：

```
无参构造函数是默认方式
```

2. 有参构造函数

示例代码如下：

```
class Person:
    def __init__(self, name):
        print(f"姓名：{name}")


Person("张三")
```

输出结果：

```
姓名：张三
```

3. 使用结构函数中的全局变量

示例代码如下：

```
class Person:
    def __init__(self, name):
        self.name = name

    def sleep(self):
        return f"{self.name}在睡觉"


p = Person("张三")
print(p.sleep())
```

输出结果：

```
张三在睡觉
```

4. 传入不定长参数

示例代码如下：

```
class Person:
    def __init__(self, *val):
        self.val = val

    def sleep(self):
        return self.val


p = Person("张三", "李四", "王五")
print(p.sleep())
```

输出结果：

```
('张三', '李四', '王五')
```

3.3.14 继承

关于继承的讲解视频可扫描二维码观看。

继承指的是类与类之间的关系，用来解决代码重用问题。继承有以下特点。

(1) 定义继承可直接在类名的括号中进行，只需加上所要继承的类即可。例如：

```
class Student(Person):
```

(2) 在 Python 中被继承的类称为父类，继承的类称为子类。

(3) 在 Python 中允许多继承。

(4) 在继承中，子类拥有父类中的成员变量和成员函数。

(5) 如果子类中定义了与父类中相同的函数名，这种方式称为重写。

(6) 如果子类继承抽象类，抽象类中有抽象函数，则子类必须重写。

继承的示例代码如下：

```
class Person:
    name = ""

    def eat(self):
        return self.name + "在吃东西"


class Student(Person):
    def study(self):
        return self.name + "在学习"


s = Student()
s.name = "张三"
print(s.eat())
print(s.study())
```

输出结果：

```
张三在吃东西
张三在学习
```

1. 导入模块

import 关键字可以在不同的包中导入指定模块，实现继承关系。项目结构如图 3.36 所示。

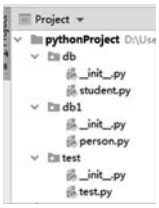

图 3.36 项目结构

示例如下：

```
person.py
class Person:
    def eat(self):
        return "吃东西"


student.py
import db1.person as p

class Student(p.Person):
    __name = "王一"

def sleep(self):
    return self.__name


test.py
from db import student

a = student.Student()
# 公共方法或变量可以调用
print(a.sleep())
# 调用父类中的方法
print(a.eat())
```

2. 多继承

Python 与 Java 不同，Python 支持多继承，即一个类继承多个类的属性与行为。

示例代码如下：

```
class Person:
    name = ""

    def eat(self):
        return self.name + "在吃东西"


class Animal:
    def sleep(self):
        return "在睡觉"


class Student(Person, Animal):
    def study(self):
        return self.name + "在学习"


s = Student()
s.name = "张三"
print(s.eat())
print(s.study())
print(s.name, s.sleep())
```

输出结果：

```
张三在吃东西
张三在学习
张三 在睡觉
```

3. 方法重写

如果用户的父类方法功能不能满足用户的需求，用户可以在子类中重写父类中的方法，称为方法重写。

示例代码如下：

```
class Person:
    name = ""

    def eat(self):
        return self.name + "在吃东西"
```

```
class Student(Person):
    def eat(self):
        return self.name + "在学校食堂里吃东西"


s = Student()
s.name = "张三"
print(s.eat())
```

输出结果：

```
张三在学校食堂里吃东西
```

4. super()函数

super()函数是用于调用父类的一个方法。
示例代码如下：

```
class Person:
    name = ""

    def eat(self):
        return self.name + "在吃东西"

class Student(Person):
    def eat(self):
        return self.name + "在学校食堂里吃东西"

    def test(self):
        return super().eat()


s = Student()
s.name = "张三"
print(s.test())
```

输出结果：

```
张三在吃东西
```

5. 重写构造函数

子类可以没有构造函数，此时表示子类同父类的构造函数一致。

示例代码如下：

```
class Person:
    def __init__(self):
        print("父类中的构造函数")


class Student(Person):
    pass


Student()
```

