

# Python 程序设计基础

## (下)

同济大学计算机基础教研室

2021 年 10 月

# 目 录

第五章 列表、元组、字典与集合.....	1
5.1 列表与元组.....	1
5.1.1 引例-学生成绩计算 .....	1
5.1.2 列表操作.....	2
5.1.3 列表对象方法.....	5
5.1.4 操作列表的内置函数.....	9
5.1.5 列表推导式.....	12
5.1.6 元组.....	14
5.2 字典与集合.....	15
5.2.1 字典的创建.....	15
5.2.2 字典项操作.....	16
5.2.3 集合运算.....	18
5.3 综合应用.....	19
习题.....	26
第六章 文件.....	29
6.1 概述.....	29
6.1.1 引例——统计学生成绩.....	29
6.1.2 文件和文件路径.....	30
6.1.3 文件分类.....	31
6.1.4 文本文件编码.....	32
6.1.5 文件访问流程.....	33
6.2 文本文件.....	34
6.2.1 文本文件的打开与关闭.....	34
6.2.2 文件文件的读写.....	35
6.2.4 异常处理.....	37
6.2.4 CSV 格式.....	39
6.3 jieba 中文分词.....	40
6.4 图像文件.....	42
6.4.1 图像基本处理.....	43
6.4.2 图像的滤波和增强.....	47
6.5 综合应用.....	48
习题.....	51
第七章 面向对象程序设计基础.....	57
7.1 面向对象程序设计概述.....	57
7.1.1 面向对象的基本概念.....	57
7.1.2 类的基本特征.....	58
7.2 类和对象.....	59
7.2.1 类的定义.....	59
7.2.2 构造函数.....	61
7.2.3 类的实例及其成员的访问.....	62
7.3 继承和派生.....	63
7.4 迭代器和生成器.....	66

7.4.1 迭代器.....	66
7.4.2 生成器.....	67
7.5 模块、库、包.....	68
7.6 综合应用.....	69
习题.....	70
<b>第 8 章 数据可视化.....</b>	<b>73</b>
8.1 Python 图表绘制基础.....	73
8.1.1 引例—销售量柱状图.....	73
8.1.2 Matplotlib 绘图基础.....	74
8.1.3 图表绘制函数.....	79
8.2 二维图表.....	81
8.2.1 绘制线条.....	81
8.2.2 绘制饼图.....	86
8.2.3 绘制散点图.....	87
8.2.4 绘制直方图.....	88
8.3 三维图表.....	90
8.3.1 三维图表基础.....	90
8.3.2 绘制三维图表.....	92
8.4 综合应用.....	95
习题.....	100
<b>第九章 访问数据库.....</b>	<b>103</b>
9.1 访问 Access 数据库.....	103
9.1.1 ADO 基础.....	103
9.1.2 数据库基本操作.....	106
9.1.3 综合应用.....	113
9.2 访问其他数据库.....	115
9.2.1 访问 SQLite 数据库.....	116
9.2.2 访问 MySQL 数据库.....	121
习题.....	121
<b>实验六 列表、元组、字典、集合.....</b>	<b>125</b>
<b>实验七 文件.....</b>	<b>126</b>
<b>实验八 面向对象程序设计基础.....</b>	<b>127</b>
<b>实验九 数据可视化.....</b>	<b>128</b>
<b>实验十 数据库应用.....</b>	<b>132</b>

# 第五章 列表、元组、字典与集合

在各种类型的计算机应用中，数据常常被组织成线性表、队列、堆栈、树、图等数据结构。在这些数据结构中，应用最为广泛的是线性表，在 Python 中可以用列表和元组实现。列表和元组是最常见的序列类型，相当于其他程序设计语言中的数组。列表和元组的基础知识在第二章“数据表示与处理”中已有过初步的介绍，本章将深入、系统地进一步讨论。另外，由于字典和集合是无序序列，本章将一起介绍。

## 5.1 列表与元组

列表是 Python 处理数据的利器，它提供的方法可以快速完成数据的操作：分片、排序、统计等等。由于列表元素的数据类型没有限制，所以列表有很强的适应性和灵活性。例如，列表的元素也可是列表类型，因而可以容易表达科学计算中的矩阵。元组与列表相似，主要在操作上面有限制，所以被认为是轻量级的列表。

### 5.1.1 引例-学生成绩计算

下面首先通过一个引例说明为什么要使用列表，然后展开相应的知识。

**【例 5.1】** 输入 100 位学生的成绩，求所有学生的平均成绩和标准差。

标准差的计算公式为：
$$\text{std} = \sqrt{\sum_{k=1}^n (x_k - \bar{x})^2 / (n - 1)}$$

分析：如果不使用列表存储学生成绩，那么学生成绩就要输入两次。因为第一次输入在计算平均成绩时没有保存下来，所以计算标准差时要再一次输入。

```
import math
n=100                                #学生人数
total=0                              #记录所有学生的成绩之和
for i in range(n):
    x=int(input("请输入成绩: "))      #为计算平均值输入成绩
    total += x
ave=total/n                          #计算平均成绩
s=0                                  #记录标准差
for i in range(n):
    x=int(input("请输入成绩: "))      #为计算标准差输入成绩
    s=s+(x-ave)**2
std=math.sqrt(s/(n-1))               #计算标准差
print('总平均=',ave,'标准差=',std)
```

如果使用列表存储学生成绩，那么只需要输入一次就可以计算平均成绩和标准差。

```
import math
n=100
Scores=[]                            # 记录学成成绩的列表
```

```

for i in range(n):
    x=int(input("请输入成绩: "))
    Scores.append(x)                #追加当前输入成绩到列表中
total=0
for x in Scores:                    #遍历列表计算成绩之和
    total=total+x
ave=total/n
s=0
for x in Scores:
    s=s+(x-ave)**2                  # 用列表存储的数据计算标准差
std=math.sqrt(s/(n-1))
print('总平均=',ave,'标准差=',std)

```

当学生成绩用列表存储时，可以使用列表提供的方法和内置函数直接计算，使代码更加简洁。

```

import math
n=100
Scores=[]
for i in range(n):
    x=int(input("请输入成绩: "))
    Scores.append(x)
ave=sum(Scores)/len(Scores)        #利用内置函数操作列表计算平均成绩
lst=[(x-ave)**2 for x in Scores]    #列表推导式计算每个成绩与平均成绩差的平方
std=math.sqrt(sum(lst)/(n-1))
print('总平均=',ave,'标准差=',std)

```

上面的代码段中，Scores 是一个列表，它可以存储大量的数据，为计算提供了存储支撑，列表还支持列表推导式快速生成新的列表 lst，支持使用内置函数 sum、len 完成计算，提高编程效率。

## 5.1.2 列表操作

列表操作是指对列表的数据进行管理、查询、运算等操作方法，在前面章节中已经涉及，现在系统地介绍。

### 1. 遍历列表

遍历列表指通过循环引用列表的每个元素。遍历列表有三种方式：枚举方式、下标方式、枚举加索引方式。

#### (1) 枚举方式

枚举方式是指将列表中的每个元素直接取出进行处理，其使用形式为：

```

for item in L:
    对 item 进行处理

```

说明：按列表中元素的顺序，每执行一次循环体就取一个元素，直到取完所有的元素。

#### (2) 索引方式

采用 for 语句配合 range 函数完成，range 函数根据列表的长度生成列表的所有索引值，其格式为：

```
for i in range(len(L)):
    对 L[i]进行处理
```

### （3）枚举加索引方式

遍历列表时有些场合需要同时知道列表元素的索引和值，就应借助 enumerate 函数（enumerate 函数在第 5.1.4 中详细介绍）实现，其使用形式为：

```
for i, item in enumerate(L):
    对 i 和 item 进行处理
```

说明：循环时，i 和 item 存在对应关系，item 是第 i 个元素的值。例如，假定执行如下的代码：

```
L=[['李成',80],['赵华',61],['华连',91]]
print('带序号的枚举遍历结果：')
for i, item in enumerate(L):
    print('第',i,'项：',item)
```

则代码运行结果为：

```
第 0 项： ['李成', 80]
第 1 项： ['赵华', 61]
第 2 项： ['华连', 91]
```

列表遍历的三种方式，各有优势。第一种方式最简单，直接按顺序枚列表中的元素；第二种方式枚举列表元素的索引，根据索引访问列表元素；第三种枚举列表元素的索引和值。这三种方式各有场合，当遍历的目的是对列表元素本身值变更时，应该采用第二或第三种方式。

如下面代码尝试将列表的每个元素平方，分别用枚举方式和下标方式。

```
R=[1,3.47,8.21,65]
for item in R:                #枚举方式
    item=item**2
print(R)
for i in range(len(R)):      #下标方式
    R[i]=R[i]**2
print(R)
```

运行结果为：

```
[1, 3.47, 8.21, 65]
[1, 12.0409, 67.4041, 4225]
```

可以发现，枚举方式列表每个元素的值没有改变，但使用下标方式则达到了目的。因此，在遍历列表时，如果需要改变元素的值，则应该按下标索引的方式完成。

## 2. 删除列表

使用 del 函数可以删除列表，其使用形式为：

```
del 列表
```

例如，执行下面的代码：

```
Li=[1,4,9]
del Li          # 删除列表
print(Li)
```

运行时显示 “NameError: name 'Li' is not defined”，表示 Li 被删除后，它被从内存清理了，不再存在列表 Li。

### 3. 列表分片

#### (1) 分片

列表分片的基础请参阅第二章字符串的介绍，这里在前述基础上对分片作进一步明确。列表的分片格式为：

列表名称[start:end:step]

说明：两个冒号，至少出现一个，如写成[:]的形式，此时该冒号是指 start 和 end 之间的冒号，结果为列表的拷贝。

分片操作中，为了得到有意义的结果，在步长为正数时，end 应大于 start；步长为负数时，start 应大于 end。否则，得到的是空列表[]。

#### (2) 分片赋值

分片赋值指将列表指定位置分片的元素值变更为另一个列表的值。这种赋值能力可以方便删除列表原有的部分元素或插入一组新的元素。例如，将一个空列表付给一个分片，就可以将该分片包含的元素全部删除，而将一个列表赋值给一个空分片，则可以将其插入到列表中。请见下面的示例代码：

```
L=[1,2,3,4,5]
L[1:1]=[-9,-8,-7]
print(L)
```

结果为：[1, -9, -8, -7, 2, 3, 4, 5]

L[1:1]是一个空表，将该空表赋值为[-9,-8,-7]，实现将新元素插入原列表的指定位置。

也可以将指定的分片赋值空表，实现元素的删除，如下面的代码：

```
L=[1, -9, -8, -7, 2, 3, 4, 5]
L[1:4]=[]
print(L)
```

结果为：[1, 2, 3, 4, 5]

#### (3) 多维列表的分片

无论列表有几维，列表的分片只支持最外层的分片，对于二维列表而言，如果需要在列方向上实现分片是比较遗憾的。如下面的二维列表分片：

```
L=[[1, -9], [-8, -7], [2, 3], [4, 5], 10]
L1=L[1:3]
print(L1)
```

结果如下：[[-8, -7], [2, 3]]

### 4. 列表运算

列表运算符不多，只有+、+=和\*、\*=、成员运算符 in。其中+、\*运算符请参阅第二章

的描述，需要提醒的是，对列表而言，下面两种语句的计算结果虽然相同，但实现机制不同。

```
a=a+b  
a +=b
```

其区别在于，第一种方式 `a` 列表的首地址变更，而第二种方式，`a` 保持原来的首地址。

`in` 运算符用于判断一个元素在列表中是否存在，存在时返回 `True`，否则返回 `False`。其格式为：

元素 `in` 列表

**【例 5.2】** 已知一个列表中有某些元素重复，编程去掉重复元素，形成一个新列表。

分析：首先生成一个空列表，然后遍历原来的列表，每从原列表中取出一个元素后，判断其在新表中是否存在，不存在就将其加到新列表，遍历完原列表后，新列表就是所求结果。程序的代码如下：

```
L1=[1,3,6,3,3,7,8,1,1,1]  
newL=[]  
for i in L1:  
    if not i in newL:  
        newL +=[i]  
print(newL)
```

程序运行结果如下：[1, 3, 6, 7, 8]

### 5.1.3 列表对象方法

列表作为一种对象，有自己的方法，即列表自身拥有的函数。利用这些方法可以快速完成相应的操作，例如追加、删除、排序、插入等。

列表方法的使用形式如下：

```
列表名称.方法(参数)
```

下面介绍列表常用方法。

#### 1. append 方法

`append` 方法用于在列表的最后追加一个新元素，其使用形式为：

列表名称.`append`(新元素)

**【例 5.3】** 已知斐波那契数列的前两个数据为 0 和 1，计算其前 10 项。

分析：假定 `fib` 存储有斐波那契数列的前 `n` 项，`fib[-1]` 和 `fib[-2]` 表示最后两项，所以新的一项的值应为 `fib[-1]+fib[-2]`。

```
fib=[0,1]  
for i in range(2,10):  
    fib.append(fib[-1]+fib[-2])  
print (fib)
```

#表中已存在 2 项，所以从 2 开始

代码运行结果为：[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

#### 2. reverse 方法

`reverse` 方法将列表逆转，是无参数方法，其使用形式为：



列表名称.reverse()

见下面的示例代码:

```
L=[1,2,3,'laoyang']
L.reverse ()
print(L)
```

结果为: ['laoyang', 3, 2, 1]

### 3. sort 方法

当列表中所有元素之间都是可比较时, 可以用 sort 方法进行排序。sort 方法的使用形式为:

列表名称.sort(key=None, reverse=False)

说明:

① sort 方法使用时一般不需要参数, 数据按升序排序。例如, 执行如下的代码:

```
lst =[8,2,3,17.8]
lst.sort()
print(lst)
```

输出结果为: [2, 3, 8, 17.8], 数据按从小到大的顺序排序了。

② 排序时列表所有元素之间都应是可比较的, 否则会报错。例如, 执行如下的代码:

```
lst =['Tongji University', 1907, 'Fudan University', 1905]
lst.sort()
```

系统提示如下错误信息:

```
TypeError: '<' not supported between instances of 'int' and 'str'
```

意思是 ‘<’不支持整数和字符串实例间的比较。

③ 参数 reverse 决定排序的顺序是升序还是降序。若 reverse 为 False, 则升序, 否则降序。reverse 的缺省值为 False。例如, 语句 lst.sort(reverse=True)将 lst 列表中的元素降序排序。

④ 当对复杂列表进行排序时, 通常需要使用参数 key 指定排序关键字。参数 key 通常是一个函数, 列表所有元素根据该函数值的大小进行排序。例如, 假定有列表 lst=[['李成',80],['赵华',61],['华连',91]], 若要按成绩排序, 则应用使用下列语句:

```
lst.sort(key=lambda item:item[1])
```

其中, key 指定的函数是一个 lambda 函数。key 指定的函数可以是自定义函数、内置函数等。例如,

```
def fun(x):                                # 定义函数
    return x[1]
lst =[['李成',80],['赵华',61],['华连',91]]
lst.sort(key=fun)                          # 指定自定义函数排序
print(lst)
```

上述代码中 key 指定了一个自定义函数 fun。

### 4. count 方法

count 方法用于统计列表中指定元素出现的次数, 其使用形式为:

列表名称.count(元素值)

说明：统计结果是 count 方法的返回值。例如，下面的代码使用 count 方法统计 lst 列表中“上海”出现的次数：

```
lst=["杭州","苏州","上海","苏州","无锡","上海","苏州"]
n=lst.count('上海')
print(n)
```

**【例 5.4】**人类发现，蛋白质是由 20 种氨基酸（每种氨基酸用一个大写字母表示）组成的长链，还发现，氨基酸的组合有保守性（即某些氨基酸组合在一起的频率比较高）。现给定一条蛋白质，以 3 个氨基酸为单位，统计其中每种组合出现的次数。假定氨基酸序列如下：

```
MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQDNLSGAEKAVQVKV KALPDAQFE
VVHSLAKWKRQTLGQHDFSAGEGLYTHMKALRPDEDRLSPLHSVYVDQWDWERVMGDGERQFSTLKS
TVEAIWAGIKATEAAVSEEFGLAPFLPDQIHVHSQELLSRYPDLDAGRERAIKDLGAVFLVGIGKLS
DGHHRHDVRAPDYDDWSTPSELGHAGLNGDILVWNPVLEDAFELSSMGRIVDADTLKHQLALTGDEDRLE
LEWHQALLRGEMPQTIGGGIGQSRLTMLLLQLPHIGQVQAGVWPAAVRESVPSLL
```

分析：首先将给定的氨基酸序列作为一个字符串，将每三个氨基酸切片保存到一个列表中，最终通过列表遍历的方式，统计出每种组合出现的次数。

程序代码如下：

```
seq='MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQDNLSGAEKAVQVKV
KALPDAQFEVVHSLAKWKRQTLGQHDFSAGEGLYTHMKALRPDEDRLSPLHSVYVDQWDWERVM
GDGERQFSTLKS TVEAIWAGIKATEAAVSEEFGLAPFLPDQIHVHSQELLSRYPDLDAGRERAIK
DLGAVFLVGIGKLS DGHHRHDVRAPDYDDWSTPSELGHAGLNGDILVWNPVLEDAFELSSMGRIVD
ADTLKHQLALTGDEDRLELEWHQALLRGEMPQTIGGGIGQSRLTMLLLQLPHIGQVQAGVWPAAVR
ESVPSLL'
li=[]                                #保存分片的结果
length=3                            #3 个氨基酸的组合
for i in range(len(seq)-(length-1)):
    segments=seq[i:i+length]
    li.append(segments)
counts=[]                            #记录每种组合的次数，元素与 segments 有一一对应关系
segments=[]                          #去掉重复后的氨基酸组合
for seg in li:
    if not seg in segments:
        segments.append(seg)         #记录一个新的组合
        n=li.count(seg)
        counts.append(n)             #记录组合出现的次数
for i in range(len(counts)):
    print(segments[i],counts[i])
```

程序的输出结果为每个独立的三个氨基酸片段出现的次数。

## 5. index 方法

index 方法用于查找指定元素在列表中第一次出现的索引值，其使用形式为：

列表名称.index(元素值)

说明：

① 查找结果是 index 方法的返回值。如下面的代码，找出元素“宁波”第一次出现的

位置。

```
lst=["上海","杭州","苏州","宁波","南京","宁波"]
n=lst.index('宁波')
print(n)
```

结果为 3，即“宁波”第一次出现在列表中的第 3（从 0 开始计数）位置。

② 将上述程序再加两个语句

```
pos2=lst.index("北京")
print(pos2)
```

程序的运行结果为：ValueError: '北京' is not in list。

说明当列表不存在指定的元素值时，系统会报错。

index 方法配合 count 方法可以完成更复杂的任务，例如要找出例 5.4 中蛋白质 3 氨基酸组合中出现最多次数的氨基酸组合及其对应的次数。在统计了每种组合出现的次数后，可首先通过 max 函数寻找列表 counts 中的最大值，然后找出该最大值的索引。因为 counts、segments 两个列表的元素是一一对应的，所以输出索引指定的 segments 元素，就是出现频数最大的 3 氨基酸组合。程序追加的代码如下：

```
maxCount=max(counts) #max 为 Python 内置函数，可直接使用
index=counts.index(maxCount)
maxSeg = segments[index]
print(maxSeg, maxCount)
```

代码的运行结果如下：IAK 2。

思考题：index 方法有时会出现异常而导致程序崩溃，请你结合列表的 in 操作符，思考如何正确使用 index 方法。

## 6. pop 方法

pop 方法返回列表指定位置元素并将其从列表中删除，其使用形式为：

列表名称.pop(index=-1)

说明：index 参数省略时返回最后元素并删除。

pop 方法使列表有堆栈的特性。堆栈具有数据先进先出的特点。可以将列表竖起来看，如下面的列表：

x=[4, 10, 1, 97, -5, -7]

-7
-5
97
1
10
4

图 1.5.1 将列表看作堆栈

作堆栈

元素 4 是最先进入列表的，所以它在栈底，-7 是最后进入堆栈的，所以它在栈顶，其示意图如图 1.5.1。

如果在操作一个列表时，仅允许使用 pop 方法和 append 方法，则列表就是堆栈。

下面实例代码将列表 x 的栈顶元素弹出。

```
x=[4, 10, 1, 97, -5, -7]
n=x.pop()
print(n)
print(x)
```

代码运行结果如下：

-7

[4, 10, 1, 97, -5]

## 7. 自学内容

除了上述描述的方法外，列表还有很多其他的方法，如 `clear` 方法将表清空，`remove` 方法移除元素值为指定值的第一个元素，`insert` 方法在指定位置插入新元素，`extend` 将其他列表的元素逐个追加到当前列表中等。这些方法，从实现目的上讲，都可以使用分片赋值或直接给列表赋值的方法实现，有兴趣的读者，可以自行查阅资料尝试学习。

## 5.1.4 操作列表的内置函数

部分内置函数可以直接操作列表，这些函数包括：加和函数 `sum`，元素个数函数 `len`，最大、最小值函数 `max`、`min`，将指定函数映射到列表每个元素的函数 `map` 等等。本节挑选其中的部分内容加以介绍。

### 1. `len` 和 `sum`

`len` 返回列表的元素个数，`sum` 返回数值列表的和。其使用形式分别为：

`len(列表名称)`

`sum(列表名称)`

**【例 5.5】** 从键盘上输入一组数，数值间用逗号间隔，计算这组数的平均值和标准偏差。

分析：程序获得一组以逗号间隔的数，通过字符串的 `split` 函数分割后得到字符串列表，其每个元素是一个数字组成的字符串，通过 `float` 函数将每个元素转化为实数，在此基础上可以完成计算任务。程序代码如下：

```
data=input('请输入一组数，数间以逗号间隔：')
data=data.split(',')          # 按逗号将数据分割
for i in range(len(data)):     # 将每个数字字符串转化为对应的实数
    data[i] = float(data[i])
aver=sum(data)/len(data)      # 求平均值
std=0
for i in data:
    std += (i-aver)**2
import math
std = math.sqrt(std/(len(data)))
print('平均值={:.3f},标准偏差={:.3f}'.format(aver,std))
```

### 2. `max` 和 `min`

`max` 和 `min` 函数从列表中返回最大值和最小值。其使用形式分别为：

`max(列表名称)`

`min(列表名称)`

**【例 5.6】** 将数值列表的每个元素的数值调整到 0-1 之间。

分析：先求列表的最大值 `ma` 和最小值 `mi`，然后将列表的每个元素 `item` 按如下公式处理。

$$\text{item} = (\text{item} - \text{mi}) / (\text{ma} - \text{mi})$$

代码如下：

```
L=[12,34,21.3,45,19,87.2,76.4]
mi=min(L)
ma=max(L)
L=[(i-mi)/(ma-mi) for i in L]
print(L)
```

输出结果为：[0.0, 0.293, 0.124, 0.439, 0.093, 1.0, 0.856]

最大最小值函数并非只可用于数值列表，只要列表的元素之间是可比较的，则都可以用其求取最大最小值。例如：

```
L=['aaa','zzz','cfd']
print(max(L))
```

代码运行的结果为：'zzz'

### 3. map

`map` 函数用于将指定函数作用于列表的每个元素，其使用形式为：

`map(函数,列表名称)`

说明：

- ① 函数可以是内置函数，也可以是用户自定义函数。
- ② `map` 函数返回的结果是 `map` 对象，可用 `list` 函数将 `map` 对象转换为列表以便观察其值。

下面的代码将给定的数字字符串列表，转换为实数值组成的列表。

```
L=['123.6','24','63.1']
z=map(float,L)
L=list(z)
print(L)
```

代码运行结果为：[123.6, 24.0, 63.1]

**【例 5.7】** 修改例 5.5，利用 `map` 函数将数字字符串列表转换为数值列表，计算其均值和标准偏差。

分析：键盘输入的字符串经 `split` 函数分割后得到数字字符串列表，用 `map` 函数将 `float` 函数映射到列表在转换为列表可实现任务，程序代码如下：

```
data=input('请输入一组数，数间以逗号间隔：')
data=data.split(',') # 按逗号将数据分割
data=list(map(float,data))
aver=sum(data)/len(data) # 求平均值
std=0
for i in data:
    std += (i-aver)**2
import math
```

```
std = math.sqrt(std/(len(data)))
print('平均值={:.3f},标准偏差={:.3f}'.format(aver,std))
```

代码的运行过程和结果为：

请输入一组数，数间以逗号间隔：12,34,21.3,45,19,87.2,76.4

平均值=42.129, 标准偏差=27.142

#### 4. zip 和 enumerate

zip 函数用于操作多个列表，enumerate 函数操作一个列表，二者都返回迭代对象。

##### (1) zip 函数

zip 函数将多个列表的对应元素整合为元组，并返回以这些元组为元素的 zip 对象。其使用形式为：

```
zip(列表名称 1,列表名称 2,...,列表名称 n)
```

说明：

- ① 所有列表的元素个数应该等同
- ② zip 对象不能直接显示其包含的值，需要用 list 函数转换为列表再显示。

例如将两个元素个数相同的列表的对应元素组成元组，作为列表的元素形成新列表，代码如下：

```
names=['张仟','石戎','万行']
ages=[23, 21, 22]
ziped=zip(names,ages)
print(ziped)
print(list(ziped))
```

代码的运行结果为：

```
<zip object at 0x0000023E0BABB808>
[('张仟', 23), ('石戎', 21), ('万行', 22)]
```

输出结果的第一行，说明 zip 对象无法直接显示数据，第二行则可以看出，zip 将两个列表的对应元素组合在一起。

利用 zip 函数可使多个列表的遍历更具可阅读性。下面代码用 zip 实现学生姓名和年龄的对应遍历显示。

```
names=['张仟','石戎','万行']
ages=[23, 21, 22]
for name,age in zip(names,ages):
    print(name,age)
```