输出结果：

```
父类中的构造函数
```

子类可以重写构造函数。示例代码如下：

```
class Person:
    def __init__(self):
        print("父类中的构造函数")


class Student(Person):
    def __init__(self, name):
        super().__init__()
        print("重写父类中的构造函数", name)


Student("张三")
```

输出结果：

```
父类中的构造函数
重写父类中的构造函数 张三
```

3.3.15 异常

关于异常的讲解视频可扫描二维码观看。

Python 有两种错误很容易辨认：语法错误和异常。

语法错误或者称为解析错误，是初学者经常遇到的错误。语法错误是语法上的错误，通常表现为少冒号、关键字输错，这类错误在解析前会由解析器进行自动检查。

异常就是一个事件，该事件会在程序执行的过程中发生，并影响程序的正常执行。

例如，下例会出现多种不同的异常。

```
# 下标越界异常
a = (1, 2, 5)
try:
    print(a[5])
except IndexError as e:
    print("下标越界", e)

# 值转换错误异常
s = "3.14"
try:
    i = int(s)
except ValueError as e:
    print("值转换错误", e)


# key 值错误异常
a = {"name": "张三", "age": 20}
try:
    print(a["name1"])
except KeyError as e:
    print("key 值错误异常", e)


# 分母为 0 异常
try:
    a = 10
    b = 0
    c = a / b
except ZeroDivisionError as e:
    print("分母不能为零: ", e)
```

输出结果：

```
下标越界 tuple index out of range
值转换错误 invalid literal for int() with base 10: '3.14'
key 值错误异常 'name1'
分母不能为零: division by zero
```

1. 异常处理

异常捕捉可以使用 try…except 语句。

语法格式如图 3.37 所示。

首先，执行 try 子句，如果没有异常发生，则忽略 except 子句，try 子句执行后结束程序的执行。如果在执行 try 子句的过程中发生异常，那么 try 子句余下的部分将被忽略，执行 except 子句中的语句。如果异常的类型和 except 之后的名称相符，那么对应的

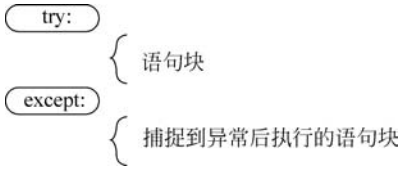

图 3.37　try…except 语法格式

except 子句将被执行。如果一个异常没有与任何 except 子句匹配，那么这个异常将会被传递到上层的 try 子句中。

2. 处理多个异常

一个 try 语句可能包含多个 except 子句，分别处理不同的特定的异常，最多只有一个分支会被执行。

处理程序将只针对对应的 try 子句中的异常进行处理，而不是其他 try 子句中的处理程序的异常。

示例代码如下：

```
while True:
    try:
        x = int(input("请输入一个数字: "))
        val = 10 / x
        print("结果: ", val)
    except ValueError as e:
        print("你输入的不是数字,错误码为: ", e)
    except ZeroDivisionError as e:
        print("分母不能为零,错误码为: ", e)
```

输出结果：

```
请输入一个数字:0
分母不能为零,错误码为: division by zero
请输入一个数字:ddf
你输入的不是数字,错误码为: invalid literal for int() with base 10: 'ddf'
请输入一个数字:0
分母不能为零,错误码为: division by zero
请输入一个数字:
```

3. else 语句

如果没有错误发生，则可以在 except 语句块后面加一个 else 语句。当没有错误发生时，会自动执行 else 语句。

示例代码如下：

```
while True:
    try:
        x = int(input("请输入一个数字: "))
        val = 10 / x
        print("结果: ", val)
    except ValueError as e:
        print("你输入的不是数字,错误码为: ", e)
    except ZeroDivisionError as e:
        print("分母不能为零,错误码为: ", e)
    else:
        print("系统正常,没有异常")
```

输出结果：

```
请输入一个数字: 8
结果: 1.25
系统正常,没有异常
请输入一个数字:
```

4. finally 语句

finally 语句是最终语句，表示无论如何都要执行。如果加上了 finally 语句，不管 except 语句是否被执行，finally 语句都将会被执行。

在 try…except…finally 中，finally 语句可以省略，也可以加上，但只能有一个 finally 语句。

示例代码如下：

```
while True:
    try:
        x = int(input("请输入一个数字: "))
        val = 10 / x
        print("结果: ", val)
    except ValueError as e:
        print("你输入的不是数字,错误码为: ", e)
    except ZeroDivisionError as e:
        print("分母不能为零,错误码为: ", e)
    else:
        print("系统正常,没有异常")
    finally:
        print("无论如何都执行")
```

输出结果：

```
请输入一个数字: 8
结果: 1.25
系统正常,没有异常
```

```
无论如何都执行
请输入一个数字：0
分母不能为零，错误码为：division by zero
无论如何都执行
请输入一个数字：
```

5. Exception 异常

Python 的错误其实也是类，所有的错误类型都继承自 Exception，所以在使用 Exception 时需要注意，它不但能够捕获该类型的错误，还能把子类也“一网打尽”。

示例代码如下：

```
while True:
    try:
        x = int(input("请输入一个数字："))
        val = 10 / x
        print("结果：", val)
    except Exception as e:
        print("Exception 异常：", e)
    except ValueError as e:
        print("你输入的不是数字，错误码为：", e)
    except ZeroDivisionError as e:
        print("分母不能为零，错误码为：", e)
    else:
        print("系统正常，没有异常")
    finally:
        print("无论如何都执行")
```

输出结果：

```
请输入一个数字：0
Exception 异常：division by zero
无论如何都执行
请输入一个数字：dd
Exception 异常：invalid literal for int() with base 10：'dd'
无论如何都执行
请输入一个数字：
```

上述例子中，Exception 异常出现在 ValueError 和 ZeroDivisionError 异常之前。由于 Exception 异常是所有异常的父类，因此，在 Exception 异常时就会被拦截。后面的 ValueError 和 ZeroDivisionError 异常不会被执行。由于 Exception 异常父类的特性，在实践中，Exception 异常一般用于所有异常之后，用来拦截没有考虑到位的异常，从而保证系统的安全性。

6. 抛出异常

抛出异常是指自己不处理异常，使用 raise 语句将异常抛出去，由调用者自己处理。Python 使用 raise 语句抛出一个指定的异常。raise 语句是唯一一个参数指定的要抛出的异常，它必须是一个异常的实例或异常类。

示例代码如下：

```
def sex():
    a = input("请输入性别: ")
    if a != "男" and a != "女":
        raise Exception("抛出异常")
    print(a)


try:
    sex()
except Exception as e:
    print("异常信息: ", e)
```

输出结果：

```
请输入性别: 公
异常信息: 抛出异常
```

7. 自定义异常

可以通过创建一个新的异常类从而拥有自己的异常，自定义异常类可以继承自 Exception 类，可以直接继承也可以间接继承。

示例代码如下：

```
class GenderException(Exception):
    pass


# 定义 sex()函数
def sex():
    a = input("请输入性别: ")
    if a != "男" and a != "女":
        raise GenderException("性别不详")
    print(a)


try:
    sex()
except GenderException as e:
    print("异常信息: ", e)
```

输出结果：

```
请输入性别：公
异常信息：性别不详
```

3.3.16 操作 MySQL

关于操作 MySQL 的讲解视频可扫描二维码观看。

1. 安装连接 MySQL 的驱动

MySQL 是最流行的关系数据库管理系统，Python 连接到 MySQL 数据库就需要安装 MySQL 官方提供的 mysql-connector 驱动。

mysql-connector 驱动可以通过 pip 命令进行安装，如图 3.38 所示。

图 3.38 安装 MySQL 驱动

2. 创建数据库连接

示例代码如下：

```
import mysql.connector as my

# 第一步：建立数据库连接
mydb = my.connect(
    host = "localhost",   # 连接地址
    user = "root",        # 用户
    password = "123456"   # 密码
)

myCursor = mydb.cursor()

# 第二步：执行数据库操作，创建数据库
myCursor.execute("create database xuedao charset = utf8")
```