程序的运行结果为：

```
张仟 23
石戎 21
万行 22
```

##### (2) enumerate 函数

enumerate 函数将一个列表的元素与其下标索引组成元组。其使用形式为：

enumerate (列表名称)

下面代码将列表的元素值及其对应的索引一起输出。

```
names=['张仟','石戎','万行']
enu=enumerate(names)
print(enu)
print(list(enu))
```

代码运行结果如下：

<enumerate object at 0x0000023E0BA94FC0>

[(0, '张仟'), (1, '石戎'), (2, '万行')]

### (3) 迭代类型对象的使用

zip 和 enumerate 对象属于迭代类型，使用时需要注意的是：这类对象遍历完最后一个元素后，其使命即宣告结束，继续对其进行操作，虽然无语法错误，但程序结果往往超乎设计者的意外。如下面的代码所示：

```
names=['张仟','石戎','万行']
ages=[23, 21, 22]
ziped=zip(names, ages)
print("第一个 print 的输出： {}".format(list(ziped)))
print("第二个 print 的输出： {}".format(list(ziped)))
for name,age in ziped:
    print(name,age)
```

上述代码中，设计者想对 ziped 对象输出两次，最后又想通过 for 循环对其再次遍历，能得到怎样的结果呢？

第一个 print 语句中，list 函数对 ziped 进行第一次遍历，输出结果是正常的。执行第二个 print 语句时，由于此时 ziped 已经完成遍历，所以，其输出结果应该是空表。而对于 for 循环，因 ziped 已处于迭代最后位置，所以 for 循环无结果输出。

运行上述程序，其输出结果如下：

第一个 print 的输出： [('张仟', 23), ('石戎', 21), ('万行', 22)]

第二个 print 的输出： []

## 5.1.5 列表推导式

列表推导式是 Python 提供了一种简洁式的快速对列表进行遍历、过滤进而达到重组的方案。其语法格式为：

```
[结果表达式 for 表达式 1 in 列表 1 [ if 条件 1
    for 表达式 2 in 列表 2 if 条件 2
    .....
    for 表达式 n in 列表 n if 条件 n]
]
```

说明：

- ① 结果表达式可以为任何类型。
- ② 根据 for 循环计算“结果表达式”，并以“结果表达式”作为元素形成新的列表。
- ③ 条件用于选择符合条件的数据。

例如，对列表 `lst=[['李成', '男', 80], ['赵华', '男', 61], ['华连', '女', 91], ['贾茜', '女', 75], ['程泽', '男', 95], ['彭鹏', '男', 89], ['张艺', '女', 100], ['黎玲', '女', 99]]`，求女生成绩列表，其列表推导式为：

```
girlScores=[stu[2] for stu in lst if stu[1]=='女']
```

**【例 5.8】** 从鸢尾花数据集中抽取山鸢、变色鸢尾两种花各 3 个样本，每株样本用萼片的长和宽、花瓣的长和宽 4 个特征，山鸢的类别用 0 表达，变色鸢尾用 1 表达。特征数据用二维列表 `X` 存储，类别用一维列表 `y` 存储，根据列表 `y` 值，将山鸢的三个样本挑选出来形成新的列表。

分析：通过下标索引遍历列表，`y` 元素为 0 的样本对应的 `X` 同位置的元素就是需要过滤的样本，程序的代码如下：

```
X=[[5.1, 3.5, 1.4, 0.2], [4.9, 3.1, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2],  
   [7.1, 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5], [6.9, 3.1, 4.9, 1.5]]  
y=[0,0,0,1,1,1]  
result=[X[i] for i in range(len(X)) if y[i]==0]
```

**【例 5.9】** 实现矩阵的平铺。矩阵是二维列表，平铺是指将矩阵的后续行，连接到前行上，形成一维列表。

分析：按行遍历二维列表可以用两个 for 循环完成，外层循环访问行，内存循环负责行中的每个元素，将内循环访问的元素作为列表的元素可完成任务，故可用下面的列表推导式实现任务：

```
X=[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
res=[item for row in X for item in row]  
print(res)
```

程序的运行结果为：[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

**【例 5.10】** 用列表推导式实现矩阵的转置。矩阵的转置是指矩阵的行变列，列变行。

分析：设定按列遍历二维列表的循环，第 `i` 次循环，取列表每行的第 `i` 个元素组成列表，将它作为列表推导式的一个元素可实现任务。所以，矩阵转置的列表推导式方案如下：

```
X=[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
Xt=[[row[i] for row in X]          #每个元素是一个列表，由矩阵的某列组成  
    for i in range(len(X[0]))]      # 循环以矩阵的列展开  
print(Xt)
```

程序的输出结果为：[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]



## 5.1.6 元组

元组被认为是轻量级的列表，其类型为 `tuple`。对列表的很多操作也适用于元组，如遍历、切片、下标的使用，`+`、`*`、`in` 运算等。元组被看作一个不可变更的列表，系统开销小，操作速度快，对无需修改的数据有一定的保护作用，所以有其自身优势。元组和列表之间，可以通过 `list` 和 `tuple` 函数互相转换，元组的初步知识请回顾 2.5.2 节。

### 1. 元组的创建

元组的创建除了用 `()` 直接表达或用 `tuple` 函数外，任何以逗号间隔的一组数值、变量、表达式，都被组织成元组。如下面语句：

```
b=12.3
a=b/3,1,2          #逗号间隔的值,自动形成元组
print(a)
```

典型的两变量值交换语句

```
x,y=y,x
```

其实质是将 `y,x` 组成元组，再通过序列解包赋值给 `x,y`，达到值交换的目的。

### 2. 元组元素操作及分片

元组元素通过“元组[下标]”使用。但由于元组元素的初值不可变更，所以对元组元素赋值是不允许的。

当元组元素的类型是复合型数据时，元组元素的属性值可以改变。如下面的代码示例。

```
Li=([11, 22], 33,88)
Li[0].append(-55)
print(Li)
```

元组 `Li` 的第一个元素是列表，此处调用了列表的 `append` 方法为其增加了一个元素，元组元素的属性值改变了，但元组 `Li` 的第一个元素的首地址并没有改变。代码的运行结果如下：

```
([11, 22, -55], 33, 88)
```

但采用如下的语句，则是不允许的：

```
Li[0] += [-55]
```

程序运行结果为：

```
TypeError: 'tuple' object does not support item assignment
```

意思为：元组对象不支持表项的赋值。

这是由于该语句试图改变第一个元素的存储首地址，被 Python 拒绝。

元组的分片与列表的分片采用同样的规则，只是返回结果的类型仍然是元组。

### 3. 操作元组的内置函数

对列表操作的函数，都可以用于元组，例如 `sum`，`len`，`max`，`min`，`map` 等，其操作与列表类似。

**【例 5.11】** 求元组的最大最小值、元组的和

```
Li=(11, 22,33,44,55,66,77,88)
maxNum=max(Li)
minNum=min(Li)
s=sum(Li)
lenth=len(Li)
print("最大={},最小={},和={},元素个数={}".format(maxNum,
    minNum,s,lenth))
Li=('11', '22','33','44','55','66','77','88')
L1=tuple(map(float,Li))
print(L1)
```

程序的运行结果如下：

```
最大=88,最小=11,和=396,元素个数=8
(11.0, 22.0, 33.0, 44.0, 55.0, 66.0, 77.0, 88.0)
```

## 5.2 字典与集合

与列表和元组相比，字典和集合是无序的数据序列。字典（dict）是按“键/值”对存储的一种数据结构，它通过关键字，快速查找对应的信息，集合则强调元素间不重复。关于字典和集合的基础内容可参阅 2.5.4。

数据采用列表存储与采用字典存储会导致查询方法和效率有很大差异。如将学生名字和成绩用列表存储时

```
name_Score=[('李成',80),('赵华',61),('华连',91),('高薇',98),('彭池',71),('钟毓',77)]
```

则查询某个同学的成绩时，需要通过循环找到该同学然后给出其成绩。但数据用字典存储时

```
name_Score={'李成':80,'赵华':61,'华连':91,'高薇':98,'彭池':51,'钟毓':77}
```

则可直接通过 `name_Score[姓名]` 的方式获得学生的成绩。

### 5.2.1 字典的创建

创建字典最直接的方式是将数据按格式写进{}或用 dict 函数，除此之外，还有多种方式生成字典。

#### 1. 逐项建立

先建立空字典，然后逐步添加新项完成，其使用格式为：

```
字典[键]=值
```

例如下面的代码，通过逐步增加表项建立字典。

```
Dcountry={}
Dcountry['中国']='北京'
Dcountry['德国']='柏林'
```

```
Dcountry['美国']='华盛顿'
print(Dcountry)
```

结果为:

```
{'中国': '北京', '德国': '柏林', '美国': '华盛顿'}
```

由于键是唯一的, 所以, 采用“字典[键]=值”方式增加字典项时, 如果原来已经存在指定的键, 则运行结果会替换原来的键值对。

## 2. 用 dict 函数转换

dict 函数常用用来将合适的序列数据转换为字典。

① 二维的列表或元组, 只要能清晰地表达“键/值”关系, 就可以通过 dict 函数转换成字典。例如:

```
data=[['中国','北京'],['德国','柏林'],['美国','华盛顿']]
Dcountry=dict(data)
print(Dcountry)
```

② 两个元素相同的列表或元组, 结合 zip 函数, 也可以经 dict 函数转化为字典。如下面例子:

```
names=['张仉', '石戎', '万行']
ages=[23, 21, 22]
dict1=dict(zip(names,ages))
print(dict1)
```

代码的运行结果为: {'张仉': 23, '石戎': 21, '万行': 22}

## 5.2.2 字典项操作

字典对象拥有很多方法, 对字典 d, 表 1.5.1 中列出了词典操作的常用方法。

表 1.5.1 字典操作的常用方法

操作方法	说明
len(d)	返回 d 中“键/值”对的数量, 即表项数目
d.keys()	返回字典所有的键
d.values()	返回字典所有的值
d.items()	返回所有键值对
d.clear()	清空字典内容
d.get(key, default=None)	返回字典键 key 的值, 如果键在字典中不存在, 返回 default 指定的值
d.pop(key,[default])	若字典中存在键 key, 则删除并返回 key 对应的值, 不存在则返回 default
d[k]	出现在表达中, 返回关联到键 k 上的值
d[k] = v	将值 v 关联到键 k 上

del d[k]	删除键 k 的项
k in d	判断 k 是否是 d 中的一个键

遍历字典，可以通过遍历字典的键完成，也可以用遍历 items 方法的返回值完成。如下面的代码通过遍历键实现了字典的遍历。

```
DzipCode={'北京':100000,'上海':200000,'南京':210000,'深圳':518000}
DzipCode['广州']=510000
for key in DzipCode:                                #字典遍历，通过遍历键获得值
    print(key,DzipCode[key])
print("项数",DzipCode.items())
print("所有的键",list(DzipCode.keys()))
print("所有的值",list(DzipCode.values()))
print("上海的邮政编码：",DzipCode['上海'])
```

代码的运行结果为：

北京 100000

上海 200000

南京 210000

深圳 518000

广州 510000

项数 dict\_items([('北京', 100000), ('上海', 200000), ('南京', 210000), ('深圳', 518000), ('广州', 510000)])

所有的键 ['北京', '上海', '南京', '深圳', '广州']

所有的值 [100000, 200000, 210000, 518000, 510000]

上海的邮政编码： 200000

**【例 5.13】**将例 4.21 统计英文句子每个字符出现的次数的函数，形成以字符为键，次数为值的字典并返回。

分析：因为统计过程是逐个字符进行的，所以可以从空字典出发，每访问一个字符形成一个字典项的方式完成任务，程序的代码如下：

```
def charFreq(s):
    chars=list(set(s))                # 整理不重复的字符
    chars.sort()
    freq={}                          #建立空字典
    for char in chars:
        times=s.count(char)          #记录字符出现次数
        freq[char]=times             #为字典增加一项
    return freq                      # 返回字典
str1=input('请输入如字符串： ')
charFreq=charFreq(str1)             #调用函数
print(charFreq)
```

## 5.2.3 集合运算

创建集合的方法请参见第 2.5.3 节，本节仅介绍集合的运算。集合运算可以通过运算符进行，表 1.5.2 列出了集合运算的运算符及实例运算结果。

表 1.5.2 集合运算符

运算符	意义	示例	结果 设初值为: s1=set((1,3,7,8)), s2=set((2,8,10))
-	差集	s1-s2	{1, 3, 7}
&	交集	s1 & s2	{8}
	并集	s1   s2	{1, 2, 3, 7, 8, 10}
^	对称差集	s1 ^ s2	{1, 2, 3, 7, 10}

验证表 1.5.2 实例结果的代码如下：

```
s1=set((1,3,7,8))
s2=set((2,8,10))
print('差集: ',s1-s2)
print('交集: ',s1 & s2)
print('并集: ',s1 | s2)
print('对称差集: ',s1 ^ s2)
print('检验元素存在: ', 7 in s2)
print('集合元素个数: ',len(s1))
```

集合运算也可以用集合对象的方法完成。设有集合对象 s 和 s1，表 1.5.3 列出了集合对象的部分方法。

表 1.5.3 集合对象的方法

方法	意义
s.add(x)	为集合增加新元素 x
s.update(x)	x 必须是序列类型，x 的每个元素成为 s 的一项，若 x 是字典，取字典的键作为元素
s.remove(x)	移除元素 x，若元素不存在，则报错
s.discard(x)	移除元素 x，若元素不存在，系统不报错
s.clear()	清空集合
s.difference(s1)	同 s-s1
s.intersection(s1)	s 与 s1 的交集，同 s & s1
s.issubset(s1)	s 是否是 s1 的子集，是则返回 True
s.union(s1)	s 与 s1 的并集，同 s   s1
s.symmetric_difference(s1)	返回两个集合中不重复的元素集合,同 s ^ s1

通过集合对象的方法实现集合运算的代码如下：

```
s1=set((1,3,7,8))
s2=set((2,8,10))
```

```
print('差集: ',s1.difference(s2))
print('交集: ',s1.intersection(s2))
print('并集: ',s1.union( s2))
print('对称差集: ',s1.symmetric_difference( s2))
```

## 5.3 综合应用

本章重点讲解了序列数据的结构及相应的处理方法，结合循环、函数，学习者能解决更加复杂的问题。本节通过几个综合例子，向学习者展示如何利用序列数据解决实际问题。

## 1. 数据分拣

数据分拣是指根据某种条件，从已知数据序列中，挑选出符合要求的元素。

**【例 5.14】** 设原始考试数据是所有班级混合考试的列表，现在想将某个班级同学成绩分拣出来。已知条件为：混考成绩表及班级同学的名单。

分析：班级同学名单是混考成绩表的子集，故可通过数据序列的 `in` 运算符实现分拣。  
求解问题的思路如下：

- ①整理学生成绩到一个对象列表 `studentList`
- ②整理班级学生名单到列表 `C`，`C` 列表中只记录学号
- ③令结果列表 `R=[]`
- ④遍历列表 `studentList`
- ⑤ `studentList` 的每个元素，通过 `in` 运算符确认其在列表 `C` 中，则将其追加到 `R` 中
- ⑥输出 `R`

完整的程序代码如下：

```
studentList=[]
studentList.append(('1200001','许枫',65,84,61))
studentList.append(('1200002','周杰',76,65,72))
studentList.append(('1200003','李丽',87,76,91))
studentList.append(('1200004','江红燕',76,90,78))
C=['1200002','1200003']
R=[]
R=[stu for stu in studentList if stu[0] in C]          # 分栋
for stu in R:
    print("{}{}{}{}{}{}".format(stu[0],stu[1],
        stu[2],stu[3],stu[4]))
```

程序的运行结果如下:

1200002 周杰	76	65	72
1200003 李丽	87	76	91

## 2. 矩阵相乘及函数实现

矩阵可以用二维列表表示，两个矩阵 A 和 B 相乘时遵循如下的原则：

左矩阵的列数等于右矩阵的行数，结果矩阵的行数等于左矩阵的行数，列数等于右矩阵

列数。

结果矩阵的第  $i$  行  $j$  列的元素，是左矩阵的第  $i$  行和右矩阵第  $j$  列，对应元素相乘的加和，如下式：

$$C_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

矩阵相乘的示意图如图 1.5.2 所示：

图 1.5.2 矩阵相乘示意图

**【例 5.15】** 根据矩阵计算的公式，编写程序，计算  $C=A \times B$ 。

程序代码如下：

```
A=[[1, 3,4],[5,1,0,0]]          # 2*3 的矩阵
B=[[3,1,5],[7,0,2],[2, -1, 3]]    # 3*3 矩阵
C=[ ]                             #结果矩阵
row=len(A)                        # 取 A 矩阵的行数
col=len(B[0])                     # 取 B 矩阵的列数
for i in range(row):
    oneRow=col*[0]                 # 结果矩阵的一行，每个元素的值设 0
    for j in range(col):
        s=0.0
        for k in range(len(B)):   #计算结果矩阵的第 i 行第 j 列元素
            s += A[i][k]*B[k][j]
        oneRow[j]=s
    C.append(oneRow)
print(C)
```

程序的运算结果为：

[[32.0, -3.0, 23.0], [22.0, 5.0, 27.0]]

矩阵相乘、转置、格式化输出应用非常普遍，将其函数化可大大提高编写矩阵计算程序的效率。下面的例子求矩阵  $X$  的协方差矩阵（ $X$  的转置和  $X$  相乘的结果）。它利用了矩阵的转置、矩阵的相乘运算。

**【例 5.16】** 编写函数，分别实现矩阵相乘、转置、格式化输出

```
def matDot(A,B):                  # 计算 C=A*B，返回 C
    C=[ ]                         #结果矩阵
    row=len(A)                    # 取 A 矩阵的行数，也是 C 的行数
    col=len(B[0])                 # 取 B 矩阵的列数，也是 C 的列数
    for i in range(row):
        oneRow=col*[0]            # 结果矩阵的一行，每个元素的值设 0
        for j in range(col):
            s=0.0
            for k in range(len(B)): #计算结果矩阵的第 i 行第 j 列元素
```

```

        s += A[i][k]*B[k][j]
    oneRow[j]=s
    C.append(oneRow)
return C
def printMat(A):
    #矩阵的格式化输出
    for row in A:
        for item in row:
            print(item,'\t',end="")
        print()
transpose=lambda X:list(map(list,zip(*X))) # 矩阵转置函数

```

在以上三个函数的支持下，求取矩阵的协方差矩阵的主程序代码如下：

```

X=[[1,2,3],[4,5,6]]
Xt=transpose(X)
matCoe=matDot(Xt,X)
printMat(matCoe)

```

程序的运行结果为：

```

17.0    22.0    27.0
22.0    29.0    36.0
27.0    36.0    45.0

```

### 3. 感知器算法分类建模

美国植物学家埃德加·安德森收集了不同类别的鸢尾花的花瓣、萼片数据形成了著名的鸢尾花数据集。从数据集中分别选择出 10 朵山鸢尾和 10 朵变色鸢尾花瓣的长、宽作为数据集，以长为横轴、宽为纵轴，将 20 朵鸢尾花作为数据点在二维平面上显示出来，如图 1.5.3 所示。图中可见两种花分布在不同的区域。现在问，能不能通过机器学习自动建立图示直线的方程将两类样本分开？

令花瓣长为  $x_1$ ，宽为  $x_2$ ，则分类直线的方程可表达为：

$$a_1x_1+a_2x_2+b=0$$

山鸢尾样本在直线的上方，变色鸢尾样本在直线的下方，所以将任一样本带入方程后，必定满足下面不等式。

山鸢尾： $a_1x_1+a_2x_2+b>0$

变色鸢尾： $a_1x_1+a_2x_2+b<0$

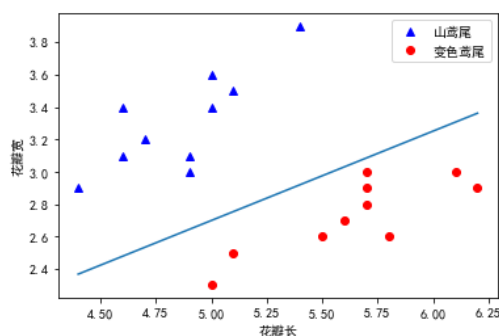


图 1.5.3 山鸢尾和变色鸢尾样本分布



鸢尾花分成两类，山鸢尾用+1 表示，变色鸢尾用-1 表示，用变量  $y$  存储。如果直线方程不合理，则会导致部分样本误判，如果样本  $i$  误判，则它必定满足下面不等式：

$$y^{(i)}(a_1x_1^{(i)}+a_2x_2^{(i)}+b)<0$$

式中，上标(i)表示第  $i$  个样本，下同。

感知器算法以误判样本来调节方程的系数，在给定初值  $a_1$ 、 $a_2$ 、 $b$  及学习速率  $r$  后，对于误判样本  $i$ ，系数调节的公式如下：

$$a_1=a_1+ryx_1^{(i)}$$

$$a_2=a_2+ryx_2^{(i)}$$

$$b=b+ry^{(i)}$$

直线方程的好坏通过如下的函数表达：

$$\text{Loss}(a_1,a_2,b)=\sum \max(0, -y^{(i)}(a_1x_1^{(i)}+a_2x_2^{(i)}+b))$$

该函数被称为损失函数，对于判断正确的样本，损失为 0，所以对函数值无贡献，而误判的样本，其对应的表达式值必定为负数，取负号后为正数，因此被计入损失函数。损失函数值越大，说明误判的样本多，直线方程不合理。当损失函数为 0 时，说明所有样本分类正确。

**【例 5.17】**设计感知器算法程序，根据给定的鸢尾花数据，求解感知器判别模型方程

分析：首先定义损失函数，损失函数返回每个样本的损失函数贡献值及函数值，主程序则根据损失函数返回值调节方程的系数，直到所有样本正确分类，算法逻辑如下：

#### (1) 损失函数定义

① 给定记录样本特征的二维列表  $x$ ，记录样本分类值的一维列表  $y$ ，系数  $a=[a_1,a_2]$ ，截距  $b$

② 将每个样本带入公式 $(a_1x_1+a_2x_2+b)*y$ ，计算其损失函数贡献值，放入列表  $\text{temp}$

③ 根据  $\text{temp}$  和损失函数公式计算损失函数值  $\text{loss}$

④ 返回  $\text{loss}$  和  $\text{temp}$

#### (2) 感知器算法逻辑

① 给定系数初值， $a=[1.0,1.0]$ ； $b=1.0$ ；学习速率  $\text{rate}=0.1$

② 指定迭代次数，进入循环

③ 根据  $a$ ， $b$ ， $\text{rate}$ ，调用损失函数获得损失函数值和每个样本损失贡献值

④ 如果损失函数等于 0，或者到达最大迭代次数，退出循环，程序终止

⑤ 找误判样本，按公式调节系数，转③进入下次循环

感知器算法程序代码如下：

```
def lossF(x,y,a,b):
    # x 是 2 维列表，每行一个样本,y 是样本分类符
    temp=[]
    for one in x:
        temp1=[one[i]*a[i] for i in range(len(one))]
        temp1=sum(temp1)+b
    # 每个样本的 a1x1+a2x2+b 的取值
```

```

        temp.append(temp1)
    temp2=[]
    for i in range(len(temp)):
        temp2.append(temp[i]*y[i])    # 每个样本函数值与真值的乘积
    loss=0
    for i in temp2:
        loss += max(0,-i)            # 计算损失函数，每项取负值，即误判样本
    return loss,temp2                # temp2 返回有用，用于找误判，调节系数
# 数据准备, x1 是山鸢尾，x2 是变色鸢尾
x1=[[5.1, 3.5 ], [4.9, 3. ], [4.7, 3.2], [4.6, 3.1], [5. , 3.6],
     [5.4, 3.9], [4.6, 3.4], [5. , 3.4 ], [4.4, 2.9], [4.9, 3.1]]
x2=[[5.5, 2.6 ], [6.1, 3. ], [5.8, 2.6], [5. , 2.3], [5.6, 2.7],
     [5.7, 3. ], [5.7, 2.9], [6.2, 2.9], [5.1, 2.5], [5.7, 2.8]]
x=x1+x2                            #合成一个矩阵
y=[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]    #样本分类标识
y1=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
y += y1
a=[1.0,1.0];b=1.0;rate=0.01        #给系数初值
for i in range(100):
    loss,temp=lossF(x,y,a,b)        #计算损失函数
    print(loss)
    if loss==0.0:
        break
    for j in range(len(temp)):
        if temp[j] <0:                #找误判样本
            a[0] +=rate*y[j]*x[j][0]    #更新系数
            a[1] +=rate*y[j]*x[j][1]
            b +=rate*y[j]
print("loss=",loss)

print("model is { :10.3f}x1 { :+10.3f}x2 { :+10.3f}".format(a[0],a[1],b))

```

程序运行的部分结果如下：

```

loss= 31.774599999999957
loss= 1.7453000000000035
loss= 33.645599999999966
loss= 0
model is      0.088x1      -0.399x2      +0.700