执行结果如图 3.39 所示。

3. 创建数据表

创建数据表需要使用 create table 语句。在创建数据表前,需要确保数据库已经存在。以下创建一个名为 student 的数据表。

示例代码如下:

```
import mysql.connector as my

# 第一步: 如果直接连接数据库,可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",   # 连接地址
    user = "root",        # 用户
    password = "123456", # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

# 第二步: 执行数据库操作,创建数据库
myCursor.execute(
    create table student (
        name varchar(50),
        sex varchar(20),
        age int
    )
)
```

执行结果如图 3.40 所示。

图 3.39 创建数据库连接执行结果

图 3.40 创建数据表执行结果

4. 插入数据

插入数据使用 insert into 语句,其中%s 是占位符,它与第二个参数中、元组中的元素位置匹配。

示例代码如下:

```
import mysql.connector as my

# 第一步：如果直接连接数据库,可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",   # 连接地址
    user = "root",        # 用户
    password = "123456", # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()
# 第二步：插入一条数据到数据表
sql = "insert into student values( % s, % s, % s)"
dataVal = ("李四", "女", 18)
myCursor.execute(sql, dataVal)
# 第三步：提交数据到数据库
mydb.commit()
```

执行结果如图 3.41 所示。

5. 批量插入数据

批量插入数据使用 executemany()方法,该方法的第二个参数是一个元组列表,包含所要插入的数据。

示例代码如下：

```
import mysql.connector as my

# 第一步：如果直接连接数据库,可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",   # 连接地址
    user = "root",        # 用户
    password = "123456", # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

# 第二步：插入一条数据到数据库
sql = "insert into student values( % s, % s, % s)"
dataVal = [("张一", "女", 8),
           ("张二", "男", 28),
           ("张三", "女", 31),
           ("张四", "女", 22),
           ("张五", "男", 19),]

# 第三步：使用 executemany()方法执行批量插入
myCursor.executemany(sql, dataVal)
# 第四步：提交数据到数据库
mydb.commit()
```

执行结果如图 3.42 所示。

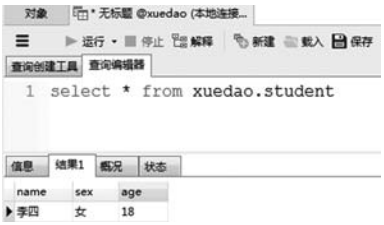

图 3.41 插入数据执行结果

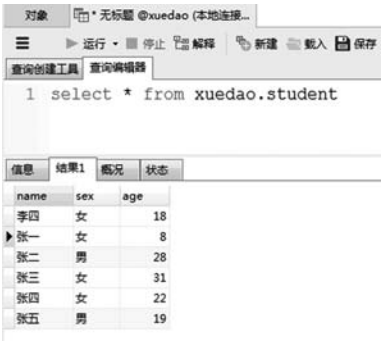

图 3.42 批量插入数据执行结果

6. 查询数据

查询数据使用 select 语句。

示例代码如下：

```
import mysql.connector as my

# 第一步：如果直接连接数据库,可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",    # 连接地址
    user = "root",         # 用户
    password = "123456",   # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

# 第二步：获取全部数据
sql = "select * from student"

# 第三步：使用 execute()执行数据查询
myCursor.execute(sql)
result = myCursor.fetchall()

# 第四步：遍历数据结果集
print("全表输出：")
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])

print()
# 转出指定字段
sql2 = "select name, sex from student"
```

```
myCursor.execute(sql2)
result2 = myCursor.fetchall()
print("指定字段输出：")
for a in result2:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"))


print()
# 读取第一条数据
sql3 = "select * from student"
myCursor.execute(sql3)
result3 = myCursor.fetchone()
print(result3[0], result3[1], result3[2])
```

输出结果：

```
全表输出：
李四 女 18
张一 女 8
张二 男 28
张三 女 31
张四 女 22
张五 男 19

指定字段输出：
李四 女
张一 女
张二 男
张三 女
张四 女
张五 男

bytearray(b'\xe6\x9d\x8e\xe5\x9b\x9b') bytearray(b'\xe5\xa5\xb3') 18
```