```

## 5. 数据协同过滤

数据协同过滤在商业推介中应用比较广，其原理是将积累的客户营销数据与目标客户比对，找到与目标客户最相似的客户，以相似客户的喜好向目标客户推介。

**【例 5.18】**某超市积累了一组近两年客户对奶制品的销售数据，包括品牌及选购次数。现有一新客户，请用数据协同过滤的原理，向新客户推介奶制品。

分析：本例中将“爱好相似客户”定义为购买了产品种类重叠最多的客户。计算现有客户群每个客户购买的产品集合与目标客户购买产品集合的交集，取交集最大的客户作为目标

客户的爱好相似客户，将其购买的产品推介给目标客户。

首先将用户购买记录整理成字典格式：每个客户为键，其购买记录作为值。购买记录本身也形成一个字典，以产品为键，购买次数为值。通过将每个客户的购买记录形成集合与目标客户的购买记录集合作交集运算，达到推荐产品的目的。程序实现的代码如下：

```
userData={'客户 1':{'伊利安慕希酸奶':3,'蒙牛纯甄':6,'光明莫斯利安':3,'三元':3,'蒙牛冠益乳':3},\
          '客户 2':{'君乐宝':5,'安佳':5,'光明研简':5,'优诺':3,'兰格格':5,'光明莫斯利安':3},\
          '客户 3':{'兰格格':5,'欧德堡':4,'三元':3,'光明如实':2,'伊利畅轻':5,'伊利安慕希酸奶':7},\
          '客户 4':{'蒙牛冠益乳':3,'三元':3,'光明如实':2,'伊利畅轻':5,'伊利安慕希酸奶':7}}

目标客户={'蒙牛冠益乳':3,'三元':3,'伊利安慕希酸奶':3}          #目标客户的购买记录
commonLen=0
for k,v in userData.items():
    theCommonMilk=set(v.keys()) & set(目标客户.keys())          #一个客户与目标客户购买产品的交集
    if commonLen<len(theCommonMilk):                             #记录交集最多的客户作为相似客户
        相似客户=k
        commonMilk=theCommonMilk                                #记录最相似客户的交集

print('喜好相似的客户为：',相似客户)
print('共同产品：',commonMilk)
# 推荐产品
userMilk=set(userData[相似客户].keys())-commonMilk              #相似客户的产品集合-交集，作为推荐
print('推荐产品',userMilk)
```

程序的运行结果如下：

喜好相似的客户为： 客户 4

共同产品： {'伊利安慕希酸奶','三元','蒙牛冠益乳'}

推荐产品 {'伊利畅轻','光明如实'}

**【例 5.19】**莫尔斯电码采用了短脉冲和长脉冲(分别为点和点划线)来编码字母和数字。将莫尔斯电码的编码规则组成字典，输入英文字母组成的英文句子，根据字典将句子翻译成莫尔斯电文，电文不同字母间用空格间隔。部分莫尔斯编码如下：

字符	电码	字符	电码	字符	电码	字符	电码
A	.-	B	-...	C	-.-.	D	-..
E	.	F	..-.	G	--.	H	....
I	..	J	.-..	K	-.-	L	.-..
M	--	N	-.	O	---	P	.-.-
Q	--.-	R	.-.	S	...	T	-
U	..-	V	...-	W	.-.-	X	-. -
Y	-.--	Z	--..	.	.-.-.-	?	..--..
空格	/						

分析：首先形成字符与对应编码的字典，读入英文句子后，通过遍历每个字符，根据字

典输出其对应的莫尔斯电码。

程序代码如下：

```
Dict={'A': '-.-', 'B': '-...', 'C': '-.-.', 'D': '-..', \
'E': '.', 'F': '..-', 'G': '--.', 'H': '....', \
'I': '..', 'J': '.---', 'K': '-.-', 'L': '-..', \
'M': '--', 'N': '-.', 'O': '---', 'P': '-.-.', \
'Q': '--.-', 'R': '-.-.', 'S': '...', 'T': '-', \
'U': '..-', 'V': '...-', 'W': '-.-', 'X': '-...', \
'Y': '-.-', 'Z': '--..', ' ': ' ', '?': '..--..', ' ': '/' }
sentence=input('please input a sentence:')

sentence=sentence.upper()
for ch in sentence:
    code=Dict.get(ch)
    print(code,' ',end="")
```

## 习题

### 一. 选择题

1. 已知列表 `L=[46,37,10]`, 令 `L1=L,L[1:2]=[65,34]`, 则 `L1` 的结果是什么?  
A `[46,65,34,10]` B `[65,34,37,10]` C `[46,37,10]` D `[37,10,65,34]`
2. 关于列表排序方法 `sort()`, 下面说法正确的是\_\_\_\_\_  
A 只要列表的元素之间可以比较, 其 `sort` 方法都有效  
B `Sort` 能自动适应列表的各种类型元素完成排序  
C 只有元素是数值类型的列表, 才能调用其 `sort` 方法  
D 只有元素是字符类型的列表, 才能调用其 `sort` 方法
3. 遍历列表 `L` 时, 同时取列表元素的值和元素的索引值, 最好的方法是使用\_\_\_\_\_  
A `for i,v in L:` B `for i,v in zip(L):`  
C `for i,v in enumerate(L):` D `for i,v in len(L):`
4. 向列表 `L` 追加一个整数元素 `1`, 然后输出整个列表, 应该使用\_\_\_\_\_  
A `print( L.append(1) )` B `print(L+1)`  
C `print( L+[1] )` D `print([L]+[1])`
5. 字典和集合在初始化为非空序列时, 都使用\_\_\_\_\_。  
A `{}` B `[]` C `()` D `' '`
6. 为字典 `di` 增加一个新的表项 “张制:78”, 应该使用语句\_\_\_\_\_  
A `di[张制]=78` B `di['张制']=78`  
C `di{张制}=78` D `di{'张制'}=78`
7. 遍历字典 `d` 的键、值对, 下面哪组语句不能实现\_\_\_\_\_  
A `for k,y in d:` B `for k in d:` C `for k,v in d.items():` D `for k in d.keys():`
8. 有列表记录了学生两门课的考试成绩, `scores=[['宋安',80,91],['周璐',61,89],['汤琳',81,67],['孙忠',95,99]]`, 现在想按学生两门课的总分为准将列表按升序重排, 使用的语句为\_\_\_\_\_  
A `scores.sort(key=lambda o:o[1]+o[2])` B `scores.sort(key=lambda p:sum(p))`  
C `scores.sort(key o:o[1]+o[2])` D `scores.sort(key=lambda obj:obj[2])`

### 二. 填空题

1. 列表、元组、字典用不同的括号表达, 它们分别是\_\_\_\_、\_\_\_\_、\_\_\_\_。
2. 列表、元组、字典、集合的函数分别是\_\_\_\_、\_\_\_\_、\_\_\_\_、\_\_\_\_。
3. 已知列表 `L=[5,1,24,65,10,4]`, 现在需要将其中 `24,65` 两项数据删除, 若采用分片实现, 其语句应该是\_\_\_\_\_。

4. 对给定的列表 L, 若采用分片的方法将其逆转, 实现该功能的语句为\_\_\_\_\_。
5. 采用列表表达式将列表 L=[5,1,24,65,10,4]中的偶数值的元素剔除, 形成新的列表的语句为\_\_\_\_\_。
6. 最简单的获取字典 d 的所有键的语句为\_\_\_\_\_。
7. 教师甲和教师乙的学生名单分别存储在集合 S1 和 S2 中, 寻找同时选修两教师课的学生名单的语句为\_\_\_\_\_。
8. 运行并记录结果

X=[1,2,3,5,9,11]

X[-1] 结果: \_\_\_\_\_

X[::-1]结果: \_\_\_\_\_

X[0:7:2]结果: \_\_\_\_\_

9. 定义 X=(1,2,3,5,9,110),运行下面代码记录结果

X[-1] 结果: \_\_\_\_\_

X[::-1]结果: \_\_\_\_\_

X[0:7:2]结果: \_\_\_\_\_

10. 运行并记录结果

L=[12,34,23,12]

L.append(30) 结果: \_\_\_\_\_

L.reverse() L 的值: \_\_\_\_\_

L.sort() L 的值: \_\_\_\_\_

sum(L) 结果: \_\_\_\_\_

len(L)结果: \_\_\_\_\_

L=[1,2,3,4,5]

L1=L

L1[1]=-10

L 的值为: \_\_\_\_\_

list(range(1,10,2))的结果: \_\_\_\_\_

11. 已知学生组 1: s1=set(('沈月月','王华','邓享东','杨丹一','居雷'))和学生组 2: s2=set(('沈月月','王海波','邓享东','杨丹一','陆龙剑')), 请运行下列代码并记录结果:

s1 & s2 的结果: \_\_\_\_\_

s1-s2 的结果: \_\_\_\_\_

s1 | s2 的结果: \_\_\_\_\_

12. 数列 1/2, 2/3, 3/5, 5/8, 8/13, ....., 其特点是第 n 项的分子是第 n-1 项的分母, 第 n 项的分母是第 n-1 项的分子与分母之和, 下面程序以每行 5 个元素的方式, 输出数列的

前 15 项。请将程序补充完整。

```
a=1
b=2
__(1)___
for i in range(15):
    print(str(a)+'/'+str(b),end="t")
    count += 1
    if __(2)__:
        print()
    temp=__(3)___
    a=b
    b=__(4)___
```

13. 以下程序实现购物结算时十万元以下货币金额的大写转换，如输入 4521，将转换为['肆', '千', '伍', '佰', '贰', '拾', '壹', '元']，请将程序补充完整。

```
def convert(num):
    s={'0':'零','1':'壹','2':'贰','3':'参','4':'肆','5':'伍','6':'陆','7':'柒','8':'捌','9':'玖'}
    danwei=['元','拾','佰','千','万']
    num=list(__(1)___)
    num.reverse()
    result=[]
    for i,j in __(2)__:
        result.append(danwei[i])
        result.append(__(3)___)
    __(4)___
    return result
num=int(input("请输入结账金额（由 0-9 的数字组成）："))
resultnum=convert(num)
print(resultnum)
```

### 三. 简答题

1. 列表的元素可以属于不同的数据类型，请讨论为何列表的部分方法虽然存在，但却不一定能使用，如列表的 `sort` 方法就不能随意使用。
2. 列表和元组都是序列数据，请简单讨论回答为何列表的很多方法如 `append`、`sort` 等，元组对象则没有，但两者都有 `index`、`count` 方法。
3. 列表推导式是生成列表的一种方法，对数据过滤很有效，请简述列表推导式的格式，通过何种语法实现数据过滤？
4. 列表、元组在某些状态下可以转换为字典，请列举将列表（或元组）转换为字典的多种方法（提示：列表的形式可以多样，如两个有相同个数元素的列表，或一个列表的每个元素是一个列表等形式）。
5. 请罗列三种字典遍历的方式，遍历时输出每个键-值对的方法。

## 第六章 文件

文件是有名的一组相关信息的集合。在计算机系统中，所有的程序和数据都是以文件的形式存放在计算机的外存储器（如磁盘等）上，操作系统也是以文件为单位对数据进行管理。因此，对程序设计初学者来说，文件基本知识及其读写方法也是必须要掌握的。

### 6.1 概述

本节下面首先通过一个引例说明为什么要使用文件以及如何读写文件的，然后介绍文件的基本知识。

#### 6.1.1 引例——统计学生成绩

**【例 6.1】**输入一批学生的学号、姓名和成绩，统计学生人数、总成绩和平均成绩。

方法 1：将读入的学生数据存储在列表中。

```
# 学生数据存储在列表中
stuList=[]
n=int(input("请学生人数: "))
for i in range(n):
    No=input("请输入学号: ")
    Name=input("请输入姓名: ")
    Mark=int(input("请输入成绩: "))
    stuList.append([No,Name,Mark])
Total=0
for i in range(n):
    Total=Total+stuList[i][2]
Average=Total/n
print("人数: ",n)
print("总成绩",Total)
print("平均成绩",Average)
```

这样的解决方案存在一个缺点：每次统计学生成绩都必须输入所有学生的数据。因为程序一旦运行结束，列表所占有的内存空间被释放了，列表中存储的学生数据也就丢失。若要永久保存，学生成绩可以保存在文件中，因为文件能在外存储器上长期保存。

方法 2：将读入的学生数据存储在如图 1.6.1 所示的 Scores.txt 文件中。



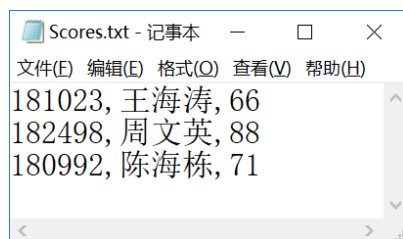


图 1.6.1 学生成绩文件

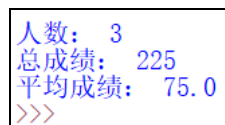


图 1.6.2 学生成绩统计结果

```
n=int(input("请学生人数: "))
fScores=open(r"D:\Test\Scores.txt","w")
for i in range(n):
    No=input("请输入学号: ")
    Name=input("请输入姓名: ")
    Mark=int(input("请输入成绩: "))
    fScores.write(No + "," + Name + "," + str(Mark) + "\n")
fScores.close()
```

# 打开文件供写数据  
# 输入三个同学的成绩  
# 输入学号  
# 输入姓名  
# 输入成绩  
# 写入文件  
# 关闭文件

若要统计文件中的成绩，则打开文件，然后读出数据，再进行处理，最后关闭文件。

```
Statistics=open(r"D:\Test\Scores.txt","r")
n=0
Total=0
while True:
    Student=Statistics.readline()
    if not Student:
        break
    n=n+1
    Temp=Student.split(",")
    Mark=int(Temp[2])
    Total=Total+Mark
Average=Total/n
print("人数: ",n)
print("总成绩: ",Total)
print("平均成绩: ",Average)
Statistics.close()
```

# 打开文件供读数据  
# 变量 n 用于统计人数  
# 变量 Total 用于统计总成绩  
# 循环读数据  
# 读一行，即读一个学生数据  
# 判断数据是不是读完  
# 若数据读完则退出循环  
# 学生人数加 1  
# 根据 “,” 分隔符切分数据成列表  
# 取出成绩  
# 累加成绩  
# 计算平均成绩  
# 关闭文件

图 1.6.2 是对如图 1.6.1 所示的含有三个学生数据的文件进行处理的结果。比较两种方法可以得出结论：数据若要长久保存供重复使用，则可以保存在文件中。

## 6.1.2 文件和文件路径

一个磁盘上的文件成千上万，为了有效地管理和使用文件，通常将磁盘上所有文件组织成树状结构，也就是在磁盘上创建文件夹（目录），在文件夹下再创建子文件夹（子目录），然后将文件分门别类地存放在不同的文件夹中，如图 1.6.3 所示。这种结构像一棵倒置的树，树根为根文件夹（根目录），树中每一个分枝为文件夹（子目录），树叶为文件。在树状结构中，用户可以将一个项目的文件放在一个文件夹中，也可以按文件类型或用途将文件分类存放。

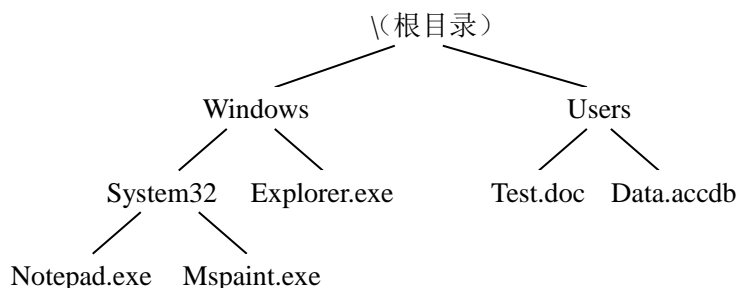


图 1.6.3 树型目录结构

当一个磁盘的目录结构被建立后，所有的文件可以分门别类地存放在所属的文件夹中，接下来的问题是如何访问这些文件。若要访问某个文件，程序中必须描述清楚文件的位置，即给出文件路径，以便操作系统可以查找到所需要的文件。

文件路径分有两种：

① 绝对路径：从根目录开始到某个文件的路径。例如，Notepad.exe 和 Test.doc 文件的绝对路径分别为 C:\Windows\System32\Notepad.exe 或和 C:\Users\Test.doc。

② 相对路径：从当前目录开始到某个文件的路径。如果当前目录为 System32，则 Data.accdb 文件的相对路径为..\..\User\Data.accdb（用“..”表示上一级目录）。

在路径中，文件夹名称之间的分隔符可以是“\”，也可以是“/”。因为 Windows 中使用“/”作为分隔符，但是 MS DOS 中使用“\”，为了支持 MS DOS 的使用习惯，所以也支持“\”。但是 Python 中“\”表示转义字符的开始，所以表示路径的字符串应该避免出现歧义。下面是三种正确的用法。以 C:\Users\Test.doc 为例。

- ◆ 用“/”作为分隔符，如"C:/Users/Test.doc"。
- ◆ 用“\\”表示一个“\”，例如"C:\\Users\\Test.doc"。
- ◆ 字符串之前用“r”或“R”可以取消“\”的转义功能，成为一个普通字符，如r"C:\Users\Test.doc"。

### 6.1.3 文件分类

在计算机系统中，文件种类繁多，处理方法和用途也各不相同。文件的分类标准主要有下列两种：

1. 按文件的内容分类，可分为程序文件和数据文件。程序文件存储的是程序，包括源程序和可执行程序，例如 Python 的源程序文件（.py）、C++源程序文件（.cpp）、可执行程序文件（.exe）等都是程序文件。数据文件存储的是程序运行所需要的各种数据，例如文本文件（.txt）、Excel 工作簿（.xlsx）都是数据文件。

2. 按存储信息的形式分类，可分为文本文件和二进制文件。文本文件存放的是各种数据的 ASCII 代码，可以用记事本打开；二进制文件存放的是各种数据的二进制编码，不能用记事本打开，必须由专用程序打开。

文本文件具有行结构，如图 1.6.4 所示。每一行的结束都有换行符“\n”，图中用“↵”表示。因此，在 Python 中对文本文件的读写通常采用两种方式：

- 一次性地读写文件的所有内容。
- 以行为单位，一行一行地按顺序进行读写。也就是说，读出时从头到尾按顺序一行一行地读，写入时也一样。

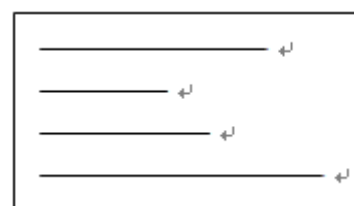


图 1.6.4 顺序文件的结构

## 6.1.4 文本文件编码

文本文件中的字符都是以某种类型编码存储的。打开文件时应指定编码类型，否则读写时会出现错误。例如，假定 SHL.TXT 文件中含有汉字，编码类型是 UTF8，若执行下面语句：

```
fp=open(r"C:\Test\SHL.TXT","r")
s=fp.read()
print(s)
```

则出现下列错误信息：

UnicodeDecodeError: 'gbk' codec can't decode byte 0xa4 in position 4: illegal multibyte sequence

这是 Unicode 解码错误。因为 SHL.TXT 文件的编码类型是 UTF8，而语句 open 函数打开文件时没有指定文件编码类型，默认为 ANSI，所以出现了错误。那么，如何文件的编码类型有哪几种类型？如何确定文件的编码类型？

用记事本打文本文件，选择“文件 | 另存为”命令，在对话框下面的“编码”组合框显示当前文件采用的编码类型，打开组合框可以看到所有支持的编码类型，如图 1.6.5 所示。若要转换文件的编码类型，则使用“文件 | 另存为”命令，选择所需要的编码类型。

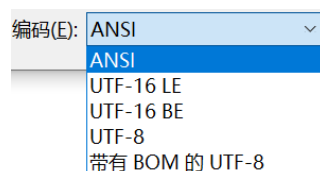


图 1.6.5 记事本“另存为”对话框中“编码”组合框

下面介绍三种常用的编码：

(1) ANSI 中文 Windows 中，西文字符采用 ASCII 码，占一个字节；中文字符采用 GBK 码，占两个字节。例如，“2021 上海 SH”，ANSI 编码（十六进制形式）为：

编码：	32	30	32	31	C9	CF	BA	A3	53	48
字符：	2	0	2	1	上	海	S	H		

(2) UTF-16 LE 针对 Unicode 的一种字符编码。它的特点是所有字符均占两个字节，低字节在前高字节在后。例如，字符“A”的 ASCII 值为 41H，UTF-16 LE 编码是 41 00H，；中文字符的 UTF-16 LE 码与 GBK 没有关系。例如，“2021 上海 SH”，Unicode 编码（十六进制形式）为：FF FE 32 00 30 00 32 00 31 00 0A 4E 77 6D 53 00 48 00。

编码: 32 00 30 00 32 00 31 00 0A 4E 77 6D 53 00 48 00  
 字符:    2      0      2      1      上      海      S      H

其中: 文件开始两个字节“FF FE”表示编码类型为 UTF-16 LE;

(3) UTF-8 针对 Unicode 的一种可变长度字符编码。它使用 1~4 字节为每个字符编码, ASCII 字符占 1 个字节, 中文字符占 3 个字节。例如, “2021 上海 SH”, UTF-8 编码(十六进制形式)为:

编码: 32 30 32 31 E4 B8 8A E6 B5 b7 53 48  
 字符: 2      0      2      1      上              海      S      H

打开文件时, 一般都通过 encoding 参数指定文件的编码类型。encoding 参数的取值见表 1.6.1 所示。

表 1.6.1 encoding 参数的常用值

编码类型	encoding 对应的值 (不区分大小写)
ANSI	MBCS、ANSI、GB2312、GBK
UTF-16LE	UTF-16LE
UTF-8	UTF、UFT8、U8、UTF-8、UTF_8

## 6.1.5 文件访问流程

一般来说, 访问文件流程分三个步骤: 打开文件、读写文件、关闭文件, 如图 1.6.6 所示。

对文件的操作主要有两类: 一是读操作, 也称为输入, 将数据从文件(存放在外存上)读入到变量(内存)供程序使用; 二是写操作, 也称为输出, 将数据从变量(内存)写入文件(存放在外存上)。

打开文件时, 系统为文件在内存中开辟了一个专门的数据存储区域, 称为文件缓冲区。将数据写入文件时, 先将数据写入文件缓冲区暂存, 等到文件缓冲区满了或文件关闭时才输出到文件。反之, 从文件读数据时, 先将数据送到文件缓冲区, 然后再提交给变量, 如图 1.6.7 所示。这样处理的目的是为了减少直接读写外存的次数, 节省操作时间。

文件操作结束后一定要关闭文件, 因为有部分数据仍然在文件缓冲区, 若不关闭文件会有数据丢失现象发生, 尽管大多数情况下操作系统会自动关闭文件。



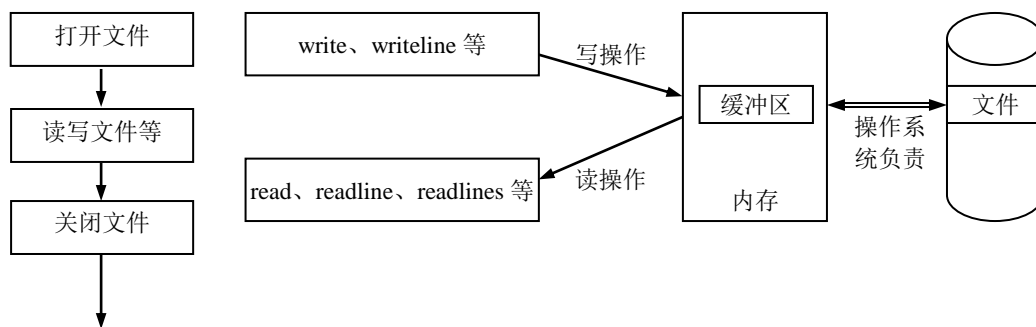


图 1.6.6 文件处理一般流程

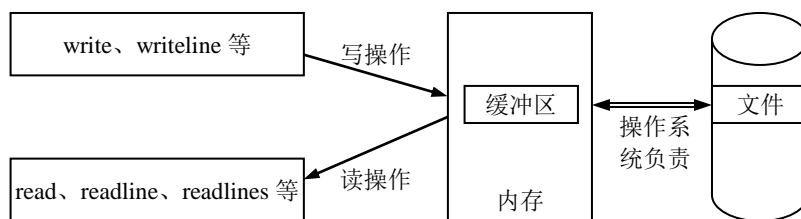


图 1.6.7 文本文件的读写与文件缓冲区

## 6.2 文本文件

在 Python 中，文本文件的读写有三个特点：

- ① 读写文件时，可以一次性读写文件所有内容，也可以行为单位进行读写。
- ② 以行为单位进行读写的规则也简单，就是按顺序进行访问。读的时候从头到尾一行一行地按顺序读，写入时也一样，不可以跳过前面的数据而直接读写某一行数据。
- ③ 写文件时，各种类型的数据都需要转换成字符串后才能写入文件。

### 6.2.1 文本文件的打开与关闭

#### 1. 打开文件

在对文件进行操作之前，必须打开文件，同时通知操作系统对文件所进行的操作是“读”还是“写”。打开文件的函数是 `open`，其常用形式如下：

```
fp = open(file, mode='r', encoding='utf-8')
```

说明：

① `file` 是要读写的文件名，可以是字符串常量，也可以是字符串变量。如 `"D:/Test/Scores.txt"`、`"D:\\Test\\Scores.txt"`或 `r"D:\\Test\\Scores.txt"`。

② `mode` 指明打开文件的模式，是可选参数，其值如表 1.6.2 所示。

表 1.6.2 mode 参数

mode	作用	说明
'r'	读模式	文件必须存在
'w'	写模式	若文件不存在，则创建文件；若文件存在，则清空文件内容
'x'	创建文件	若文件已经存在，则失败
'a'	追加模式	若文件存在，则向文件的末尾追加内容； 若文件不存在，则创建新文件
'b'	二进制模式	可以与其他模式结合使用
't'	文本模式（默认）	可以与其他模式结合使用
'+'	读 / 写模式	可以与其他模式结合使用

例如，如果要打开 D:\Python 目录下一个文件名为 Students.txt 文件，供写入数据，则语句应为：

```
fp = open(r"D:\Python\Students.txt", "w")
```

③ encoding 指定文件的字符编码方式，缺省值是"utf-8"，常用值见表 1.6.1。

④ open 函数的返回值是文件对象，文件的读写和关闭都需要文件对象的方法实现的。

⑤ 文件对象内置了一个迭代器，因此可以把文件对象当作由文件所有行所组成的列表一样遍历文件。例如，

```
f=open("C:\\Test\\t1.txt","r")
for line in f:
    print(line, end="")
f.close()
```

## 2. 关闭文件

当结束各种读写操作以后，还必须关闭文件，否则会造成数据丢失等现象。因为通常数据是被送到缓冲区，关闭文件时才将缓冲区中数据全部写入文件。关闭文件所用的方法是 close，其使用形式如下：

```
文件对象.close()
```

例如，语句 fp.close() 关闭 fp 文件对象所代表的文件。

## 6.2.2 文件文件的读写

### 1. 读文件

从文本文件中读出数据常用的方法有 read、readline、readlines。

#### (1) read 方法

功能：将文件的所有内容作为一个字符串一次性读出。

常用格式：变量=文件对象.read()

说明：read 方法的返回值是字符串，它包含文件所有的内容。

例如，下列程序将 D:\Test\Scores.txt 读出并且输出。

```
fp=open("D:\Test\Scores.txt","r")
all=fp.read()
print(all)
```

#### (2) readline 方法

功能：从文本文件中读出一行数据，并将它作为函数的返回值。

常用格式：变量=文件对象.readline()

说明：readline 的返回值是一字符串，字符串的内容是文件的一行，包含换行符。

例如，下列程序将 D:\Test\Scores.txt 读出并且输出。

```
fp=open(r"D:\Test\Scores.txt","r")
while True:
    str=fp.readline()
    if not str:
```

```
break
print(str, end="")
fp.close()
```

### (3) readlines 方法

功能：把文件中的多行数据一次性读入一字符串列表。

常用格式：变量=文件对象.readlines()

格式：变量=fp.readlines()

说明：readlines 的返回值是一列表，列表的元素是文件的一行，包含换行符。

例如，读取并且输出文本文件 Scores.txt 中的内容。

```
fp=open(r"D:\Test\Scores.txt","r")
lines=fp.readlines()
print(lines)
fp.close()
```

程序运行结果为：['181023,王海涛,66\n', '182498,周文英,88\n', '180992,陈建栋,71\n']

## 2. 写文件

将数据写入文本文件常用的方法是 write、writelines。

### (1) write

格式：文件对象.write(字符串)

说明：

① 一次只能写一个字符串。若要写入多个数据，只能使用多个语句。例如，语句 fp.write("185125","徐文静","88")是错误的，应写成：

```
fp.write("185125")
fp.write("徐文静")
fp.write("88")
```

② 参数只能是字符串，使用其他类型的数据均会出错。例如，fp.write(88)是错误的，而应写成 fp.write("88")

③ 写数据时不会自动换行，若要换行，应添加换行符："\n"。例如，fp.write("88\n")

④ 尽管写的是字符串，但是不像其他程序语言一样写的字符串自动添加双引号。

```
fp=open(r"D:\Test\Sushi.txt","w")
lines=["竹外桃花三两枝，","春江水暖鸭先知。","蒌蒿满地芦芽短，","正是河豚欲上时。"]
for line in lines:
    fp.write(line+"\n")
fp.close()
```

### (2) writelines 方法

功能：将存放在列表中的多行数据一次性写入文本文件

格式：文件对象.writelines(字符串列表)

说明：若要换行，则应在列表元素的末尾添加换行符。

```
fp=open(r"D:\Test\Sushi.txt","w")
lines=["竹外桃花三两枝，\n","春江水暖鸭先知。\n","蒌蒿满地芦芽短，\n","正是河豚欲上时。"]
fp.writelines(lines)
```

```
fp.close()
```

程序运行结果如图 1.6.8 所示。

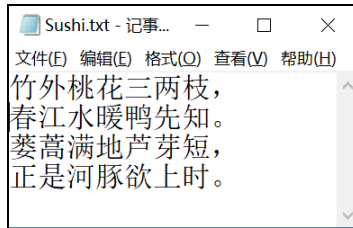


图 1.6.8 writelines 运行结果

【例 6.2】用 readlines 方法读出例 6.1 中文件的数据并且实现统计功能。

```
Statistics=open(r"C:\Test\Scores.txt","r")           # 打开文件供读数据
Total=0                                              # 变量 Total 用于统计总成绩
Students=Statistics.readlines()                    # 一次读出所有的行
n=len(Students)                                    # 统计人数
for item in Students:                              # 对列表 Students 元素进行循环处理
    Temp=item.split(",")                           # 根据 “,” 分隔符切分一行的数据
    Mark=int(Temp[2])                              # 取出成绩
    Total=Total+Mark                               # 累加成绩
Average=Total/len(Students)                        # 计算平均成绩
print("人数: ",n)
print("总成绩",Total)
print("平均成绩",Average)
Statistics.close()                                 # 关闭文件
```

## 6.2.4 异常处理

在文件处理过程中，经常会出现各种各样错误或异常，例如，以“r”模式打开一个不存在的文件。为了不影响程序的正常运行，避免因错误或异常导致程序中止运行，程序中可以使用 try...except...语句进行异常处理。

需要读者注意的是，错误和异常导致的结果往往相似，但是两个不同的概念。一般来说，错误是指语法错误和逻辑错误，而异常是指一个事件，该事件会在程序运行过程发生，从而影响程序的正常执行。不管是错误还是异常，发生时程序都需要捕获和处理它，否则程序会终止运行。

### 1. try...except...简单形式

try...except...语句的简单使用形式为：

```
try:
    <语句块 1>
except <异常类型>:
    <语句块 2>...
```

使用 try...except...语句的关键是了解异常的类型，不知道异常类型就无法使用 try...except...语句。了解异常类型的方法是在调试时人为制造异常，从提示信息中可以发现



异常类型。例如，假定 D:\Test\Data.txt 文件不存在，则执行 `fp=open(r"D:\Test\Data.txt","r")` 语句时会显示：

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    fp=open(r"D:\Test\Data.txt","r")
FileNotFoundError: [Errno 2] No such file or directory: 'C:\\Test\\Data.txt'
```

其中指出异常类型为“`FileNotFoundError`”。

【例 6.3】为例 6.2 设置异常处理。

```
n=int(input("请学生人数: "))
try:
    fScores=open(r"D:\Test\Scores.txt","w")           # 打开文件供写数据
    for i in range(n):                                # 输入三个同学的成绩
        No=input("请输入学号: ")                     # 输入学号
        Name=input("请输入姓名: ")                   # 输入姓名
        Mark=int(input("请输入成绩: "))               # 输入成绩
        fScores.write(No + "," + Name + "," + str(Mark) + "\n") # 写入文件
    fScores.close()                                    # 关闭文件
except FileNotFoundError:
    print("文件不存在")
```

【例 6.4】分班程序：n 个同学分成若干个班，一个班有 k 个同学。k 是键盘输入的，若输入的是 0，则程序会终止运行。添加异常处理语句处理除数为 0 的情况。

```
try:
    n=int(input("请输入学生人数: "))
    k=int(input("请输入班级人数: "))
    if n%k==0:
        m=n/k
    else:
        m=n/k+1
    print("班级数=",m)
except ZeroDivisionError:
    print("输入的班级人数为零")
```

## 2. try...except...复杂形式

一个语句块可能会发生多种错误或异常，此时需要分门别类捕捉和处理，就应使用 `try...except...` 语句的复杂形式，其语法格式如下：

```
try:
    <语句块 0>
except <异常类型 1>:
    <语句块 1>
except <异常类型 2>:
    <语句块 2>
.....
except <异常类型 n>:
    <语句块 n>
except:
    <语句块 n+1>
```

```
[else:
    <语句块 n+2>]

[finally:
    <语句块 n+3>]
```

说明：

① 这种形式的异常处理语句指定了各种异常发生的情况下所执行的语句块，最后一个 `except` 子句没有指定异常类型，表示处理剩下的异常类型。

② `else` 后的语句块是对 `try` 后语句块正常执行后的一种追加处理，与 `for` 和 `while` 语句中的 `else` 子句相同。

③ 不论会不会发生异常，`finally` 后的语句块都是执行，一般是将执行<语句块 1>的一些收尾工作放在这里。

**【例 6.5】** 修改例 6.4，使之还能够处理 `n` 和 `k` 是非整数的情况。

```
n 个同学分成若干个班，一个班中个同学，
try:
    n=int(input("请输入学生人数："))
    k=int(input("请输入班级人数："))
    if n%k==0:
        m=n/k
    else
        m=n/k+1
    print("班级数=",m)
except ValueError:
    print("输入了非整数")
except ZeroDivisionError:
    print("输入的班级人数为零")
except:
    print("其他错误")
```

显然，使用 `try...except...` 语句可以提高程序的稳定性和可靠性，但是也不能过度使用。因为过度使用会降低程序的可读性，增加维护的难度。因此，一般来说，均在关键的少数地方使用 `try...except...` 语句。

## 6.2.4 CSV 格式

CSV（Comma-Separated Values，逗号分隔值）格式是一种常用的文本格式，其特点是存储表格数据时用逗号作为分隔符（也可以使用其他字符），如图 1.6.9 所示。CSV 格式应用非常广泛，常用于程序之间传递表格数据。目前很多应用软件都支持 CSV 格式。例如，使用 MS Excel 可以打开 CSV 文件，也可以另存为 CSV 文件，但是另存为 CSV 文件后其他数据格式信息全部丢失。

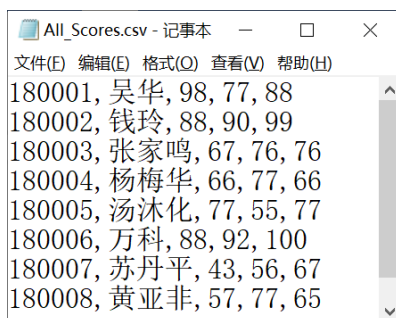


图 1.6.9 CSV 格式文件

因为直到 2005 年 CSV 才有标准，所以难以统一，多种 CSV 格式都在使用，它们自成体系，相互不兼容，导致处理 CSV 文件时会遇到麻烦。对于 Python 程序员来说，最好有一套自己的标准并且遵循，这样才不会犯低级错误。

CSV 文件具有下列特点：

- ① 纯文本文件，扩展名为.csv
- ② 像普通文本文件一样具有行结构，没有空行。
- ③ 数据之间通常用逗号作为分隔符，但是也可以不用逗号，因此 CSV 也称字符分隔值。
- ④ 字符串数据没有双引号

尽管 Python 提供了 CSV 库可以使用，但是因为 CSV 格式非常简单，程序员可以自己编写程序或函数，这样更显灵活和个性化。

【例 6.6】图 1.6.7 所示的 CSV 文件一行的数据由学号、姓名以及数学、外语、计算机三门课程成绩组成。读出文件中的数据，统计所有同学的平均总成绩。

```
fp=open(r"D:\Test\All_Scores.csv","r")
lines=fp.readlines()
Total=0
Count=0
for line in lines:
    temp=line.split(",")
    Total=Total+int(temp[2])+int(temp[3])+int(temp[4])
    Total=Total+1
    Count=Count+1
Average=Total/Count
print("平均总成绩",Average)
```

## 6.3 jieba 中文分词

中文分词是指将一个汉字序列切分成一个个独立的词。在英文中，单词之间是以空格、标点符号作为自然分界符的，使用字符串的 `split` 方法就可以完成分词；在中文中，词与词之间没有一个形式上的分界符，分词是一件非常困难的事情。然而中文分词又是非常重要的，是中文信息处理的基础，因而需要使用 `jieba` 库。

`jieba` 是一个用于中文分词的第三方库，使用之前必须安装。安装命令的形式为：

```
pip install jieba
```

## 1. 中文分词模式

jieba 库的中文分词模式有三种：精简模式、全模式、搜索引擎模式。

### ① 精简模式

精准模式尝试将句子最精确地分词，适合文本分析。精简模式通常使用 lcut 方法实现，其返回值是列表。例如：

```
import jieba
lst=jieba.lcut('小明本科毕业于同济大学，后在耶鲁大学深造')
print(lst)
```

程序运行结果为：['小明', '本科毕业', '于', '同济大学', ',', '后', '在', '耶鲁大学', '深造']

### ② 全模式

全模式把句子中所有可以成词的词语都扫描出来，速度非常快，但其分词得到的结果有可能有歧义。精简模式通常也是使用 lcut 方法实现，其返回值也是列表，但是指定形式参数 cut\_all 的值为 True。例如：

```
import jieba
lst=jieba.lcut('小明本科毕业于同济大学，后在耶鲁大学深造',cut_all=True)
print(lst)
```

程序运行结果为：['小', '明', '本科', '本科毕业', '毕业', '于', '同济', '同济大学', '大学', ',', '后', '在', '耶鲁', '耶鲁大学', '大学', '深造']

从输出结果可以可以发现，全模式比精确模式得到的词的组合多了很多。

### ③ 搜索引擎模式

在精确模式的基础上，对长词再次切分，提高召回率，适合于搜索引擎分词。搜索引擎模式通常使用 lcut\_for\_search 方法实现，其返回值是列表。例如：

```
import jieba
list1=jieba.lcut_for_search('小明本科毕业于同济大学，后在耶鲁大学深造')
print(list1)
```

程序运行结果为：['小明', '本科', '毕业', '本科毕业', '于', '同济', '大学', '同济大学', ',', '后', '在', '耶鲁', '大学', '耶鲁大学', '深造']

**【例 5.21】**统计红楼梦小说中主要人物的出现次数。

分析：首先是读入红楼梦.txt 文件，是需要将读入的文本进行分词，然后进行中文分词，最后对主要人物进行统计，统计结果存储在字典中。由此程序代码如下：

```
import jieba
file = open(r"C:\Test\红楼梦.txt", "r", encoding='utf-8')
txt=file.read() #读文件
wordsList=jieba.lcut(txt) # 分词结果
actors=[('贾宝玉',"宝玉"),("林黛玉","黛玉"),("薛宝钗","宝钗"),
        ("王熙凤","凤姐"),("贾母","老太太"),("袭人"),("探春"),
        ('贾琏'),('王夫人',"夫人")] # actors 是人物列表，每个人物使用了元组
dictActors={} # 拟建立的字典
```

```

for actor in actors:                                #从人物表中查分词
    if len(actor)==2:
        count1=wordsList.count(actor[0])
        count2=wordsList.count(actor[1])
        dictActors[actor[0]]=count1+count2
    else:
        count1=wordsList.count(actor[0])
        dictActors[actor[0]]=count1+count2

items = list(dictActors.items())                    # 将字典表项转化成列表
items.sort(key=lambda x:x[1], reverse=True)        # 按频次从大到小排序
for i in range(len(items)):                        # 输出统计结果
    word, count = items[i]
    print("{}\t{}".format(word, count))

```

程序的最终运行结果如下：

贾宝玉	3665
贾母	2091
王熙凤	1201
王夫人	1110
林黛玉	697
贾琏	659
袭人	580
薛宝钗	572
探春	420

## 2. 文本关键词抽取

使用 `jieba.analyse` 中的 `extract_tags` 方法可以提取文本中出现的关键词，其使用格式为：

```
extract_tags(文本, topK=20, withWeight=False, allowPOS=( ))
```

说明：

- ① `topK` 参数指定按权重提取的关键字个数，默认 20。
- ② `withWeight` 指定是否返回关键字的权重。

下面的代码获取了红楼梦小说中权重最大的 5 个关键词。

```

import jieba.analyse as analyse
file = open(r"F:\teach\python 语言\例子数据\红楼梦.txt", "r", encoding='utf-8')
txt=file.read()
lists=analyse.extract_tags(txt, topK=5, withWeight=False, allowPOS=( ))
print(lists)

```

程序运行结果为：['宝玉', '凤姐', '贾母', '王夫人', '老太太']

## 6.4 图像文件

图像文件是典型的二进制文件。Python 中常用的图像处理库是 PIL (Python Image Library)。它拥有 21 个处理图像的类，如 `Image`、`ImageFilter`、`ImageEnhance` 等，支持图像

的存储（格式转换）、处理、显示等功能。

PIL 是 Python 的第三方库，使用之前必须安装。安装库的名字是 `pillow`，安装命令为：

```
pip install pillow
```

## 6.4.1 图像基本处理

`Image` 是 PIL 中最重要的类，可以用来实现图像处理的所有基本功能。Python 处理图像是从导入 `Image` 开始的。下面首先通过一个例子说明使用 PIL 包处理图像的基本过程，然后再介绍 `Image` 类。

**【例 6.7】** 将彩色图像转换成黑白图像

```
from PIL import Image                                # 导入 PIL 库的 Image 模块
im = Image.open(r"D:\Test\Tulips.jpg")                # 使用 open 方法打开图像文件，返回图像对象
im.show()                                             # 显示图像对象 im 中的图像
im = im.convert("L")                                 # 转换为灰度 8 位图像
im.show()                                             # 显示图像
im.save(r"D:\Test\New-Tulips.jpg")                   # 保存图像文件
```

程序运行结果如图 1.6.10 所示。

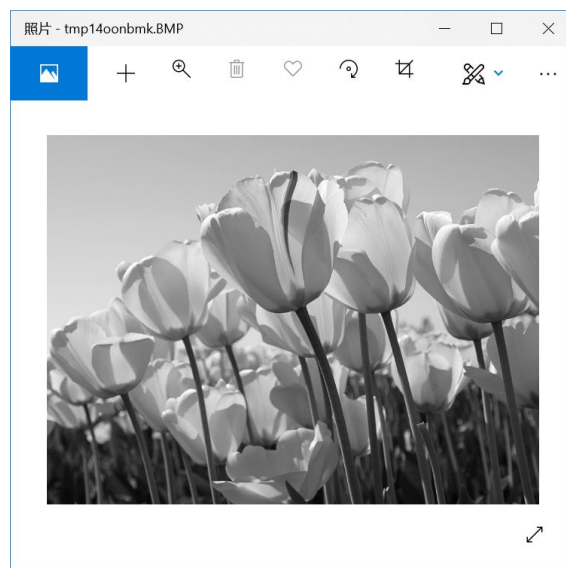


图 1.6.10 Tulips.jpg 转换后的效果

从上述例子可以看到，PIL 处理图像的基本过程是：

- ① 导入 PIL 库的 `Image` 等模块
  - ② 使用 `open` 方法打开图像文件，返回图像对象，以后通过图像对象实现图像处理的。
  - ③ 图像显示或处理
  - ④ 若需要，则保存图像
1. 图像打开、显示、转换和保存
    - (1) 图像打开

`Image` 类的 `Open` 用于打开图像文件，返回一个 `Image` 对象，其使用形式为：

变量=open(文件名)

说明:

① open 的返回值一个 Image 对象，图像文件打开后所有的操作都是通过 Image 对象实现的;

② Image 对象具有 size、format、mode 等属性。

- size 属性 一个元组，由图像的宽度和高度两个数据组成;
- format 属性 图像文件格式，常见的包括 JPEG、PNG、GIF 等格式;
- mode 属性 颜色模式，如表 1.6.3 所示等。

表 1.6.3 图像颜色模式

格式	说明
1	1 位像素，表示黑和白，但是存储的时候每个像素仍然为 8bit
L	8 位像素，黑白图像
P	8 位像素，使用调色板映射到其他模式
RGB	3x8 位像素，真彩色
RGBA	4x8 位像素，有透明通道的真彩色
CMYK	4x8 位像素，印刷四色模式
YCbCr	3x8 位像素，彩色视频格式
I	32 位整型像素
F	32 位浮点型像素

例如，下列代码显示 Tulips.jpg 图像的 size、mode 和 format 属性

```
from PIL import Image
im = Image.open(r"D:\Test\Tulips.jpg")
print(im.size)
print(im.mode)
print(im.format)
```

程序结果如下:

```
(1024, 768)
RGB
JPEG
```

(2) 图像显示

Image 对象的 show 方法用于显示图像，其使用形式为:

图像对象.show()

(3) 保存图像

Image 对象的 save 方法用于保存图像，其使用形式为:

图像对象.save(文件名, 格式)

说明: save 方法的格式参数表示以何种图像格式存储，其值与 Image 对象的 format 属性相同，但使用时应加引用，如"JPEG"。若缺省则根据文件扩展名表示的格式存储。

#### (4) 转换图像

Image 对象的 `convert` 方法用于转换图像格式，其使用形式为：

图像对象.`convert`(格式)

说明：图像格式如表 1.6.3 所示。

#### (5) 获取指定位置的像素值

Image 对象的 `getpixel` 方法用于获取指定位置的像素值。如果图像为多通道，则返回一个元组。其使用形式为：

图像对象.`getpixel`((x,y))

例如，下列代码显示 Tulips.jpg 图像中位置 (100,80) 像素值。

```
from PIL import Image # 导入 PIL 库的 Image 模块
im = Image.open(r"D:\Test\Tulips.jpg") # 使用 open 方法打开图像文件，返回图像对象
print(im.getpixel((100,80)))
```

程序结果如下：

(156, 204, 253)

表示位置 (100,80) 像素的红绿蓝三种颜色的值分别为 156、204、253。

#### (6) 处理像素

Image 对象的 `point` 方法用于处理图像的像素。其使用形式为：

图像对象.`point`(函数)

例如，下列代码首先将 Tulips.jpg 转换成 8 位灰度图像，然后将灰度值 127 以上的转换成 255 (白)，127 以下的转换成 0 (黑)。结果如图 1.6.11 所示。

```
from PIL import Image
im = Image.open(r"D:\Test\Tulips.jpg")
im = im.convert("L")
im = im.point(lambda x: 255 if x > 127 else 0)
im.show()
```

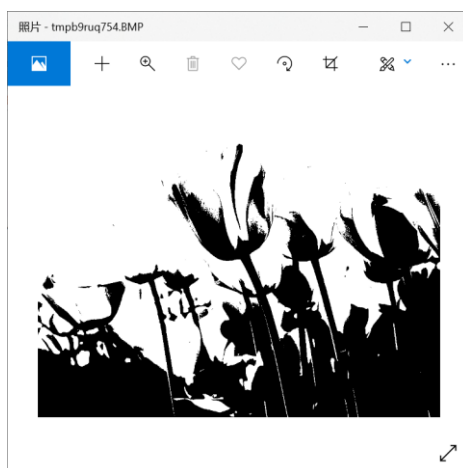


图 1.6.11 用 point 方法处理后的黑白图像

## 2. 缩略图、缩放、旋转、翻转

Image 类可以制作缩略图，也可缩放、旋转、翻转图像，所用方法如表 1.6.4 所示。

表 1.6.4 图像的基本操作方法



功能	方法	示例	说明
缩略图	thumbnail	<code>img2=img1.thumbnail((128,128))</code>	制作 128×128 的缩略图
缩放	resize	<code>img2=img1.resize((200,200))</code>	将图像缩放到 200×200
旋转	rotate	<code>img2 = img1.rotate(-45)</code>	顺时针旋转 45 度 若数值为正，则逆时针旋转
翻转	transpose	<code>img2=img1.transpose(Image.FLIP_LEFT_RIGHT)</code>	左右翻转
		<code>img2=img1.transpose(Image.FLIP_TOP_BOTTOM)</code>	上下翻转
		<code>img2=img1.transpose(Image.ROTATE_90)</code>	逆时针 90
		<code>img2= img1.transpose(Image.ROTATE_180)</code>	逆时针 180
		<code>img2=img1.transpose(Image.ROTATE_270)</code>	逆时针 270

【例 6.8】将图像左右翻转。

```
from PIL import Image
img1 = Image.open(r"D:\Test\Tulips.jpg")
img2 = img1.transpose(Image.FLIP_LEFT_RIGHT)    #左右翻转
img2.save("New_Tulips.jpg")
```

### 3. 色彩通道的分离和合并

色彩通道的分离是指将 RGB 图像的 3 个颜色通道分离开来，生成三个列表，所有的方法是 `split`，常用使用形式为：

```
r,g,b=im.split()
```

色彩通道的合并是分离的逆操作，即将通过 3 个颜色通道合并成一个新的图像，所有的方法是 `imerge`，常用使用形式为：

```
om=Image.merge("RGB",(r,g,b))
```

【例 6.9】将图像中的红色和蓝色两个颜色通道进行交换。

```
from PIL import Image
im = Image.open(r"D:\Test\birdnest.jpg")
im.show()
r,g,b=im.split()
om=Image.merge("RGB",(b,g,r))
om.show()
```

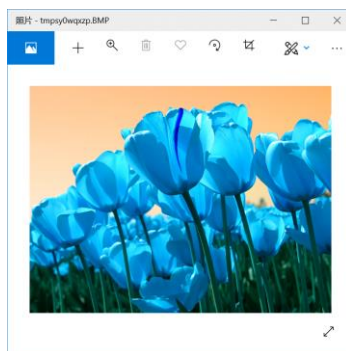


图 1.6.12 红、蓝两个颜色通道交换后的图像

# 6.4.2 图像的滤波和增强

## 1. 图像滤波

在图像处理中，经常需要对图像进行平滑、锐化、边界增强等滤波处理。在使用 PIL 图像处理库时，可以通过 Image 类中的成员函数 filter()来调用滤波函数对图像进行滤波，而滤波函数则通过 ImageFilter 类来定义的。filter()函数的使用形式为：

im.filter(滤波函数)

其中，常用的滤波函数如表 1.6.5 所示。

表 1.6.5 常用的滤波函数

滤波函数	滤波效果说明
ImageFilter.BLUR	模糊效果
ImageFilter.CONTOUR	轮廓效果
ImageFilter.DETAIL	细节效果，即图像变得清晰
ImageFilter.EDGE_ENHANCE	边界增强效果
ImageFilter.EMBOSS	灰色浮雕效果
ImageFilter.FIND_EDGES	边界效果，即寻找边界滤波
ImageFilter.SMOOTH	平滑效果
ImageFilter.SHARPEN	锐化效果

【例 6.10】图像浮雕处理。