7. where 语句

如果要读取指定条件的数据，可以使用 where 语句。

```
import mysql.connector as my

# 第一步：如果直接连接数据库，可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",    # 连接地址
    user = "root",         # 用户
    password = "123456",   # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()
```

```
sql = "select * from student where name = '张一'"

myCursor.execute(sql)
result = myCursor.fetchall()
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])

# 第二步：可以使用通配符%进行模糊查询
print()
sql2 = "select * from student where name like '%张%'"
myCursor.execute(sql2)
result2 = myCursor.fetchall()
for a in result2:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])
```

输出结果：

```
张一 女 8

张一 女 8
张二 男 28
张三 女 31
张四 女 22
张五 男 19
```

为了防止数据库查询发生 SQL 注入的攻击，可以使用%s 占位符来转义查询的条件。

execute()执行时需要传入两个参数：第一个是参数化的 SQL 语句；第二个是对应的实际参数值。函数内部会对传入的参数值进行相应的处理，以防止 SQL 注入。传入一个列表之后，MySQLdb 模块内部会将列表序列化成一个元组，然后调用 escape()函数把注入的 SQL 按空格分隔后，将正确的值赋给 SQL 语句。

示例代码如下：

```
import mysql.connector as my

mydb = my.connect(
    host = "localhost",    # 连接地址
    user = "root",         # 用户
    password = "123456",   # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

sql = "select * from student where name like %s"
data = ("%张%", )
myCursor.execute(sql, data)
result = myCursor.fetchall()
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])
```

输出结果：

```
张一 女 8
张二 男 28
张三 女 31
张四 女 22
张五 男 19
```

8. 排序

查询结果排序可以使用 order by 语句，默认的排序方式为升序，关键字为 asc，如果要设置降序排序，可以设置关键字 desc。

示例代码如下：

```
import mysql.connector as my

# 如果直接连接数据库，可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",    # 连接地址
    user = "root",         # 用户
    password = "123456",   # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

sql = "select * from student order by age desc"
myCursor.execute(sql)
result = myCursor.fetchall()
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])

print()
# 如果要设置查询的数据量，可以通过 limit 语句来指定，例如，获取 student 表中
# 年龄最大的 3 位同学的详细信息
sql2 = "select * from student order by age desc limit 3"
myCursor.execute(sql2)
result = myCursor.fetchall()
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])
```

输出结果：

```
张三 女 31
张二 男 28
张四 女 22
张五 男 19
```

```
李四 女 18
张一 女 8

张三 女 31
张二 男 28
张四 女 22
```

9. 删除语句

要慎重使用删除语句。删除语句要确保指定了 where 条件语句,否则会导致整表的数据被删除。

为了防止数据库查询时发生 SQL 注入的攻击,可以使用%s 占位符来转义删除语句的条件。

示例代码如下:

```
import mysql.connector as my

# 第一步: 如果直接连接数据库,可以把数据库写到 connect()中
mydb = my.connect(
    host = "localhost",    # 连接地址
    user = "root",         # 用户
    password = "123456",   # 密码
    database = "xuedao"
)

myCursor = mydb.cursor()

sql = "delete from student where name = %s"
data = ("张三",)
myCursor.execute(sql, data)
mydb.commit()

myCursor2 = mydb.cursor()
sql2 = "select * from student"
myCursor2.execute(sql2)
result = myCursor2.fetchall()
for a in result:
    print(a[0].decode("utf-8"), a[1].decode("utf-8"), a[2])
```