```
from PIL import Image
from PIL import ImageFilter as IF
im = Image.open(r"D:\Test\Tulips.jpg")
im1 = im.filter(IF.EMBOSS)
im1.show()
```

程序结果如图 1.6.13 所示。

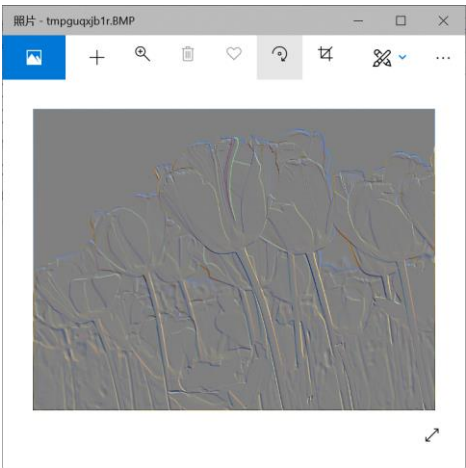


图 1.6.13 图像浮雕效果

## 2. 图像增强

PIL 模块中的 ImageEnhance 类专门用于图像的增强处理，可以增强（或减弱）图像的亮度、对比度、色度、锐度等。

`ImageEnhance` 类中所有功能都是一个类（派生类），所以用法与一般模块不同。例如，要使得某个图像亮度增加一倍，则应首先用 `im` 实例化一个 `ImageEnhance.Brightness` 对象（假定被变量 `en` 引用），然后 `en` 调用 `enhance` 方法增强图像，返回值是增强后的图像对象（假定被变量 `im1` 引用），最后 `im1` 调用 `show` 方法显示增强后的图像。代码如下：

```
from PIL import Image
from PIL import ImageEnhance
im=Image.open(r"D:\Test\Tulips.jpg")
en=ImageEnhance.Brightness(im)
im1=en.enhance(2)
im1.show()
```

`ImageEnhance` 的图像增强功能对应所用的类以及用法如表 1.6.6 所示。

表 1.6.6 `ImageEnhance` 的图像增强方法

功能	所用的类	用法示例
亮度增强	<code>ImageEnhance.Brightness</code>	<code>en=ImageEnhance.Brightness(im)</code> <code>im1=en.enhance(2)</code> <code>im1.show()</code>
色度增强	<code>ImageEnhance.Color</code>	<code>en=ImageEnhance.Brightness(im)</code> <code>im1=en.enhance(2)</code> <code>im1.show()</code>
对比度增强	<code>ImageEnhance.Contrast</code>	<code>en=ImageEnhance.Brightness(im)</code> <code>im1=en.enhance(2)</code> <code>im1.show()</code>
锐度增强	<code>ImageEnhance.Sharpness</code>	<code>en=ImageEnhance.Brightness(im)</code> <code>im1=en.enhance(2)</code> <code>im1.show()</code>

## 6.5 综合应用

【例 6.11】生成 Fibonacci 数列的前 10 个数写入文件 `D:\Test\Fibonacci.txt`，格式如图 1.6.14 所示。

分析：Fibonacci 数列又称黄金分割数列，因数学家列昂纳多·斐波那契（Leonardoda Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、13、21、34……在数学上，Fibonacci 数列以如下被以递推的方法定义： $F(1)=1$ ， $F(2)=1$ ， $F(n)=F(n-1)+F(n-2)$ （ $n \geq 3$ ）。

程序：

```
fp=open(r'D:\Test\Fibonacci.txt','w')
f1=1
fp.write("Fib(1)=%d\n"%f1)
f2=1
fp.write("Fib(2)=%d\n"%f2)
for i in range(3,11):
    f3=f1+f2
```

```

fp.write("Fib(%d)=%d\n"%(i,f3))
f1=f2
f2=f3
fp.close()

```

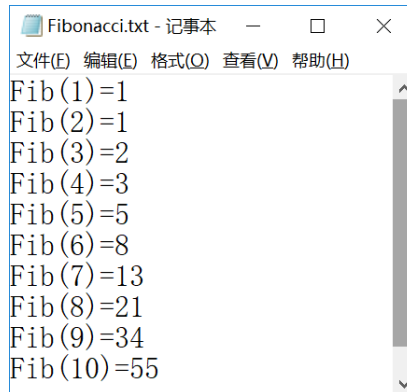


图 1.6.14 Fibonacci 数列的前 10 个数据

【例 6.12】设计一个文件加密程序，如图 1.6.15 和图 1.6.16 所示。它将 D:\Test\Source 文件中的内容加密，加密后的内容存入 D:\Test\Target。加密方法是：A→F、a→f、B→G、b→g……Y→D、y→d、Z→E、z→e，即每一个字母加一个密钥（5），其他字符不变。

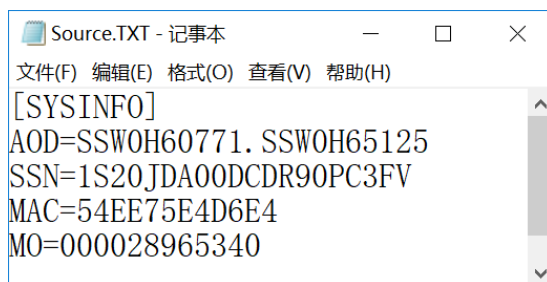


图 1.6.15 源文件

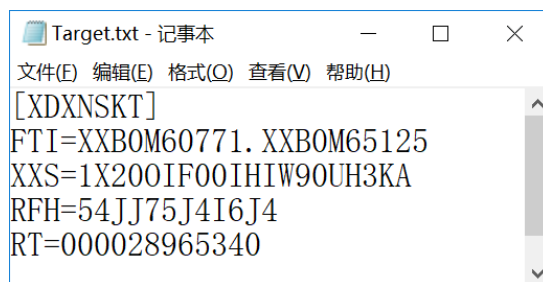


图 1.6.16 加密文件

程序：

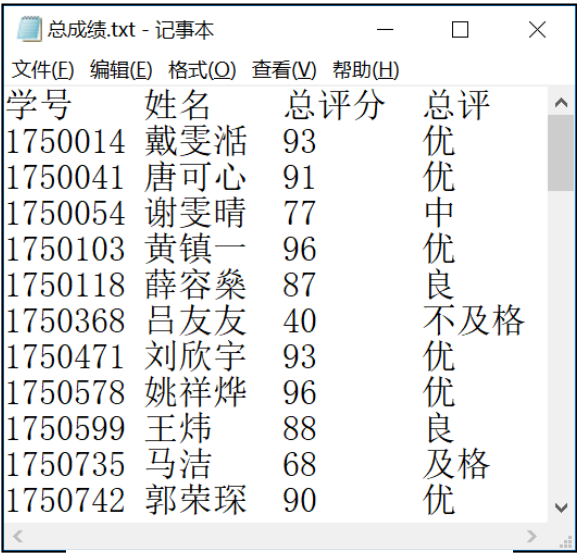
```

fp1=open(r'D:\Test\Source.txt','r')
fp2=open(r'D:\Test\Target.txt','w')
lines=fp1.readlines()
for line in lines:
    newline=""
    for ch in line:
        if "A"<=ch<="U" or "a"<=ch<="u":
            ch=chr(ord(ch)+5)
        elif "V"<=ch<="Z" or "v"<=ch<="z":
            ch=chr(ord(ch)+5-26)
        newline=newline+ch
    fp2.write(newline)
fp1.close()
fp2.close()

```

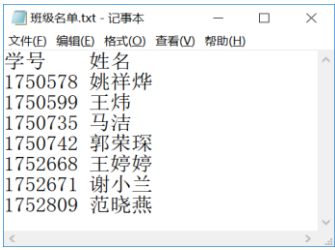
【例 6.13】数据分拣。现有一个总成绩文件和一个班级名单文件，如图 1.6.17 和 1.6.18 所示，数据分拣就是根据班级名单中的学号从总成绩文件把班级同学的成绩分拣出来，组成

一个新的文件，如图 1.6.19 所示。



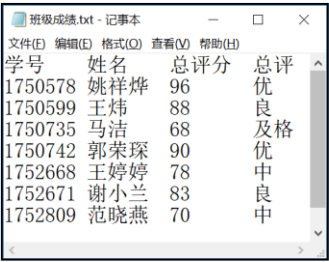
学号	姓名	总评分	总评
1750014	戴雯滢	93	优
1750041	唐可心	91	优
1750054	谢雯晴	77	中
1750103	黄镇一	96	优
1750118	薛容燊	87	良
1750368	吕友友	40	不及格
1750471	刘欣宇	93	优
1750578	姚祥烨	96	优
1750599	王炜	88	良
1750735	马洁	68	及格
1750742	郭荣琛	90	优

图 1.6.17 学生总成绩



学号	姓名
1750578	姚祥烨
1750599	王炜
1750735	马洁
1750742	郭荣琛
1752668	王婷婷
1752671	谢小兰
1752809	范晓燕

图 1.6.18 班级名单文件



学号	姓名	总评分	总评
1750578	姚祥烨	96	优
1750599	王炜	88	良
1750735	马洁	68	及格
1750742	郭荣琛	90	优
1752668	王婷婷	78	中
1752671	谢小兰	83	良
1752809	范晓燕	70	中

图 1.6.19 班级成绩文件

```
TotalScoreFile=r"D:\test\总成绩.txt"
myClassListFile=r"D:\test\班级名单.txt"
myClassScoreFile=r"D:\test\班级成绩.txt"
fmyClass=open(myClassListFile,"r")
myClassStudentList=[]
lines=fmyClass.readlines()
for line in lines:
    temp=line.split("\t")
    studentNo=temp[0]
    myClassStudentList.append(studentNo)
fmyClass.close()
fTotal=open(TotalScoreFile,"r")
Title=fTotal.readline()
fmyClassScore=open(myClassScoreFile,"w")
fmyClassScore.write(Title)
for line in fTotal:
    temp=line.split("\t")
    thisStudentNo=temp[0]
    if thisStudentNo in myClassStudentList:
        fmyClassScore.write(line)
fTotal.close()
fmyClassScore.close()
```

【例 6.14】微球金纳米颗粒覆盖度计算。一个 SiO<sub>2</sub>，上面覆盖了金颗粒(小白点)，如图 1.6.20 所示，计算覆盖度。

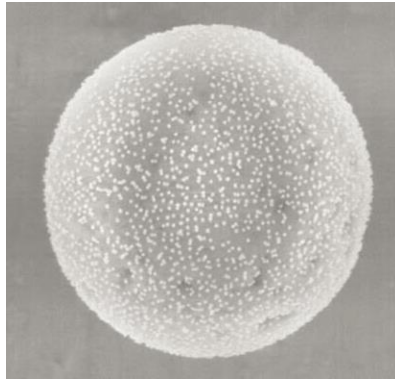


图 1.6.20 覆盖金颗粒的 SiO<sub>2</sub>

分析：从照片上可以看到，金颗粒（小白点）周围有明显边界，球的边缘附近，白色背景，看不清金纳米颗粒。

解决方案：对图像进行二值化处理，找到合理阈值，再计算像素点加和除以圆的总像素点。

```
from PIL import Image
im = Image.open(r"D:\Test\edgeDetect2.jpg")
im = im.convert("L")
im = im.point(lambda x: 255 if x > 200 else 0)
im.show()                                # 2 值化后
count1=0
w,h = im.size                             # 获得图像的宽度和高度，像素单位
for i in range(w):
    for j in range(h):
        pixel=im.getpixel((i,j))         # i,j 位置的像素
        if pixel != 0:
            count1 +=1    #
print(count1/(w*h)*100)
```

## 习题

### 一、选择题

- 文件按存储信息的形式分类，可分为\_\_\_\_\_。
  - 系统文件和用户文件
  - 文本文件和二进制文件
  - 程序文件和数据文件
  - 磁盘文件和 U 盘文件
- 在 Python 中，从计算机内存中将数据写入文件，称为\_\_\_\_\_。
  - 输入
  - 输出
  - 修改
  - 删改
- 若要指定 D:\Data 文件夹下的 T.txt，应可以使用\_\_\_\_\_。
  - r'D:\Data\T.txt'
  - 'D:\Data\T.txt'
  - r'D:\\Data\T.txt'
  - 'D:\\Data\T.txt'



D. `fp=open(C:\\T1.txt,"a")`

11. 程序的运行过程中经常会出现各种各样错误或异常, 为了不影响程序的正常运行, 避免因为错误或异常导致程序中止运行, 程序中使用\_\_\_\_\_语句进行处理。

- A. `if`
- B. `while` 和 `break`
- C. `for` 和 `continue`
- D. `try...except...`

12. 在下列关于 `try...except...` 使用的说法中, 错误的是\_\_\_\_\_。

- A. 若最后一个 `except` 子句没有指定异常类型, 表示处理剩下的异常类型。
- B. `else` 后的语句块是对 `try` 后语句块正常执行后的一种追加处理, 与 `for` 和 `while` 语句中的 `else` 子句相同。
- C. `finally` 后的语句块是在异常发生的情况下执行的。
- D. 使用 `try...except...` 语句的关键是了解异常的类型, 不知道异常类型就无法使用 `try...except...` 语句

13. 在下列关于 `jieba` 的说法中, 错误的是\_\_\_\_\_。

- A. `jieba` 中文分词有三种模式, `jieba.lcut(s)` 属于精简模式。
- B. `jieba` 也可以用于英文文章的分词。
- C. 使用 `jieba` 之前需要安装, 安装命令是 `pip install jieba`。
- D. `jieba.lcut(s)` 分词的结果是一个列表。

14. `jieba` 的分词模式有三种, 下面\_\_\_\_\_不属于 `jieba` 的分词模式。

- A. 精简模式
- B. 全模式
- C. 匹配模式
- D. 搜索引擎模式

15. 执行\_\_\_\_\_函数可进行文档的关键词抽取。

- A. `jieba.analyse.extract_tags()`
- B. `jieba.lcut()`
- C. `jieba.analyse.lcut()`
- D. `jieba.extract_tags()`

16. 执行语句 `from PIL import Image` 后, 要打开指定的图像文件, 从语法上来说, 下面\_\_\_\_\_是正确的。

- A. `im = Image.open(r"E:\pyteach\digit.jpg")`
- B. `im = open(r"E:\pyteach\digit.jpg")`
- C. `im = Image.open("E:\pyteach\digit.jpg")`
- D. `im = open("E:\pyteach\digit.jpg")`

17. 假定有: `im=Image.open(r"C:\Test\tu1.jpg")`, 则在下列关于 `PIL` 的说法中, 错误的是\_\_\_\_\_。

- A. 语句 `im.show()` 用于显示图像



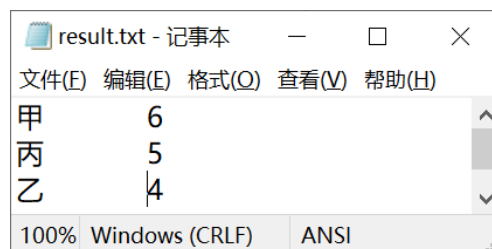
- B. 语句 `im.convert("L")`是将图像转换成 8 位灰度图像。
- C. 语句 `im = im.point(lambda x: 255 if x > 196 else 0)`将图像变成真彩色图像
- D. 属性 `im.format` 的值是图像的格式

18. PIL 库中专门用于图像的增强处理类是\_\_\_\_\_。

- A. Image
- B. ImageEnhance
- C. ImageFilter
- D. pillow

## 二、填空题

1. 在 Windows 中，文件路径分有两种：绝对路径和\_\_\_\_\_路径。
2. 若文件内容编码类型为 ANSI，则中文字符采用 GBK 码，占\_\_\_\_\_个字节。
3. \_\_\_\_\_是针对 Unicode 的一种可变长度字符编码，其特点使用 1~4 字节为每个字符编码。
4. \_\_\_\_\_语句用于处理异常。
5. \_\_\_\_\_格式是一种常用的文本格式，其特点是存储表格数据时常常使用逗号作为分隔符。
6. jieba 库的\_\_\_\_\_方法用于中文分词，其返回值是一个列表。
7. 用于提取关键字的 `extract_tags` 方法位于 jieba 库的\_\_\_\_\_子库中。
8. PIL 是 Python 的第三方库，使用之前必须安装。安装库的名字是\_\_\_\_\_。
9. Image 对象的 `show` 方法用于显示图像
10. Image 对象的\_\_\_\_\_方法用于获取指定位置的像素值。
11. 可以通过 Image 类中的成员函数\_\_\_\_\_来调用滤波函数对图像进行滤波。
12. PIL 模块中的\_\_\_\_\_类专门用于图像的增强处理，可以增强（或减弱）图像的亮度、对比度、色度、锐度等。
13. 统计某项选举活动的投票结果。假定有 3 个候选人甲、乙、丙，每个选举人投一票，投票情况为：甲、甲、丙、甲、乙、乙、甲、乙、甲、丙、乙、丙、甲、丙、丙。统计每位候选人的得票数，并按照得票数的降序排序后写入文件中，数据格式如图 1.6.21 所示。



甲	6
丙	5
乙	4

图 1.6.21 结果文件数据格式

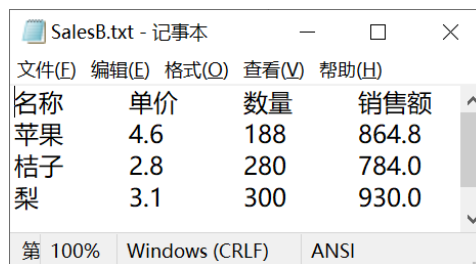
```
candidate=['甲','乙','丙']
vote=['甲','甲','丙','甲','乙','乙','甲','乙','甲','丙','乙','丙','甲','丙','丙']
n=len(candidate)
```

```

count=[0]*n
for i in range(n):
    count[i]=vote.count(candidate[i])
result=____(1)____(zip(candidate,count))
result.sort(key=lambda x:____(2)____,reverse =True)
f=open(r'D:\result.txt','w')
for candi in result:
    s=candi[0]+'t'+____(3)____(candi[1])+'\n'
    f.____(4)____(s)
f.close()

```

14. SalesA.txt 文件中存入了水果的名称、单价和数量，如图 1.6.22 所示。从文件中读出数据，计算销售额，按照销售额降序排列后，写入到文件 SalesB.txt 中，文件内容如下图所示。



名称	单价	数量	销售额
苹果	4.6	188	864.8
桔子	2.8	280	784.0
梨	3.1	300	930.0

```

fp1=open( r'c:\Test\SalesA.txt", "r" )
fp2=open( r'c:\Test\SalesB.txt", ____ (1) ____ )
Titles=fp1.readline()
NewTitles=Titles.rstrip()+"t"+"销售额"+"n"
fp2.write(NewTitles)
data1=fp1.readlines()
data2=[]
for item in ____ (2) ____:
    item=item.rstrip()
    lst=item.split()
    value=float(lst[1])*float(lst[2])
    newitem=item+"t"+str(value)+"n"
    data2.append(____ (3) ____ )
fp2.writelines(____ (4) ____ )
fp1.close()
fp2.close()

```

### 三、问答题

- 什么是文件？文本文件与二进制文件有什么区别？
- 写出满足下列条件的 open 函数的使用形式。
  - 建立一个新的文本文件 new.txt，供用户写入数据。
  - 打开一个老的文本文件 old.txt，用户将从该文件读出数据。
  - 打开一个老的文本文件 append.txt，用户将在该文件后面添加数据。
- 常用文本编码有哪些？如何打开文件时如何指定文本编码？

4. 请用三种不同的方法，将文本文件 `Text.dat` 中的所有内容读出输出在显示器上，要求写出程序代码片段。
5. 请说明 `read`、`readline` 和 `readlines` 方法之间的区别。
6. 请说明 `write` 方法和 `writelines` 方法的区别。
7. 为什么不使用 `close` 语句关闭文件可能会导致文件数据的丢失？
8. CSV 文件有什么特点？如何读写 CSV 文件？
9. 请简述 `jieba` 三种分词模式的区别？
10. PIL 中的 `point` 函数的作用是什么？

## 第七章 面向对象程序设计基础

前面几章介绍了结构化程序设计（SP，Structure Programming）的内容和方法，本章将介绍面向对象程序设计（OOP，Object Oriented Programming）的一些基础知识。面向对象程序设计是以软件的形式对现实世界的对象及其属性及行为进行建模，从而提供一种更加自然直观的程序设计方式。

### 7.1 面向对象程序设计概述

面向对象的编程方法是伴随着软件工程技术的快速发展和软件规模的日益扩大而产生的，它很好地解决了结构化程序设计中数据和操作分离带来的问题，从而更加有利于程序的维护，对数据的操作更加安全、方便。另一方面，客观世界本身就是由各种各样的对象构成的，每种对象都有各自的属性和行为，不同对象之间的相互作用和联系构成了不同的系统。而面向对象的方法就是从现实世界的实体出发，以对象为基本单位，分析、设计和实现一个系统，这种编程思想更符合人们的认知规律。

#### 7.1.1 面向对象的基本概念

面向对象的程序设计从根本上解决了结构化程序设计的缺陷，方法是将数据和所有与这些数据相关的模块一起打包在一个称为“类”的软件构件中。该软件构件中的数据对其内部的所有模块来说，相当于全局变量，操作起来非常方便；而对该构件以外的其它模块来说，通过对这些数据访问权限的设置让其称为不能被访问的数据，从而保障了数据的安全。下面介绍面向对象方法中的一些基本概念。

##### 1. 对象

现实世界中存在着各种各样的实体，包括具体的事物和抽象的事物。这些实体就代表了一个个对象。如一名学生、一门课程、一个圆形都是一个个对象。

每个对象有自己的特征和作用在该对象上的行为。如一个学生对象，具有学号、姓名、性别、成绩等特征，具有选课、查询成绩、退课等行为。当然，一个学生对象也具有身高、体重、视力等特征；也具有吃饭、睡觉等行为。

##### 2. 类

在现实世界的问题领域中，类是对一组对象中共有的属性特征和行为进行的抽象。它的定义中包含了两方面的内容：

- (1) 该类对象的数据描述，包括数据名称及类型。

(2) 该类对象的行为描述，包括具有哪些行为，这些行为是如何实现的。

### 3. 抽象

一个类是一组对象的抽象。抽象是通过特定的实例抽取共同特征以后形成概念的过程。抽象强调主要特征，忽略次要特征；强调给出与应用相关的特性，抛弃不相关的特性。

例如，作为一种抽象，游览线路图提供了给定地理区域中的特征，以便于游客游览。这些特征通常包括主要街道、标志性建筑及游客感兴趣的景点，而不会包括诸如每棵树木，每盏交通信号灯及每家店铺等次要信息。再比如，要处理学生的成绩信息，就应该抽取与此问题相关的学生的学号、姓名、成绩等特征，而抛弃与问题无关的学生的身高、体重、视力等特征。

## 7.1.2 类的基本特征

类具有封装性、继承性和多态性三大特征，在这三大特征支持下，数据操作更加安全，程序维护更加方便，代码复用效率进一步提高，处理上也更加灵活。

### 1. 封装性

在面向对象的程序设计中，为了保护对象的状态不受外界影响，而将对象的状态和其行为打包在一个软件构件中，这种特性被称为封装性，它是面向对象程序设计的基础。

这好比飞机上的“黑匣子”，“黑匣子”内部封存了飞机停止工作前的一段时间内的语音对话和飞行高度、速度、航向等飞行参数，但这些内部细节是被保护起来的，对外是隐而不见的。只有通过相应的设备连通黑匣子上的接口才可以获取黑匣子中的信息。使用黑匣子的用户不必关心黑匣子的内部组成和工作原理，只需要关心各种参数指标（属性）和各种接口的使用（方法）。

又如，我们在使用手机时，也从不需要了解其内部工作原理是怎样的，只需要通过显示屏、按键、听筒这些接口设备即可操作手机。

### 2. 继承性

在工作和生活中，继承性无所不在。例如，要设计一款新电脑，有两个途径：一是从头开始，费时费力；二是对现有的型号加以改进，事半功倍。工程师一般是从原有型号的基础上进行改进，增加一些功能，新的电脑很快设计出来了。旧款电脑派生出了新款电脑，新款电脑继承了旧款电脑的功能，这就是继承和派生的实例。另外，像子承父业、遗产继承都是继承和派生的实例。

在面向对象程序设计中，可以在原有类的基础上定义新的类。原有的类称为基类（或称父类），新的类称为派生类（或称子类）。继承性为代码的复用提供了技术支持，更方便程序的扩充。

### 3. 多态性

多态性一词最早用于生物学，是指同一种族的生物体具有相同的特性。在面向对象程序

设计中，多态性是指同一消息作用在不同对象之上产生不同的行为，它使得程序实现更加灵活方便。例如，教室中有教师和学生两种不同类的对象，当上课铃响时，教师打开投影仪和教案准备讲课，学生则打开笔记本和教材准备听课。也就是说，当上课铃响这一消息传来时，教师和学生这两类不同的对象产生了不同的行为。

## 7.2 类和对象

Python 中有许多已经定义好的类（如列表、元组等），也允许用户自定义类。在这一节中，我们将介绍类的定义和实例化，目的是让大家能更好地理解、使用 Python 或第三方所提供的各种各样的类。

### 7.2.1 类的定义

Python 中，类是通过 `class` 保留字定义的，一般形式如下：

```
class 类名:
    """类文档字符串"""
    类体
```

与定义函数时一样，定义类时可以写一行类文档字符串，用于提供帮助信息。类体需要缩进，用于定义类的属性、方法等。

例 7.1 定义一个表示学生的类 `Student`，并编写一个应用该类的主程序。

```
# 定义Student类
class Student:
    """定义学生类Student"""
    # SetValue方法（函数）用于定义该类对象的数据属性
    def SetValue(self,a,b,c):
        self.No=a
        self.Name=b
        self.Score=c
    # Disp方法（函数）用于输出该类对象的数据属性值
    def Disp(self):
        print(self.No)
        print(self.Name)
        print(self.Score)

p=Student()                # 创建类Student的实例（对象）
p.SetValue("1900001", "王涛", 82) # 对象p调用SetValue方法（函数）定义自己的数据成员
p.Disp()                   # 调用Disp方法（函数）输出对象p中数据成员的值
```

下面按照代码的执行顺序说明类是如何定义和使用的。

① 语句 `p=Student()` 用于创建 `Student` 类的一个实例（对象），并且赋值给 `p`，由 `p` 引用这个实例。

② 类 `Student` 中定义了 `SetValue` 和 `Disp` 函数，类中的函数被称为方法。实例（对象）

创建后，方法被绑定到实例（对象）中，可以通过实例（对象）调用。

③ 语句 `p.SetValue("1900001", "王涛", 82)` 是 `p` 调用自己的 `SetValue` 函数定义自己的数据成员，调用时三个实在参数分别赋值给 `a`、`b`、`c`。

④ `SetValue` 函数的第一个参数是 `self`，是对象自身的引用。

⑤ `p.Disp()` 输出 `p` 中的三个数据成员。

```
1900001
王涛
82
```

图 1.7.1 程序运行结果

从上面的例子可以看到：

（1）定义类的关键字为 `class`，`class` 后是类名。

（2）类的方法就是指类中定义的那些函数，形式上除了第一个参数是必须是 `self` 以外其他的与普通函数相同。一般形式为：

对象.方法名(实在参数表)

（4）类的成员

类的成员可以简单地分为私有成员、保护成员和公有成员，他们的形式和访问权限如表 1.7.1 所示。

表 1.7.1 类的成员

类的成员	形式	示例	访问权限
私有成员	以两个下划线开头	<code>__x</code>	只能在类的内部可以访问，在类的外部不能直接访问，
保护成员	以一个下划线开头	<code>_x</code>	只能在类和派生类中可以访问，在类的外部不能直接访问
公有成员	没有以下划线开头	<code>Name</code>	在类的内部可以访问，在类的外部也可以访问。

对于初学者来说，类的数据成员一般都简单地处理为公有成员。当基本掌握了面向对象程序设计思想后再精细地区别私有成员、保护成员和公有成员。

（5）可以使用 `isinstance` 内置函数测试一个实例是否为某个类的实例，也可以使用 `type` 内置函数查看实例类型。例如，在例 7.1 最后添加如下代码：

```
print(isinstance(p, Student))
print(type(p))
```

则显示结果如下：

```
True
<class '__main__.Student'>
```

表示 `p` 引用的实例是 `Student`。

例 7.2 定义表示点的类 `PyPoint`，拥有 `SetValue` 函数和 `Distance` 函数。`SetValue` 函数用于定义对象的 `x`、`y` 数据成员，`Distance` 用于计算对象到原点的距离函数。

```
import math
class MyPoint:
    "定义表示平面上点的类MyPoint"
```

```

def SetValue(self,x,y):
    self.x=x
    self.y=y
def Distance(self):
    return(math.sqrt(self.x**2+self.y**2))
d=MyPoint()
d.SetValue(3,4)
print(d.Distance())

```

若将上述程序中的数据成员改为私有成员，则存在错误。

```

import math
class MyPoint:
    "定义表示平面上点的类MyPoint"
    def SetValue(self,x,y):
        self.__x=x                # __x为私有的，只能在类中可以访问
        self.__y=y                # __y为私有的，只能在类中可以访问
    def Distance(self):
        return(math.sqrt(self.__x**2+self.__y**2))    # __x和__y是私有的，在类中可以访问
d=MyPoint()
d.SetValue(3,4)
print(d.__x,d.__y)                # 错误：__x和__y是私有的，在类的外部不能访问
print(d.Distance())

```

## 7.2.2 构造函数

Python 中，如果类的每一个实例（对象）都需要完成相同的初始化工作，则可以定义 `__init__` 函数。`__init__` 函数是一个特殊的函数，在创建实例（对象）时由 Python 自动调用，在 C++、VB.NET 等面向程序设计语言中都有类似函数，尽管形式有所不同，但均被称为构造函数。

Python 中构造函数的特点：

- ① 构造函数名是 `__init__`，第一参数是 `self`，是代表实例（对象）自身；
- ② `__init__` 函数除了 `self` 参数以外，可以拥有其他的形式参数；
- ③ `__init__` 函数可以重载，即可以定义多个参数个数不同的构造函数，但是第一个构造函数必须是 `self`；

- ③ 构造函数是在创建实例（对象）时由系统自动调用，程序中不能直接调用。

例 7.3 为例 7.1 中的类 `Student` 添加构造函数，使之能执行如下的语句。

```
p=Student("1900001", "王涛", 82)
```

构造函数如下：

```

def __init__(self,a,b,c):
    self.No=a
    self.Name=b
    self.Score=c

```



当类实例化时，Python 首先创建实例对象，然后检查是否定义了\_\_init\_\_函数。如果定义了则调用\_\_init\_\_函数，调用时将实例对象作为实参传递给 self，将实例化时的实在参数传递给相应的形式参数，实例化时的参数要与\_\_init\_\_的形式参数匹配。例如，定义了上面的\_\_init\_\_函数，则下面的语句都是错误的

```
p=Student()  
p=Student("1900001", "王涛")
```

因为实例化时的参数个数与定义时的形式参数数量不匹配。

例 7.4 定义表示一元二次方程的类 Equation，含有构造函数和求解方程根的函数。

```
import math  
class Equation:  
    def __init__(self,x,y,z):  
        self.a=x  
        self.b=y  
        self.c=z  
    def solve(self):  
        delta=self.b**2 - 4.0*self.a*self.c  
        if (delta>=0):  
            root1=(-self.b+math.sqrt(delta))/(2.0*self.a)  
            root2=(-self.b-math.sqrt(delta))/(2.0*self.a)  
            ans=[root1,root2]  
            return ans  
        else:  
            return []  
equ=Equation(1,-3,2)  
roots=equ.solve()  
print(roots)
```

例 7.5 表示圆的类 Circle。圆有三个数据：圆心（x，y）和半径 r，能够求圆的面积。

```
import math  
class Circle:  
    def __init__(self,x,y,r):  
        self.x=x  
        self.y=y  
        self.r=r  
    def area(self):  
        return math.pi*self.r**2  
p=Circle(0,0,2)  
print(p.area())
```

## 7.2.3 类的实例及其成员的访问

### 1. 类的实例化

定义了类以后，就可以创建类的实例（对象）。类的实例化的一般形式为：

变量=类(实在参数表)

说明：

① 若没有定义构造函数，则实例化时不应有实在参数表；若定义了构造函数，则一定要有实在参数表。

② 将实例（对象）赋值给一个变量，实质是将实例（对象）的地址赋予对象变量，变是与实例的关系是引用，如图 1.7.2 所示。

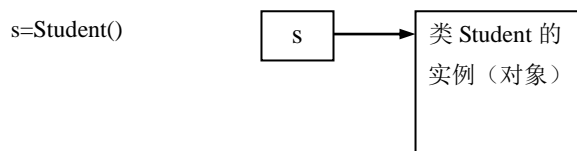


图 1.7.2 对象变量

## 2. 对象成员的访问

对象成员是通过对象变量才能访问其成员。其一般形式为：

数据成员：对象变量.数据成员名

方法：对象变量.方法(<实在参数表>)

## 7.3 继承和派生

继承性是面向对象程序设计的重要特性。利用继承性，可以实现代码重用，提供无限重复利用程序资源的途径，节省程序开发的时间和资源。

所谓继承，就是在已有类的基础上构造新的类，新类继承了原有的类的数据成员、属性、方法和事件。已有的类称为基类，新的类称为派生类。从集合的角度来说，派生类是基类的子集。例如，若在类的基础上定义表示学生干部的派生类 `StudentLeader`，则 `StudentLeader` 类是 `Student` 的子集，如图 1.7.3 所示。若用图来表示基类与派生类的关系，则用带箭头的直线从派生类指向基类。



图 1.7.3 基类与派生类的关系

在 Python 中，一个派生类可从一个基类派生，称为单继承；也可从多个基类派生，称为多继承。

图 1.7.3 描述了形状类的继承层次结构关系，形状类是个基类，它包括二维形状和三维形状，称这两个类是形状类的派生类，而梯形和三角形又属于二维形状，称它们是二维形状类的派生类。在该继承关系里，每个派生类只有一个基类，因而都是一种单继承的关系。而图 1.7.4 所示的为计算机系组成人员的继承层次结构中，系主任既是教师，又是行政管理人

员，因而系主任类是从这两个类派生而来的，是一种多继承关系。限于篇幅，本节只介绍单继承。

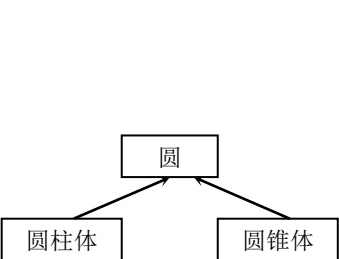


图 1.7.3 形状类的继承

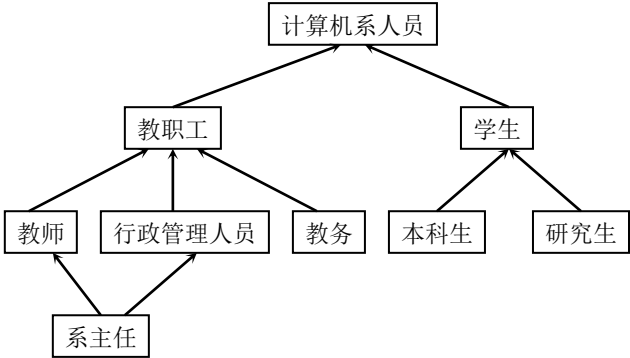


图 1.7.4 计算机系组成人员的继承层次结构

### 1. 引例

首先看一个学生干部类(StudentLeader)的定义。

```
class StudentLeader:
    # SetValue定义对象的属性
    def SetValue(self,a,b,c,d):
        self.No=a           # 学号
        self.Name=b         # 姓名
        self.Score=c        # 成绩
        self.Duty=d         # 职务
    # Disp用于输出对象的数据成员值
    def Disp(self):
        print(self.No)
        print(self.Name)
        print(self.Score)
        print(self.Duty)
```

学生干部类与学生类的区别是增加了一个表示职务的 Duty 数据成员。

对比例 7.1 所定义的 Student 类可以发现，StudentLeader 类的许多信息都与他相同，只是在该类中增加了一个 Duty 属性，类中的方法也基本不同。若能利用继承机制使用已有 Student 类的成员，则可避免代码的重复编写，采用的方法是将 StudentLeader 类定义为 Student 类的派生类。

例 7.6 定义派生类 StudentLeader。

```
class Student:
    def SetValue(self,a,b,c):
        self.No=a
        self.Name=b
        self.Score=c
    def Disp(self):
        print(self.No)
```

```

        print(self.Name)
        print(self.Score)
class StudentLeader(Student):
    def New_SetValue(self,a,b,c,d):
        Student.SetValue(self,a,b,c)
        self.Duty=d
    def New_Dispatch(self):
        Student.Dispatch(self)
        print(self.Duty)
sLeader=StudentLeader()
sLeader.New_SetValue("1900001","王涛",82,"班长")
sLeader.New_Dispatch()

```

## 2. 派生类的定义

在 Python 中，定义派生类的形式非常简单，只需要在派生类的类名后的括号中加上父类的名字即可。一般形式如下：

```

class 派生类名(基类名):
    "类文档字符串"
    类体

```

## 3. 派生类的构造函数

派生类继承了基类的所有成员，但是基类的构造函数是不能继承的。因此，若需要对派生类对象进行初始化，则需要定义新的构造函数。定义派生类的构造函数的一般形式：

```

__init__(派生类构造函数总参数表)
    基类名.__init__(基类构造函数参数表)
    定义派生类对象属性数据成员初始化
End Sub

```

例 7.7 为例 7.6 中的 StudentLeader 类添加构造函数，并且编写程序测试。

```

class Student:
    def __init__(self,a,b,c):
        self.No=a
        self.Name=b
        self.Score=c
    def Disp(self):
        print(self.No)
        print(self.Name)
        print(self.Score)
class StudentLeader(Student):
    def __init__(self,a,b,c,d):
        Student.__init__(self,a,b,c)
        self.Duty=d
    def New_Dispatch(self):
        Student.Dispatch(self)
        print(self.Duty)
sLeader=StudentLeader("1900001","王涛",82,"班长")

```

## 7.4 迭代器和生成器

Python 之所被认为是美丽的程序设计语言，原因之一是已将迭代器、生成器等功能内置在语言中，这些功能的使用有时会简化程序设计的复杂度。例如，处理大量数据时常遇到计算机内存不足的问题，使用迭代器或生成器后就没有必要将所有数据一次性都放入内存中，而是根据处理所需调入数据，减少了计算机内存的负载，这就是迭代器和生成器的作用。

### 7.4.1 迭代器

#### 1. 可迭代对象

可迭代对象是能够一次返回其一个成员的对象。简单地说，任何可以用 for 循环遍历的对象都是可迭代对象，例如：

```
# iterables
sample = ["复旦大学", "上海交通大学", "同济大学", "华东师范大学"]
for i in sample:
    print(i)
```

这里，sample 可以用 for 循环遍历，所以是一个可迭代对象。因此，字符串、列表、元组、字典、集合都是可迭代对象。从语法上来说，for 循环语句中 in 后面应该是一个可迭代对象。可迭代对象为什么可以用 for 遍历？因为可迭代对象内部有一个迭代器。

#### 2. 迭代器

迭代器就是用来从可迭代对象中进行取值的循环方法，必须用\_\_iter\_\_()和\_\_next\_\_()实现。拥有\_\_iter\_\_方法的对象是可迭代的，拥有\_\_next\_\_()方法的对象是迭代器。下面通过一个例子说明\_\_iter\_\_()和\_\_next\_\_()是如何实现的。

**【例 7.7】** 自定义一个迭代器，实现斐波那契数列。

```
class Fibs():
    def __init__(self,max):
        self.max=max
        self.n,self.a,self.b=0,0,1
    #定义__iter__方法
    def __iter__(self):
        return self
    #定义__next__方法
    def __next__(self):
        if self.n<self.max:
            tmp=self.b
            self.a,self.b=self.b,self.a+self.b
            self.n+=1
            return tmp
        raise StopIteration # 停止迭代
```

```
for item in Fibs(10):
    print(item,end=" ")
```

程序运行结果为：1 1 2 3 5 8 13 21 34 55。

说明：

- ① `__init__()`方法是类构造函数，调用类时会首先执行该函数。`n`、`a`、`b`的初值设置为0、0、1。
- ② `__iter__()`和`__next__()`方法使这个类变成了迭代器。
- ③ `__iter__()`方法返回迭代器对象并对迭代进行初始化。由于类对象本身是迭代器，因此它返回自身；
- ④ `__next__()`方法从迭代器中返回当前值，并改变下一次调用的状态。

## 7.4.2 生成器

### 1. 生成器

生成器也是迭代器，但更加优雅。使用生成器，可以实现与迭代器相同的功能，但不必在类中编写`__iter__()`和`__next__()`函数。生成器就是一个能够完成与迭代器相同功能的简单函数。

```
def Fibs(n):
    a,b=0,1
    while a<n:
        value=a
        a,b=b,a+b
        yield value
for item in Fibs(10):
    print(item,end=" ")
```

这个生成器函数和常规函数是有区别的。普通函数使用 `return` 关键字返回值，但是这个生成器函数使用 `yield` 关键字返回值。这就是生成器函数与常规函数不同的地方，除了这种区别其他完全相同的。

`yield` 关键字的工作方式类似于普通的 `return` 关键字，但有额外的功能：它能记住函数的状态。因此，下次调用生成器函数时，它不是从头开始，而是从上次调用中停止的位置开始。

### 2. 生成器表达式

为了简化编写生成器函数的工作，Python 提供了生成器表达式，就像列表生成式一样。唯一的区别是生成器表达式包含在圆括号内。例如，

```
square_gen = (x*x for x in range(1,10))
for i in square_gen:
    print(i)
```

最后，通过比较产生 10000 个整数的列表推导式和生成器表达式所占用的内存空间说明

为什么要使用生成器。

```
import sys
mylist = [i for i in range(10000000)]          # 列表推导式所产生的列表
print('列表所占用内存的大小: ',sys.getsizeof(mylist))
mygen = (i for i in range(10000000))          # 生成器表达式
print('生成器表达式所占用内存的大小: ',sys.getsizeof(mygen))
```

程序运行结果为：

```
列表所占用内存的大小:  81528056
生成器表达式所占用内存的大小:  120
```

## 7.5 模块、库、包

### 1. 模块

随着代码量的不断增多，程序的维护成了一件困难的事情。为了解决这一问题，通常会进行拆分，将具有类似功能的代码放同一文件中。这样每个文件包含的代码就相对较少。在 Python 中，一个.py 文件称为一个模块（module），每一个模块中往往定义了一些变量、函数、类等。例如，图 1.7.X 中的 Test1.py 和 Test2.py 都可以称为模块，模块名分别为 Test1 和 Test2。

```
""" Test1.py 模块 """
import Test2
Angle=float(input("请输入一个角度: "))
Answer=Test2.MySin(Angle)
print(Answer)
```

```
""" Test2.py 模块 """
import math
def MySin(x):
    s = 0
    t = x
    n = 1
    while (math.fabs(t) >= 1e-6):
        s=s+t
        n=n+1
        t=-t*x*x/(2 * n - 1)/(2 * n - 2)
    return s
```

使用模块的好处如下所示：

1. 提高代码的可维护性
2. 提高代码的可复用性。当一个模块编写完成之后，可以被其他地方引用。
3. 使用模块还可以避免变量名、函数名、类名等命名冲突。相同名字的函数、变量名、类等可以放在不同的模块中，但要避免与内置函数名冲突

### 2. 包

模块虽然解决了函数名、变量名和类名等命名冲突，但是模块之间也会存在命名冲突的情况。为了解决模块命名冲突问题，Python 引入了按目录来组织模块的方法。包（package）就是一个内含模块或子包的有层次的文件目录结构，其中一定要\_\_init\_\_.py 模块。例如，可

以将 Test1 和 Test2 模块组合成 Test 包，如图 1.7.X 所示。

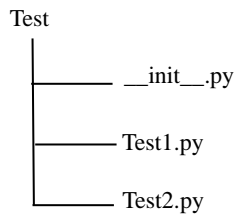


图 1.7.X Test 包的目录结构

包目录具有如下两个特点：

- ① 每个包目录中必须含有\_\_init\_\_.py 模块，否则 Python 会将视为普通目录，而非一个包。\_\_init\_\_.py 文件内容可以为空，也可以有相应的代码。
- ② 一个包目录中还可以存在多个目录，称为子包，组成多层级的包。

### 3. 库

库（lib）是具有相关功能模块的集合。库是一个抽象概念，任意一个模块或包都可以称为库。库通常分成三类：标准库、第三方库以及自定义模块。

- ◆ 标准库 Python 自带的库。math、random 等模块都是其中的一模块。
- ◆ 第三方库 就是第三方发布的具有特定功能的模块。Jieba、Matplotlib、NumPy、pandas、scikit-learn、TensorFlow、PyTorch、Keras、SciPy 等。
- ◆ 自定义模块 用户自己可以自行编写模块，然后使用。

## 7.6 综合应用

本章简要地介绍了类的定义以及类的继承性，下面介绍 3 个综合应用案例，希望能帮助读者理解本章的内容。

例 7.8 定义一个时间类 MyDate，具有年、月、日三个属性，以及一个求第二天日期的函数和输出日期的函数。

```
import datetime                                # 导入datetime库
class MyDate:
    def __init__(self):
        t=datetime.datetime.now()              # 获取当前时间
        self.MyYear=t.year
        self.MyMonth=t.month
        self.MyDay=t.day
    def NextDate(self):
        TotalDays=[[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31], \      # \是续行符
                    [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]]
        self.MyDay = self.MyDay + 1
        leap = ((self.MyYear % 400==0) or (self.MyYear%4==0 and self.MyYear%100!=0))
        if self.MyDay>TotalDays[leap][self.MyMonth - 1]:
            self.MyDay = 1
            self.MyMonth = self.MyMonth + 1
            if self.MyMonth>12:
                self.MyMonth=1
```



```

        self.MyYear=self.MyYear + 1
    def Disp(self):
        print(self.MyYear)
        print(self.MyMonth)
        print(self.MyDay)
d=MyDate()
d.NextDate()
d.Disp()

```

例 7.9 从例 7.5 派出新类--圆柱体类 **Cylinder**，它具有新属性 **h**（高度）和计算体积的 **volume** 函数。

```

import math
class Circle:
    def __init__(self,x,y,r):
        self.x=x
        self.y=y
        self.r=r
    def area(self):
        return math.pi*self.r**2
class Cylinder(Circle):
    def __init__(self,x,y,r,h):
        Circle.__init__(self,x,y,r)
        self.h=h
    def volume(self):
        return self.area()*self.h
p=Cylinder(0,0,2,3)
print(p.area())
print(p.volume())

```

## 习题

### 一、选择题

1. 在面向程序设计中，类有三个基本特征，不属于三个基本特征的是\_\_\_\_\_。
 

A. 封装性	B. 继承性
C. 多态性	D. 交互性
2. 在面向程序设计中，对象具有三要素，不属于三要素的是\_\_\_\_\_。
 

A. 属性	B. 方法
C. 类	D. 事件
3. 有如下类的定义：

```

class person:
    def __init__(self,name):
        self.name=name
    def show(self):
        print(self.name)

```

下面代码能正常执行的是\_\_\_\_\_。

- A. `h=person('张三');h.show()`      B. `h=person();h.show('张三')`  
C. `h=person();h.show()`      D. `h=person('张三');h.show('张三')`
4. 在 Python 中，构造函数的名称是\_\_\_\_\_。
- A. 与类的名称相同      B. `new`  
C. `New`      D. `__init__`
5. 定义类时，以两个下划线开头的是\_\_\_\_\_。
- A. 公有成员      B. 私有成员  
C. 保护成员      D. 构造函数
6. 设计一个类的代码时，类中每个方法的第一个参数都必须是\_\_\_\_\_。
- A. `this`      B. `it`  
C. `me`      D. `self`
7. 在以下说法中，错误的是\_\_\_\_\_。
- A. 一个基类可以派生出很多个派生类  
B. 一个类只能创建一个对象  
C. 派生类除了可以继承基类的属性和方法，还可以定义自己的属性和方法。  
D. 类定义中可以没有 `__init__` 方法。
8. 在以下说法中，错误的是\_\_\_\_\_。
- A. 任何可以用 `for` 循环遍历的对象都是可迭代对象。  
B. 生成器函数不一定要使用 `yield` 关键字返回值。  
C. 生成器表达式包含在圆括号内。  
D. 使用迭代器或生成器后可以节省内存。
9. 在以下说法中，错误的是\_\_\_\_\_。
- A. 一个 `.py` 文件称为一个模块（`module`）。  
B. 每个包目录中必须含有 `__init__.py` 模块。  
C. 任意一个模块或包都可以称为库。  
D. 使用自定义模块时不需要用 `import` 导入。

## 二、填空题

1. 类具有\_\_\_\_\_、继承性和多态性三大特征。
2. 在类中，私有成员是以\_\_\_\_\_个下划线开头。
3. 构造函数的名称是\_\_\_\_\_函数。
4. 所谓继承，就是在已有类的基础上构造新的类，新类继承了原有的类的数据成员、属性、方法和事件。已有的类称为\_\_\_\_\_，新的类称为派生类
5. 拥有 `__iter__` 方法的对象是可迭代的，拥有\_\_\_\_\_方法的对象是迭代器
6. 生成器函数使用\_\_\_\_\_关键字返回值。

7. 包（package）就是一个内含模块或子包的有层次的文件目录结构，其中一定要有\_\_\_\_\_模块。

### 三、问答题

1. 什么是类？什么是对象？类和对象的关系是什么？
2. 面向对象程序设计的主要特性是什么？
3. 构造函数有什么特点？如何编写构造函数？
4. 类的方法实质是什么？
5. 什么是派生？派生类的构造函数如何编写？
6. self 代表什么？

## 第 8 章 数据可视化

数据可视化是指用图形图表的方式呈现数据，可以直观清晰地展示出数据包含的信息，有助于分析数据中隐含的变化规律或发展趋势，是数据处理过程中非常重要的一个环节。数据可视化广泛应用于销售数据的可视化，统计数据的可视化和机器学习数据的可视化等。实现数据可视化的第三方库主要包括 Matplotlib、Seaborn、ggplot、Bokeh 等，Matplotlib 是 Python 中使用最广泛的可视化库，其他许多可视化库大多是建立在其基础上或者直接调用该库，本章将介绍 Matplotlib 库的使用和常见图表的绘制方法。

### 8.1 Python 图表绘制基础

下面通过某公司销售数据的图表化来了解图表绘制的一般过程。

#### 8.1.1 引例—销售量柱状图

【例 8.1】绘制某公司某年度的销售柱状图（又叫条形图），显示某产品各季度的销售量，效果如图 1.8.1 所示。

分析：首先准备好绘制柱状图所需的数据，这里是某产品的销售量，可以是列表类型，也可以是元组类型，分别控制各个柱形的高度，然后确定每个柱形的宽度和位置，最后调用 matplotlib.pyplot 库中的 bar 函数绘制柱状图。

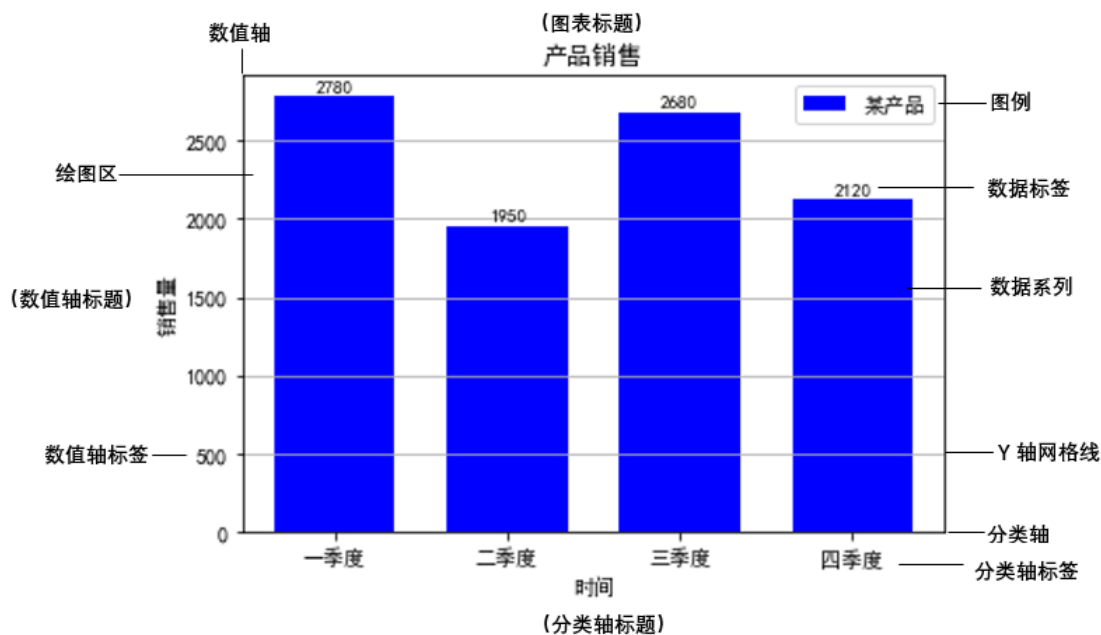


图 1.8.1 销售柱状图

① 导入库