输出结果:

```
李四 女 18
张一 女 8
张二 男 28
张四 女 22
张五 男 19
```

3.4 本章小结

本章讲解了 Python 3 简介、环境搭建，以及基础语法的讲解，帮助学习者为后续深入学习 Spark 生态系统打下坚实基础。

3.5 课后习题

一、填空题

1. 在 Python 中，布尔类型的值是_____和_____。

2. Python 可以使用_____、_____和_____来表示字符串，引号的开始与结束必须是相同的类型。

3. _____函数用于将元组或字符串转换为列表。

4. 在 Python 定义的 6 个标准类型分别是 Number、String、Tuple、_____、_____和_____。

5. Python 的基础类型可以分为_____和_____两大类。

二、判断题

1. Python 中的变量不需要声明，每个变量在使用前都必须赋值，变量赋值以后该变量才会被创建。（　　）

2. break 语句用来终止循环语句，可以使用在循环中，也可以使用在循环外。（　　）

3. continue 语句用来跳出本次循环，告诉当前循环略过当前语句，然后继续进行下一轮循环。（　　）

4. pass 语句是空语句，为了保护程序结构的完整性，在循环语句中必须要有 pass 语句。（　　）

5. 在 Python 中，return 语句用于结束当前方法和返回值。（　　）

三、选择题

1. （　　）函数可以移除字符中所有的空格。

A. lstrip()　　B. startwith()　　C. endswith()　　D. replace()

2. 下列选项中，（　　）不属于 Python 语句。

A. pass　　B. break　　C. println()　　D. return

3. 已知列表 x=list(range(9))，执行语句 del x[:2]之后，x 的值是（　　）。

A. [1,3,5,7,9]　　B. [2,4,6,8]

C. [2,3,4,5,6,7,8]　　D. [0,1,2,3,4,5,6,7,8]

4. 运行下列程序后，输出结果是（　　）。

```
list = ["Python", "Java", "C++"]
print(len(list))
```

A. 1　　B. 2　　C. 3　　D. 13

5. 下列选项中，属于不可变类型的是(　　)。

A. str="Python"　　B. list=["Python","Java"]

C. dict={"name":"Python"}　　D. set={"Java","Python","Scala"}

四、编程题

1. 自定义函数计算三角形面积，将计算结果返回给调用者。

2. 利用函数编写计算方法，要求任意输入一个数(x)和步长(l)，判断 0 到 x 之间步长为 l 的所有数之和，并返回给调用者。

3. 如果有一组学生考试成绩数据，编写程序筛选出每位学生的成绩等级(筛选规则：学习成绩大于或等于 90 分的同学用 A 表示，60～89 分的用 B 表示，60 分以下的用 C 表示)。

4. 一个球从 100 米高度自由落下，每次落地后会反跳回原高度的一半再落下，求它从开始下落到第 10 次落地时，共经过多少米。

3.6 实训

1. 实训目的

掌握 Python 语言的基础语法。

2. 实训任务

利用 Python 语言编写基于控制台的商品购物系统，要求该系统包括商品插入、商品查询和商品筛选等模块，并提供安全退出系统的机制，系统运行效果如图 3.43 所示。

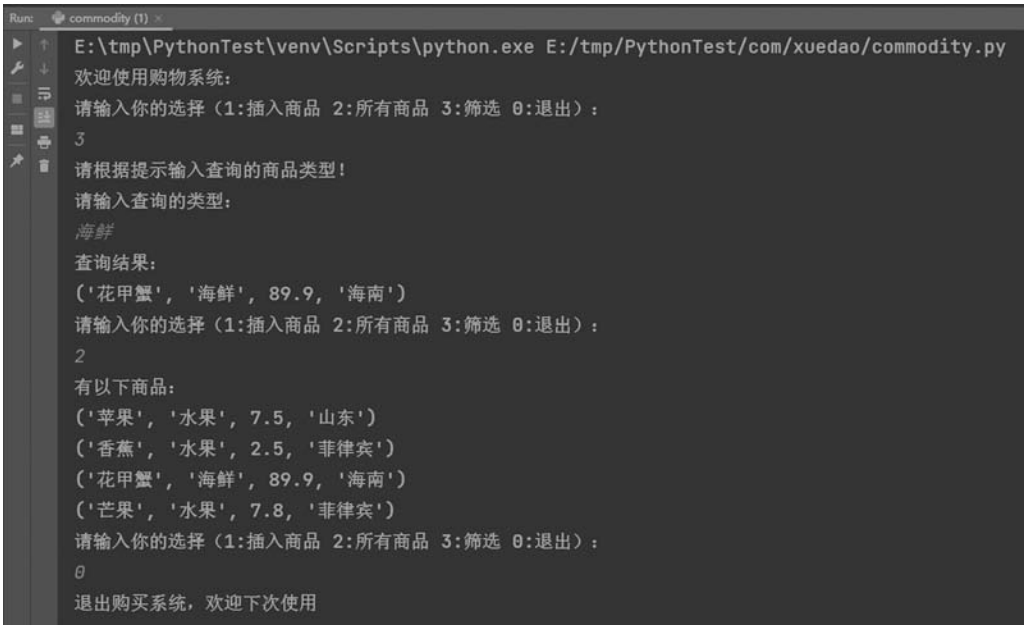

图 3.43　系统运行效果

3. 实训步骤

(1)创建数据模型。

在 MySQL 数据库中创建 xuedao 数据库，设置编码集为 UTF-8，如图 3.44 所示。

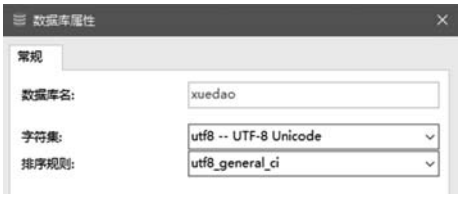

图 3.44　xuedao 数据库

在 xuedao 数据库中创建 commodity 表，表结构如图 3.45 所示。

图 3.45　commodity 表的结构

往 commodity 表中随意插入一些数据用于数据测试，如图 3.46 所示。

图 3.46　测试数据

(2) 在 PyCharm 开发工具中创建项目，并在项目中创建 commodity.py 文件，在 commodity.py 文件中编写业务逻辑，具体代码如下所示。

```
import mysql.connector as my


class Goods:
```

```
    # 查询所有商品
    def query(self):
        mydb = my.connect(
            host = "localhost", user = "root",
            password = "123456", database = "xuedao"
        )
        mc = mydb.cursor()
        sql = "select * from commodity"
        mc.execute(sql)
        result = mc.fetchall()
        print("有以下商品: ")
        for a in result:
            print(a)
        mydb.close()

    # 根据类型查询商品
    def typeQuery(self):
        mydb = my.connect(
            host = "localhost", user = "root",
            password = "123456", database = "xuedao"
        )
        mc = mydb.cursor()
        sql = "select * from commodity where type = %s"
        print("请输入查询的类型: ")
        t = input()
        dataVal = (t,)
        mc.execute(sql, dataVal)
        result = mc.fetchall()
        print("查询结果: ")
        for a in result:
            print(a)
        mydb.close()

    # 插入商品
    def insert(self):
        mydb = my.connect(
            host = "localhost", user = "root",
            password = "123456", database = "xuedao"
        )
        mc = mydb.cursor()
        sql = "insert into commodity values( %s, %s, %s, %s)"
        print("请输入商品名称(字符串类型): ")
        n = input()
        print("请输入商品类型(字符串类型): ")
        t = input()
        print("请输入商品价格(数值类型): ")
        p = input()
        price = float(p)
        print("请输入商品产地(字符串类型): ")
```

```
            m = input()
            dataVal = (n, t, price, m)
            mc.execute(sql, dataVal)
            mydb.commit()
            mydb.close()


    # 程序主入口
    if __name__ == '__main__':
        print("欢迎使用购物系统:")
        g = Goods()
        flag = True
        while flag:
            print("请输入你的选择(1:插入商品 2:所有商品 3:筛选 0:退出):")
            i = input()
            if i == "1":
                print("请根据提示插入商品信息!")
                try:
                    g.insert()
                except Exception as e:
                    print("插入失败!")
            elif i == "2":
                g.query()
            elif i == "3":
                print("请根据提示输入查询的商品类型!")
                g.typeQuery()
            elif i == "0":
                flag = False
                print("退出购买系统,欢迎下次使用")
            else:
                print("没有该选项,请重新输入!")
```