```
import matplotlib.pyplot as plt
```

```
#导入 pyplot 子库
```

## ② 新建绘图区

<code>plt.figure(figsize=(6,4))</code>	#绘图区大小为 6*4 英寸
<code>plt.rcParams['font.sans-serif']=['SimHei']</code>	#设置显示汉字（黑体）

## ③ 准备数据

<code>seasons=['一季度', '二季度', '三季度','四季度']</code>	#设置分类轴显示文本
<code>sales=[2780,1950,2680,2120]</code>	#设置某产品的销售量数据
<code>index=[0,1,2,3]</code>	#index 控制分类轴刻度
<code>barW=0.7</code>	#barW 控制条形的宽度

## ④ 设置图表属性

<code>plt.title("产品销售")</code>	#设置图表标题
<code>plt.xlabel("时间")</code>	#设置 X 坐标轴标签
<code>plt.ylabel("销售量")</code>	#设置 Y 坐标轴标签
<code>plt.xticks(index,seasons)</code>	#设置分类轴标记
<code>plt.grid(axis="y")</code>	#显示横向网格线

## ⑤ 绘制图表

<code>plt.bar(index,sales,barW,color='b',label='某产品')</code>	#绘制柱状图
<code>for a,b in zip(index,sales):</code>	#显示数值标注
<code>plt.text(a,b+30, '%.0f'%b, ha='center', fontsize=9)</code>	#ha 设置水平对齐方式

## ⑥ 显示图例

<code>plt.legend(loc='upper right')</code>	#在右上角显示图例
--	-----------

## ⑦ 显示图表

<code>plt.show()</code>
-------------------------

从引例中看到，Python 中图表的绘制过程比较简单，关键是要准备好绘制所需的数据，控制好各图表元素的显示，掌握图表绘制函数的调用方法。

## 8.1.2 Matplotlib 绘图基础

Matplotlib 是 Python 环境下常用的一个绘图库，可生成指定分辨率的图形文件，满足各种出版级图形的要求，官网下载地址为：<https://matplotlib.org/>。

### 1. 子绘图库

(1) `pyplot` Matplotlib 库下的一个模块，又叫子绘图库，能够绘制出各种常见的图表类型，包括二维图表和三维图表，还可同时绘制多个子图。

(2) `pylab` 也是 Matplotlib 的子绘图库，同样可以绘制二维和三维图表，区别是 `pylab` 中包括了许多 `numpy` 和 `pyplot` 模块中常用的函数，方便用户快速进行计算和绘图，更适合在 IPython 交互式环境中使用。

在开始绘制图表前，先用“`import matplotlib.pyplot as plt`”导入绘图库，后面用别名 `plt` 来指代 `matplotlib.pyplot` 子绘图库。

### 2. 图表的组成

Matplotlib 的图表区包含三个层次：容器层，辅助显示层和图像层。

(1) 容器层 由 `Canvas`（画布）、`Figure`（绘图区）和 `Axes`（坐标系）三个部分组成，`Canvas` 位于容器层的最底部（由系统自动创建），`Figure` 则建立在 `Canvas` 上面，而 `Axes` 则建立在 `Figure` 上面。一个 `Figure` 对象内可以包含多个 `Axes` 对象，每个 `Axes` 对象都是一个独立的坐标系，绘图过程中所有图像都是基于坐标系绘制的。

(2) 辅助显示层 包括 `title`（标题）、`axis`（坐标轴）、`axis label`（坐标轴标题）、`tick`（坐标

轴刻度)、tick label (坐标轴刻度标签)、spines (图表边框线)、grid (网格线)、legend (图例)、facecolor (前景色)等。

(3) 图像层 调用各个绘图函数绘制的柱状图、折线图、饼图、散点图等图像。

注意：辅助显示层和图像层都必须创建在 Axes 对象上。

### 3.绘图区

绘图区是图表绘制的基础，既可以由系统自动创建，也可以根据需要调用 Figure()函数来创建，还可以将一个绘图区划分为多个子绘图区，各个子绘图区拥有自己独立的坐标系，可以在指定的子绘图区上绘制图表。

#### (1) 自动创建绘图区

在 Matplotlib 中可以直接调用图表绘制函数，系统将自动创建一个绘图区，默认大小为 6.4\*4.8 英寸，即 figsize=(6.4,4.8)，分辨率为 100，即 dpi=100，背景为白色，即 facecolor="white"。

例如：

```
import matplotlib.pyplot as plt
plt.plot((3,5))      #自动创建一个绘图区，绘制一条从(0,3)到(1,5)的直线
```

#### (2) 手动创建绘图区

如果需要指定绘图区的大小、分辨率和背景颜色等参数，那么先调用 figure()函数创建一个绘图区，然后再调用图表绘制函数。

例如：

```
import matplotlib.pyplot as plt
plt.figure(figsize=(4,3),dpi=72)      #绘图区指定为 4*3 英寸，分辨率为 72
plt.plot((6,2))      #绘制一条从顶点(0,6)到顶点(1,2)的直线
```

常用绘图区控制函数如下：

① figure 函数，用于生成一个全局绘图区，figure 函数的使用形式为：

```
figure(num=None,figsize=None,dpi=None,facecolor=None,**kwargs)
```

说明：

- 所有参数均可省略，创建的绘图区上没有建立坐标系，默认不显示该绘图区，可以调用函数 plt.show()来显示。

例如：

```
>>>plt.figure()
```

在 IDLE 中显示执行结果为<Figure size 640x480 with 0 Axes>，即绘图区大小为 640\*480 像素，无坐标系。

- 参数 figsize 用于指定绘图区大小，dpi 指定分辨率，facecolor 指定绘图区的颜色。

例如：

```
>>>plt.figure(figsize=(4,3),facecolor='b')
```

#指定绘图区大小为 4\*3 英寸，背景色为蓝色

```
>>>plt.show()      #显示绘图区，效果如图 1.8.2 所示。
```

② axes 函数，用于创建一个坐标系风格的绘图区并显示，axes 函数的使用形式为：

```
axes((left, bottom, width, height), facecolor='w')
```

说明：所有参数均可省略，也可以在参数中指定绘图区的位置与大小，设置绘图区的颜色等。

例如：

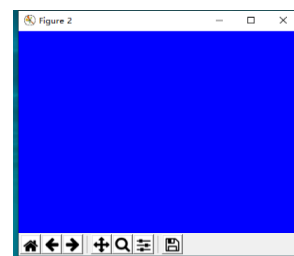
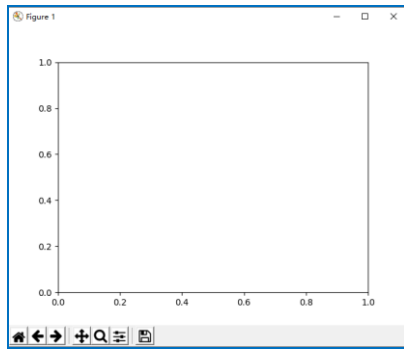


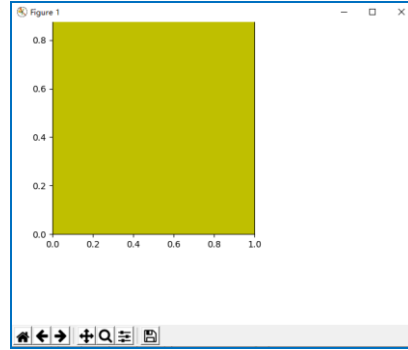
图 1.8.2 全局绘图区

```
>>> plt.axes( ) #显示效果如图 1.8.3(a)所示
>>> plt.axes((0.1,0.3,0.5,0.8),facecolor='y') #显示效果如图 1.8.3(b)所示
>>> plt.show( ) #显示坐标系风格的绘图区
```

注意：坐标系的原点默认在绘图区的左下角（0，0），图 1.8.3(b)的坐标原点设在了绘图区的（0.1，0.3）处。如果绘图区大小为 640\*480 像素，这里的（0.5，0.8）则对应宽高为 320\*384 像素。



(a) 默认坐标系位置和大小



(b) 指定坐标系位置和大小

图 1.8.3 IDLE 中坐标系风格的绘图区

### （3）创建子绘图区

在一个全局绘图区中可以同时绘制多个子图，首先创建多个子绘图区，每个子绘图区拥有独立的坐标系，然后在指定的子绘图区中绘制图表。

常用子绘图区控制函数如下：

① subplot 函数，用于在全局绘图区中创建子绘图区并显示指定子绘图区，subplot 函数的使用形式为：

```
subplot(nrows, ncols, index, **kwargs)
```

说明：

- **nrows** 表示子绘图区的行数，**ncols** 表示子绘图区的列数，**index** 表示当前子绘图区的编号，按照从左到右，从上往下的顺序从 1 开始依次编号。

- 如果 **nrows**、**ncols** 和 **index** 的值都小于 10 的话，可以把三个参数的值缩写为一个整数，如 subplot(323)和 subplot(3,2,3)都表示有 3 行 2 列共 6 个子绘图区，显示第 3 个子绘图区。

例如：

```
>>>plt.subplot(1,1,1) #表示有 1 行 1 列，绘制在第 1 个子绘图区
>>>plt.subplot(232) #有 2 行 3 列共 6 个子绘图区，只显示第 2 个子绘图区
>>>plt.subplot(346) #有 3 行 4 列共 12 个子绘图区，只显示第 6 个子绘图区
```

- **kwargs** 参数用于接收图表辅助显示对象的值，例如以下代码创建 2 行 2 列共 4 个子绘图区，在 3 个子绘图区中分别绘制 3 条不同的直线，代码运行结果如图 1.8.4 所示。

```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure()
plt.subplot(221,title="No 1",xticks=np.arange(6)) #该语句可以省略
plt.plot((2,4),(3,6),c='g') #设置图表标题和 X 轴刻度范围
plt.subplot(223,xlabel="Number",ylabel="Value") #设置坐标轴标签
plt.plot((1,2),(4,3),c='b')
```

```
plt.subplot(224, facecolor='#00FFFF')           #设置子绘图区填充颜色为青色
plt.plot((1,2),(3,4),c='y')
plt.subplots_adjust(wspace=0.4,hspace=0.6)       #调整各子绘图区之间的间隔
plt.show()
```

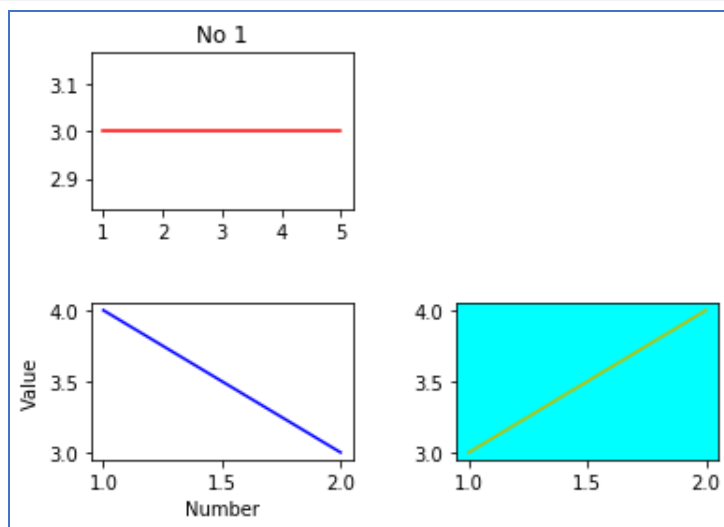


图 1.8.4 在指定子绘图区绘制线条

② `add_subplot` 函数，用于给已有全局绘图区添加子绘图区，`add_subplot` 函数的使用形式为：

```
add_subplot(nrows, ncols, index, **kwargs)
```

说明：参数 `nrows` 和 `ncols` 用于指定子绘图区的行数与列数，仅显示 `index` 参数值对应编号的子绘图区。

例如：创建一个 2 行 2 列的全局绘图区，在第 1 行第 1 列显示一条红色折线，在第 2 行第 2 列显示随机生成的 10 个绿色圆点，参考代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure(figsize=(6,4))           #创建一个指定大小的全局绘图区
fig.add_subplot(221)                    #添加第 1 个子绘图区
plt.plot((1,6,2),c='r',marker='o')     #在第 1 个子绘图区绘制红色折线与圆点标记
fig.add_subplot(224)                    #添加第 4 个子绘图区
x=np.random.randint(0,100,10)           #随机生成 10 个坐标点的 x 坐标序列
y=np.random.normal(0,100,10)            #随机生成 y 坐标序列
plt.scatter(x,y,c='g')                  #在第 4 个子绘图区绘制随机生成的坐标点
plt.show()
```

上面程序执行效果如图 1.8.5 所示。



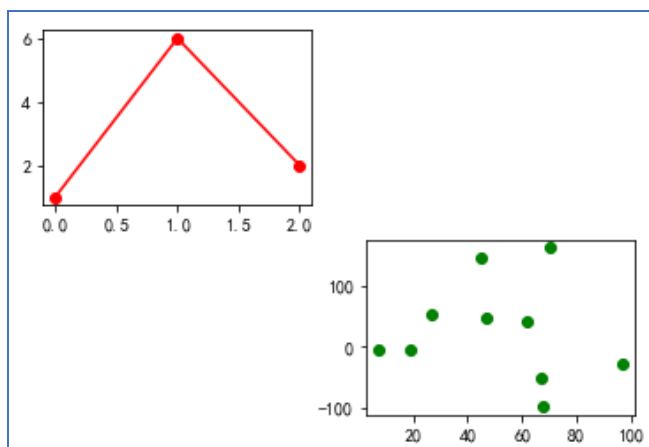


图 1.8.5 添加并显示在指定子绘图区

③ `subplots` 函数，用于创建一个全局绘图区，然后添加多个子绘图区并显示所有子绘图区，`subplots` 函数的使用形式为：

```
subplots(nrows, ncols, **fig_kw)
```

例如：

```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(nrows=2,ncols=3,figsize=(8,4),dpi=100)
#创建 2 行 3 列共 6 个子绘图区，全局绘图区大小为 8*4 英寸，分辨率为 100
plt.subplots_adjust(wspace=0.3,hspace=0.3) #调整子绘图区之间的间隔
ax[0][2].plot((1,'rD')) #在第 0 行第 2 列的子绘图区绘制一个红色菱形标记
ax[1][0].plot((1,2),'b') #在第 1 行第 0 列的子绘图区绘制一条蓝色线条
plt.show()
```

上面程序执行后显示的子绘图区效果如图 1.8.6 所示。

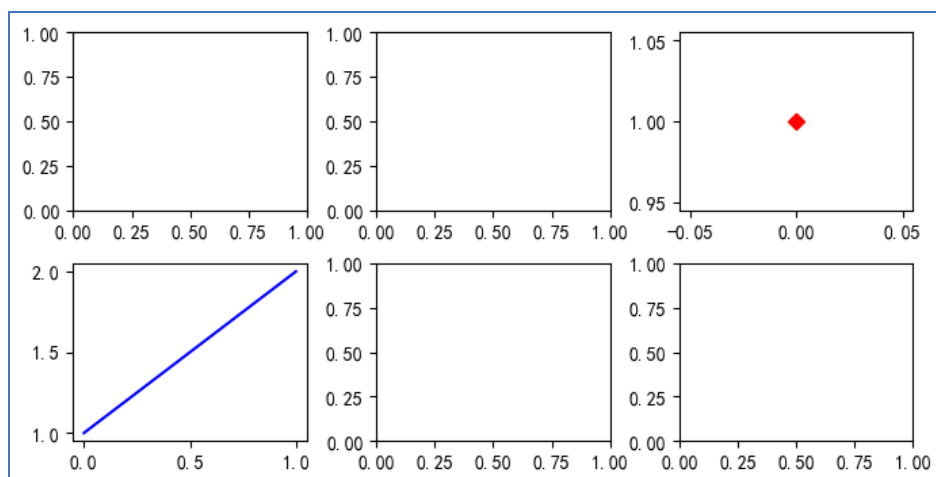


图 1.8.6 显示所有子绘图区

注意：

- 当子绘图区的行数与列数值均大于 1 时，用 `ax[i][j]` 可以访问第  $i+1$  行第  $j+1$  列对应的坐标系，如 `ax[0][0]` 表示的是第 1 行第 1 列的坐标系。

如果将上面代码中的 `ax[0][2]` 与 `ax[1][0]` 都替换成 `plt`，那么红色菱形标记和蓝色线条都绘制到了最后一个子绘图区。

- 当子绘图区的行数或列数只有一个值等于 1 时，可以通过 `ax[i]` 来访问第  $i+1$  个坐标系。

例如：

```
import matplotlib.pyplot as plt
fig,ax=plt.subplots(nrows=3,ncols=1,figsize=(4,3)) #创建 3 行 1 列共 3 个子绘图区
ax[1].plot((1,2,3)) #绘制在第 2 行对应的子绘图区
fig,ax=plt.subplots(nrows=1,ncols=4,figsize=(6,4)) #创建 1 行 4 列共 4 个子绘图区
ax[2].plot((5,3)) #绘制在第 3 列对应的子绘图区
```

如果改用 `plt.plot((1,2,3))`和 `plt.plot((5,3))`来绘制前面的线条，那么线条都绘制到了当前绘图区的最后一个子绘图区。

- 当子绘图区的行数与列数值均等于 1 时，绘图区只有一个坐标系，既可以通过 `plt` 来访问该坐标系，也可以通过 `ax` 来访问。

例如：

```
fig,ax=plt.subplots(nrows=1,ncols=1,figsize=(3,2)) #创建 1 行 1 列共 1 个子绘图区
ax.scatter([5,3,8],[1,3,5],marker="D") #绘制在当前坐标系
```

#上一行代码可以改写成 `plt.scatter([5,3,8],[1,3,5],marker="D")`，输出结果完全相同。

下面代码通过 `ax` 来访问图表的各个元素并绘制一个柱状图。

```
import matplotlib.pyplot as plt
fig,ax=plt.subplots(figsize=(6,4)) #ax 用于访问指定坐标系
plt.rcParams['font.sans-serif']=['SimHei'] #SimHei 指定中文字体为黑体
subjects=['语文', '数学', '英语', '体育'] #设置分类轴显示的文本内容
score= [80,98, 92,70] #某同学的分
ax.set_title("考试成绩") #设置子绘图区的标题
ax.set_xlabel("科目") #设置子绘图区 X 分类轴的标签
ax.set_ylabel("分数") #设置子绘图区 Y 数值轴的标签
ax.set_xticks([0,1,2,3]) #设置 X 分类轴的标记数据
ax.set_xticklabels(subjects) #设置 X 分类轴标记显示文本内容
ax.set_yticks([0,30,60,90,120]) #设置 Y 数值轴的标记数据
ax.bar([0,1,2,3],score,label="Name") #绘制柱状图
ax.legend(loc='upper right') #显示图例
```

### 8.1.3 图表绘制函数

`pyplot` 子库中提供了许多绘制图表的函数，可以绘制常见的各种图表，当绘图区只有一个坐标系时，一般直接调用 `plt` 的图表绘制函数，默认绘制在当前坐标系，常见图表绘制函数见表 1.8.1。

表 1.8.1 常见图表绘制函数

函 数	描 述
<code>plot([x],y, [fmt],**kwargs)</code>	绘制线条或标记
<code>bar(x, height, width=0.8,**kwargs)</code>	绘制柱状图
<code>pie(x, explode=None,**kwargs)</code>	绘制饼图
<code>scatter(x, y, s=None, c=None, marker=None, cmap=None,**kwargs)</code>	绘制散点图
<code>hist(x,bins=None,range=None, ,rwidth=None, **kwargs)</code>	绘制直方图

图表由多个元素组成，包括坐标轴、图表与坐标轴的标题、坐标轴的刻度和标签、图例

和数据标签等。每个图表元素的输出都可以调用相应的函数或方法来实现，常见图表设置函数见表 1.8.2 所示。

表 1.8.2 图表设置函数

函 数	描 述
<code>plt.axis()</code>	显示当前坐标轴的范围，默认输出为：(0.0, 1.0, 0.0, 1.0)
<code>plt.axes()</code>	显示坐标系风格的绘图区
<code>plt.xlim(left,right)</code> <code>plt.ylim(bottom,top)</code>	设置坐标轴的取值范围 如 <code>plt.xlim(left=1,right=5)</code> #设置 X 轴的取值范围为[1,5] 如 <code>plt.ylim(bottom=3,top=6)</code> #设置 Y 轴的取值范围为[3,6]
<code>plt.text(x,y,s)</code>	在指定位置添加注释文本 如 <code>plt.text(0.5,0.5,'Max')</code> #在两个坐标轴的中间输出字符串 “Max”
<code>plt.grid()</code>	设置绘图区的网格显示 如 <code>plt.grid(True)</code> #显示两个轴向的网格线 如 <code>plt.grid(axis='both')</code> #显示两个轴向的网格线 如 <code>plt.grid(axis='x')</code> #只显示 x 轴的纵向网格线
<code>plt.title(label)</code>	在指定位置显示图表标题 如 <code>plt.title("Title",loc='center')</code> #居中显示标题文本
<code>plt.xlabel(xlabel)</code> <code>plt.ylabel(ylabel)</code>	设置 X 轴的标题，如 <code>plt.xlabel(“分类轴”)</code> 设置 Y 轴的标题，如 <code>plt.ylabel(“数值轴”)</code>
<code>plt.xticks(ticks,labels)</code> <code>plt.yticks(ticks,labels)</code>	设置分类轴和数值轴的刻度及其标签文本 如 <code>plt.xticks(range(4),('一季度','二季度','三季度','四季度'))</code> 如 <code>plt.yticks(range(3),('10','20','30'))</code>
<code>plt.legend(loc)</code>	设置图例显示 如 <code>plt.legend(loc='upper right')</code> #图例显示在右上角 如 <code>plt.legend(bbox_to_anchor=(0.5,1))</code> #通过 xy 轴的相对位置定位
<code>plt.fill(x,y,color)</code>	填充由坐标点构成的连续封闭区域或函数曲线与坐标轴之间的区域 如 <code>plt.fill([3,2,4,3],[5,1,1,5],color='b')</code> #显示一个用蓝色填充的三角形

补充：rcParams，又叫 rc 参数，用于设置图表各个属性的默认值，如绘图区的大小、图像分辨率、线条样式、坐标轴、字体等。由于 Matplotlib 库默认设置的是英文字体，不支持中文字符的显示，所以需要手动设置字体属性或修改 rcParams 中的字体属性，使汉字能够正常显示。

- 设置中文字体，正常显示汉字

方法 1：通过 rcParams 设置指定中文字体。

```
plt.rcParams['font.sans-serif']=['SimHei']      # SimHei 指定中文字体为黑体
```

其他常见中文字体有：SimSun（宋体）、KaiTi（楷体）、Microsoft YaHei（微软雅黑体）等。

方法 2：在需要输出汉字的函数参数中增加 fontproperties 属性，但是只能显示当前函数调用中需要显示的汉字。

```
plt.xlabel('X 轴标签',fontproperties='Microsoft YaHei',fontsize=12)
```

- 设置坐标轴参数，正常显示负号

```
plt.rcParams['axes.unicode_minus']=False
```

- 设置绘图区大小、图像分辨率、线条样式、线条宽度等

```
plt.rcParams['figure.figsize'] = (8,6)      #设置绘图区大小为 8*6 英寸
plt.rcParams['figure.dpi'] = 72            #设置图像的分辨率为 72dpi
plt.rcParams['lines.linestyle'] = '--'      #设置线条样式为长虚线
plt.rcParams['lines.linewidth'] = 5        #设置线条宽度为 5
```

## 8.2 二维图表

二维图表中只有 x 和 y 两个坐标轴，绘制的图形是一种平面图形。

### 8.2.1 绘制线条

plot 函数用于绘制线条和标记，不仅可以绘制折线图、曲线图，还可以制作出类似散点图的效果。plot 函数的使用形式为：

```
plt.plot([x],y, [fmt],**kwargs)
```

说明：

① x 和 y 通常是列表或元组等序列。x 中存储所有顶点的 x 坐标序列，y 中存储所有顶点的 y 坐标序列。例如，图 1.8.7 所示的折线图由 4 个顶点组成，坐标分别为 (1, 2)、(3, 4)、(2, 6) 和 (2, 1)，所以参数 x 的值应为[1, 3, 2, 2]，参数 y 的值应为[2, 4, 6, 1]，plot 函数的具体调用形式为：

```
plt.plot([1,3,2,2],[2,4,6,1])
```

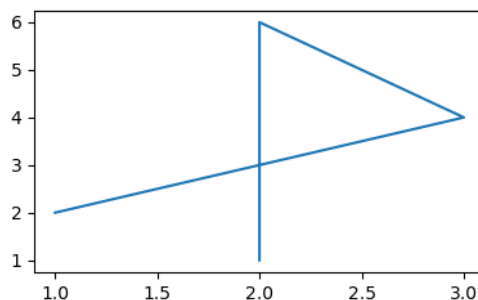


图 1.8.7 plot 函数绘制的折线图

② 参数 x 可以缺省，缺省值为列表[0, 1, 2, ..., n-1]，此处 n 表示顶点数量。例如，函数 plt.plot([0, 1, 2, 3], [2, 5, 3, 6])可以简写为 plt.plot([2, 5, 3, 6])。

③ 参数 fmt 是一个字符串变量，用于定义图表的基本属性，如颜色 (color)、标记 (marker)、线型 (linestyle) 等。参数 fmt 是由各个属性的单个字母或符号缩写组合而成的字符串，组成形式如下（其排列顺序可以改变）：

'[颜色][标记][线型]'

例如，'g'表示绿色，'Dr'表示菱形标记红色，'bo-'表示蓝色圆点实线。用于表示颜色、线型、标记的单个字母或符号见表 1.8.3、表 1.8.4、表 1.8.5。

注意：颜色也可以赋值为十六进制格式的数值，如 `color='#FF0000'` 等价于 `color='r'`。

表 1.8.3 表示颜色的字母缩写

缩写	颜色
'r'	red
'g'	green
'b'	blue
'c'	cyan
'm'	magenta
'y'	yellow
'w'	white
'k'	black

表 1.8.4 表示线型的字母缩写

线型缩写字母	说明	线型
'-'	1 个减号	实线
'--'	2 个减号	长虚线
'-.'	冒号	短虚线
'None'		不画线
'-.'	减号加点	点划线

表 1.8.5 marker 标记

标记	描述	标记	描述	标记	描述	标记	描述
'o'	● (圆圈)	'.'	● (圆点)	'_'	-水平线	'v'	▼ (下三角形)
'D'	◆ (菱形)	's'	■ (正方形)	'8'	●八边形	'<'	◀ (左三角形)
'h'	⬡ (六边形)	'*'	★ (五角星)	'p'	⬡五边形	'>'	▶ (右三角形)
'H'	⬡ (六边形)	'd'	◆ (菱形)	'.'	· 像素点	'^'	▲ (上三角形)
'+'	⊕ (加号)	'X'	⊗ (粗线叉)	'x'	⊗ (细线叉)	' '	⊥ (垂直线)
'1'	⋏ (下三叉)	'2'	⋏ (上三叉)	'3'	⋏ (左三叉)	'4'	⋏ (右三叉)

④ `kwargs` 是一个关键字参数 (`dict` 类型)，用于接收 0 个或多个由属性与属性值组成的键值对。若颜色属性赋值为 “green”、“black” 等单词形式，则不能用 `fmt` 参数来组合赋值，必须对单个颜色属性赋值。

例如：绘制一条从(3,1)到(6,5)的粗细为 2 的绿色短虚线且两端显示菱形标记的命令为：

```
plt.plot([3,6],[1,5], color='green', marker='D', linestyle='-',linewidth=2)
```

可将颜色、线型和线宽的属性名缩写为：

```
plt.plot([3,6],[1,5], c='green', marker='D', ls='-',lw=2)
```

注意：颜色、标记和线型可以采用组合字符串“`gD:-`”，但不能写成“`greenD:-`”。

【例 8.2】绘制三角函数  $y=\sin(x)$  和  $y=\cos(x)$  的曲线图。

方法 1：调用 `math` 库中的三角函数，利用 `plot` 函数输出标记来绘制曲线，效果如图 1.8.8 (a) 所示。

参考代码如下：

```
import math                                #导入数学函数库
import matplotlib.pyplot as plt           #导入绘图库
plt.title("三角函数曲线")                 #显示图表标题
plt.ylim(-1,2)                            #设置 y 轴的取值范围
plt.rcParams['font.sans-serif']=['SimHei'] #设置中文字体
plt.rcParams['axes.unicode_minus']=False   #显示负号
```

```

x=-math.pi
while x<=math.pi:
    sin=math.sin(x)          #计算当前坐标 x 对应的 sin 函数值
    cos=math.cos(x)          #计算当前坐标 x 对应的 cos 函数值
    plt.plot(x,sin,marker='*') #在 sin 函数的每个坐标点输出一个星形标记
    plt.plot(x,cos,marker='D') #在 cos 函数的每个坐标点输出一个菱形标记
    x+=0.1
plt.show()

```

注意：上面代码中绘制图表的 `plot` 函数放在循环内可以正常输出图表，但是当循环次数越来越多时，图表绘制速度会越来越慢，甚至无法正常输出。建议利用 `python` 的列表将绘制的数据计算好并保存，然后再绘制图表，优化后的程序代码如下：

```

import math                    #导入数学函数库
import matplotlib.pyplot as plt #导入绘图库
plt.title("三角函数曲线")      #显示图表标题
plt.ylim(-1,2)                 #设置 y 轴的取值范围
plt.rcParams['font.sans-serif']=['SimHei'] #设置中文字体
plt.rcParams['axes.unicode_minus']=False   #显示负号
x=-math.pi; xx=[]
sinx=[]; cosx=[]
while x<=math.pi:
    sinx.append(math.sin(x))    #sinx 列表添加当前坐标点的 sin 函数值
    cosx.append(math.cos(x))    #cosx 列表添加当前坐标点的 cos 函数值
    xx.append(x)                #xx 列表添加当前坐标点的 x 坐标值
    x+=0.1
plt.plot(xx,sinx,marker='*')    #绘制 sin 函数的曲线
plt.plot(xx,cosx,marker='D')    #绘制 cos 函数的曲线
plt.show()

```

方法 2：引入 `numpy` 库，利用其数学函数快速生成 `x` 轴上的数据系列，并计算对应的函数值，无需通过循环即可生成每个坐标点的数值序列，结合 `numpy` 库来绘制图表将更加简单方便，效果如图 1.8.8（b）所示。

参考代码如下：

```

import numpy as np            #导入 numpy 库
import matplotlib.pyplot as plt #导入绘图库
x=np.linspace(-np.pi,np.pi,100) #快速生成 x 轴上  $(-\pi, \pi)$  之间的 100 个数
sin,cos = np.sin(x), np.cos(x)  #计算相应的 sin 函数值和 cos 函数值
plt.title("三角函数曲线")      #显示图表标题
plt.ylim(-1,2)                 #设置 y 轴的取值范围
plt.rcParams['font.sans-serif']=['SimHei'] #设置中文字体
plt.rcParams['axes.unicode_minus']=False   #显示负号
plt.plot(x,sin,color='b',linewidth=2, ls='-',label='sin 函数') #绘制 sin 函数曲线
plt.plot(x,cos,color='r',linewidth=5, ls='--',label='cos 函数') #绘制 cos 函数曲线
plt.legend(loc= 'upper right' ) #右上显示图例文字
plt.show()

```

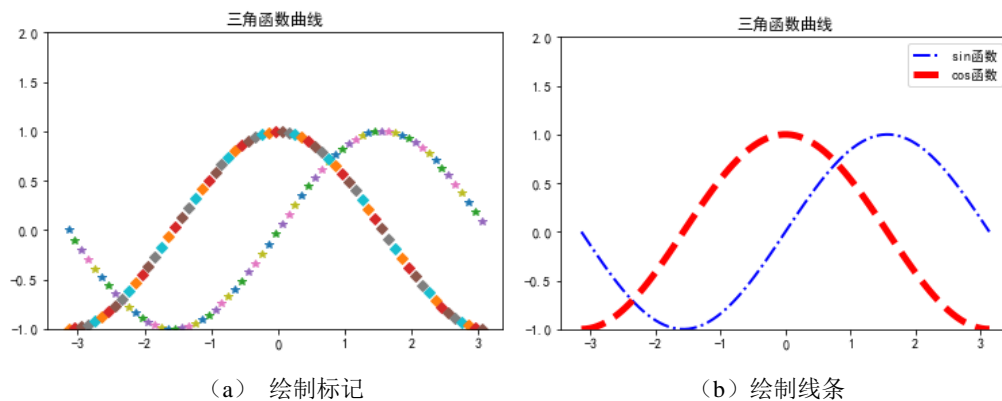


图 1.8.8 三角函数的曲线图

【例 8.3】绘制中国 1979 年至 2018 年的国民生产总值（GDP）的变化趋势图，统计数据见表 1.8.6，将所有数据存放到“gdp.txt”数据文件中，绘制的图表效果如图 1.8.9 所示。

分析：折线图适合显示随时间变化的连续数据，类别沿水平轴均匀分布，垂直轴方向显示相应的数据值，一般调用 plot 函数绘制折线图，也可以调用 scatter 函数绘制散点图，当点比较密集时，就可以达到折线图的效果。

表 1.8.6 1979~2018 年中华人民共和国国内生产总值（GDP）统计数据

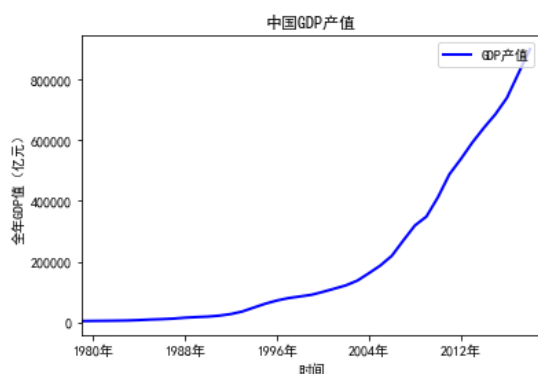
时间	全年 (亿元)	时间	全年 (亿元)	时间	全年 (亿元)	时间	全年 (亿元)
2018 年	900309	2008 年	319245	1998 年	85196	1988 年	15180
2017 年	820754	2007 年	270092	1997 年	79715	1987 年	12175
2016 年	740061	2006 年	219439	1996 年	71814	1986 年	10376
2015 年	685993	2005 年	187319	1995 年	61340	1985 年	9099
2014 年	641281	2004 年	161840	1994 年	48638	1984 年	7279
2013 年	592963	2003 年	137422	1993 年	35673	1983 年	6021
2012 年	538580	2002 年	121717	1992 年	27195	1982 年	5373
2011 年	487940	2001 年	110863	1991 年	22006	1981 年	4936
2010 年	412119	2000 年	100280	1990 年	18873	1980 年	4588
2009 年	348518	1999 年	90564	1989 年	17180	1979 年	4101

方法 1：读取文本文件“gdp.txt”中的数据（数据中无标题行）并绘制图表，效果如图 1.8.9（a）所示。

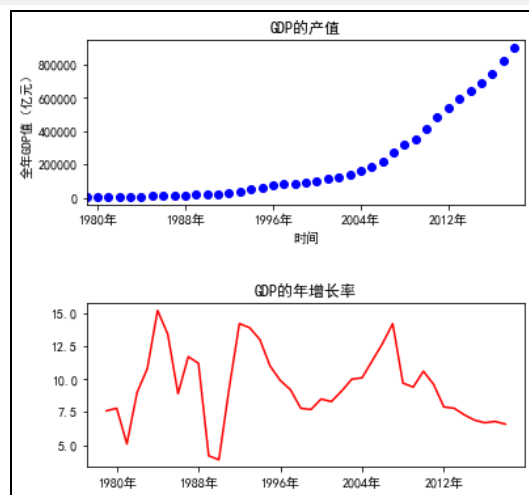
参考代码如下：

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']          #设置中文字体
fp=open("gdp.txt","r")
dt=fp.readlines()                                #从文本文件中读取全部 GDP 数据到列表中
n=len(dt)                                       #获得列表中记录个数
gdpYear=[' ']*n                                #初始化年份列表
gdpValue=[0]*n                                #初始化产值列表
for i in range(0,n):
    dt[i]=dt[i].rstrip('\n')                    #去除每一行字符串末尾的换行符
    dt[i]=dt[i].split('\t')                     #按制表符将年份和产值分隔开
    gdpYear[i]=dt[i][0]                          #将年份列表取出
    gdpValue[i]=int(dt[i][1])                    #将 GDP 产值列表取出并转换成整型
```

```
plt.title("中国 GDP 产值")           #设置图表的标题
plt.xlabel('时间')                   #设置 X 轴的标题
plt.ylabel("全年 GDP 值（亿元）")    #设置 Y 轴的标题
plt.xlim(0,40)                       #设置 X 轴的取值范围
plt.xticks(range(1,40,8))            #设置 X 轴的刻度
plt.plot(gdpYear,gdpValue,color='b',linewidth=2,label="GDP 产值")
plt.legend(loc='upper right')         #显示图例
plt.show()
```



(a) 绘制折线



(b) 绘制散点图和折线图

图 1.8.9 中国 GDP 产值和增长率图

方法 2: 引入 pandas 库, 读取电子表格文件 “gdp-1.xlsx” 中的数据, 该文件增加了 GDP 的年增长率数据, 用 2 个子图分别绘制 GDP 产值的散点图以及年增长率的折线图, 效果如图 1.8.9 (b) 所示。

注意: 电子表格中的第一行为数据的标题行, 分别为年份、产值和增长率, 利用 pandas 库提供的 DataFrame 结构可以很方便地读取到表格中每一行和每一列的数据。本例中首先用 `import pandas as pd` 导入 pandas 库, 然后用 `dt=pd.read_excel("gdp-1.xlsx")` 来读取电子表格文件中的数据, 最后通过 `dt['年份']`, `dt['产值']`, `dt['增长率']` 分别获取表格中的三列数据。

参考代码如下:

```
import matplotlib.pyplot as plt
import pandas as pd
plt.rcParams['font.sans-serif']=['SimHei']
dt=pd.read_excel("gdp-1.xlsx")
fig,axes = plt.subplots(nrows=2,ncols=1,figsize=(6,6))
plt.subplots_adjust(hspace=0.6)
plt.subplot(211)
plt.title("GDP 的产值")
plt.xlabel('时间')
plt.ylabel("全年 GDP 值（亿元）")
plt.xlim(0,40)
plt.xticks(range(1,40,8))
plt.scatter(dt['年份'],dt['产值'],color='b')
plt.subplot(212)
#导入 pandas 库, 后面用 pd 别名来引用
#设置中文字体
#从电子表格文件中读取 GDP 数据
#调节子绘图区之间的垂直间距
#绘制在第 1 个子绘图区
#设置子图表 1 的标题
#设置子图表 1 的 X 轴标题
#设置子图表 1 的 Y 轴标题
#设置子图表 1 的 X 轴取值范围
#设置子图表 1 的 X 轴的刻度
#绘制 GDP 产值的散点图
#绘制在第 2 个子绘图区
```



<code>plt.title("GDP 的年增长率")</code>	<code>#设置子图表 2 的标题</code>
<code>plt.xticks(range(1,40,8))</code>	<code>#设置子图表 2 的 X 轴的刻度</code>
<code>plt.plot(dt['年份'],dt['增长率'],color='r')</code>	<code>#绘制 GDP 增长率的折线图</code>
<code>plt.show()</code>	

## 8.2.2 绘制饼图

`pie` 函数用于绘制饼图，饼图适合显示一个数据系列中各项数值与其总和的占比关系，`pie` 函数的使用形式为：

```
plt.pie(x,explode=None)
```

说明：

① `x` 是一个列表或元组序列，`x` 中的每一个数值与其总和的比例等于饼图中对应扇形块所占的比例。

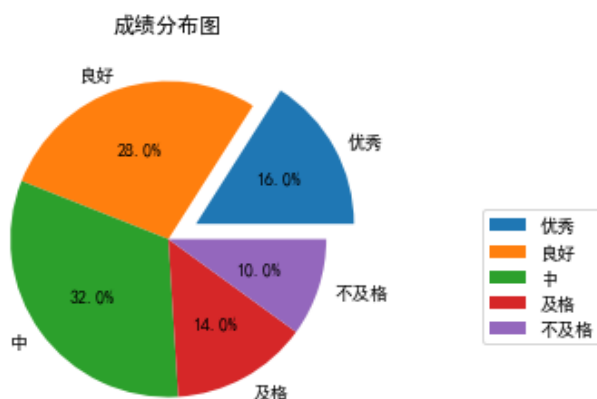
② `explode` 也是一个列表或元组序列，其长度与参数 `x` 一致，其中每个数值用于设置对应扇形块从整个饼图中分裂出来的偏移量。

**【例 8.4】**绘制一个饼图，显示各等级成绩的占比情况，效果如图 1.8.10 所示。

分析：首先需要准备好各等级成绩所占百分比的数据，然后设置图表中各个组成元素的值，比如图表标题，每一块的颜色，每一块分裂出来的偏移量列表，图例的标签文本及图例显示位置，数据标记值及其显示格式等。

参考代码如下：

<code>import matplotlib.pyplot as plt</code>	
<code>score = [0.16,0.28,0.32,0.14,0.1]</code>	<code>#各等级的成绩数据</code>
<code>plt.rcParams['font.sans-serif']='[SimHei]</code>	<code>#设置中文字体</code>
<code>plt.title("成绩分布图")</code>	<code>#图表标题</code>
<code>labels = ['优秀','良好','中','及格','不及格']</code>	<code>#数据标记</code>
<code>explode = (0.2, 0, 0, 0, 0)</code>	<code>#第 1 块分裂出来</code>
<code>plt.axis("equal")</code>	
<code>plt.pie(score, explode=explode, autopct='%4.1f%%', labels=labels)</code>	
<code># autopct='%4.1f%%'控制数据标记的格式化显示，显示 1 位小数</code>	
<code>plt.legend(bbox_to_anchor=(1, 0.6))</code>	<code>#显示图例</code>
<code>plt.show()</code>	



## 8.2.3 绘制散点图

`scatter` 函数用于绘制散点图，散点图适合展现两个变量之间的关系，用两组数据构成多个坐标点，通过坐标点的分布来判断两个变量之间是否存在某种关联，或分析坐标点的分布情况。`scatter` 函数的使用形式为：

```
plt.scatter(x, y, s=None, c=None, marker=None, cmap=None, **kwargs)
```

说明：

① `x` 和 `y` 通常是列表或元组等序列。`x` 中存储所有顶点的 `x` 坐标序列，`y` 中存储所有顶点的 `y` 坐标序列。

② 参数 `s` 用于指定标记大小，默认为 36.0。如果指定 `s=10`，那么所有标记点的大小都是 10。如果要显示不同大小的标记，可以给 `s` 赋值一个列表或元组序列，其长度必须与标记点的数量一致。

例如：

```
import matplotlib.pyplot as plt
x=[1,5,9,3]; y=[6,13,20,32]          #设置 4 个顶点的坐标序列
plt.scatter(x,y,s=(36,120,60,10))     #4 个标记的大小不同
```

③ 参数 `c` 用于标记每个顶点的颜色，如果 `c='r'`，表示所有标记的颜色都是红色。如果要显示不同颜色的标记，则给 `c` 赋值一个颜色字符串或者一个表示颜色的序列或元组，其中字符串的长度或序列的长度必须与标记点的数量一致。

例如：

```
import matplotlib.pyplot as plt
from numpy import random
x=random.randint(10, size=5)          #随机生成 5 个数值赋给 x 坐标序列
y=random.randint(20, size=5)          #随机生成 5 个数值赋给 y 坐标序列
plt.scatter(x,y,c='rgbgb')            #5 个标记的颜色分别为“红绿蓝绿蓝”
```

注意：参数 `c` 如果赋值一个表示颜色的字符串，每个字母必须对应表示的是一种颜色，`c='rgbgb'` 可以改写成 `c=['r','g','b','g','b']`。如果同时有参数 `cmap="rainbow"`（指定为彩虹色系），则颜色由 `c` 参数决定，`cmap` 不起作用。

参数 `c` 也可以赋值一个数值列表或元组，相同数值用相同的颜色显示，这时可以通过参数 `cmap` 来指定一种配色方案，如 `plt.scatter(x,y,c=[0,0,2,0,1],cmap='ocean')`。

在聚类信息可视化时可以将表示类别信息的标注值序列赋给参数 `c`，使同一个类别的顶点用相同的颜色来显示。

**【例 8.5】** 绘制一个散点图，显示鸢尾花数据集中的数据分布情况。

分析：iris 数据集由 Fisher 于 1936 年收集整理，是常用的分类实验数据集，包含 3 类鸢尾花（`setosa`：山鸢尾花，`versicolor`：变色鸢尾花，`virginica`：维吉尼亚鸢尾花），共 150 个数据样本，每个类别有 50 条数据记录，每条记录包含 4 个属性，分别是 `sepal length(cm)`：花萼长度，`sepal width (cm)`：花萼宽度，`petal length (cm)`：花瓣长度，`petal width (cm)`：花瓣宽度。

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"]=["SimHei"]
```

```

iris=load_iris()           #载入鸢尾花数据集
dt=iris.data               #获得数据集中 150 条数据记录
tg=iris.target             #tg 用于存放每个样本的类别数据
x=dt[:,0]                  #x 存放所有样本的花萼长度
y=dt[:,2]                  #y 存放所有样本的花瓣长度
label="山鸢尾花\n 变色鸢尾花\n 维吉尼亚鸢尾花"           #图例文本
plt.scatter(x,y,c=tg,marker="^",cmap= "viridis",label=label)   绘制散点图
plt.legend()
plt.show()

```

上面代码的运行效果如图 1.8.11 所示。

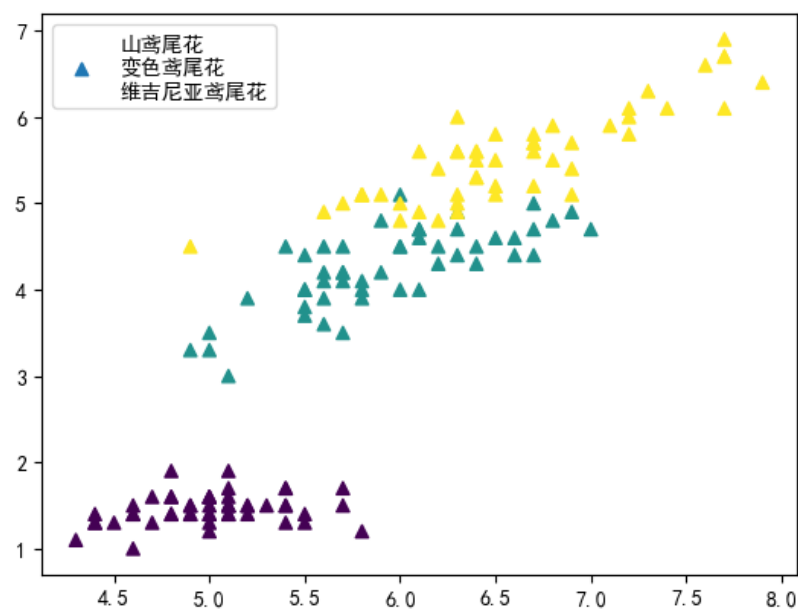


图 1.8.11 散点图

## 8.2.4 绘制直方图

`hist` 函数用于绘制直方图，直方图是一种统计报告图，用一系列高度不等的纵向条纹或线段表示数据的分布情况，一般用横轴表示数据类型，纵轴表示当前类别的统计结果。`hist` 函数的使用形式为：

```
plt.hist(x,bins=None,range=None, ,rwidth=None, **kwargs)
```

说明：

- ① `x` 是一个列表或元组序列，直接存放原始数据序列。
  - ② `bins` 参数指定直方图条形的个数，对应原始数据的统计类别个数，默认值为 10，每个条形图的高度对应当前类别的统计结果。
  - ③ `range` 参数指定直方图在 X 轴上数值的显示范围。
  - ④ `rwidth` 参数指定每个条形的宽度。
- 例如，随机生成 50 个 100 以内的整数，显示 30 到 100 之间的数据分布情况。

```

import matplotlib.pyplot as plt
from numpy import random
data=random.randint(100,size=50)

```

```
plt.hist(x=data,bins=10,range=[30,100],rwidth=0.75)
```

程序的运行结果如图 1.8.12 所示。

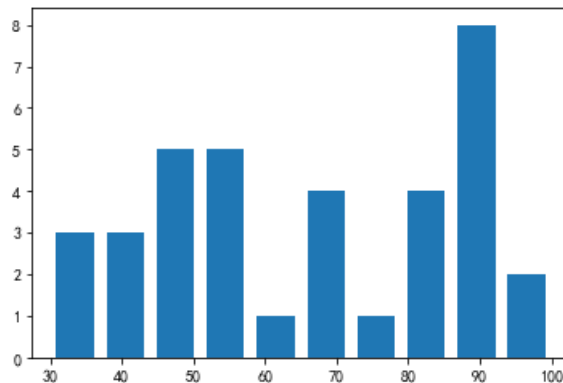


图 1.8.12 随机生成 50 个整数的直方图

【例 8.6】绘制指定图像文件的直方图，效果如图 1.8.13 所示。

分析：图像直方图的横轴表示灰度等级数（0 表示最黑，255 表示最白），纵轴的高度表示对应当前灰度等级像素点数目多少，常用于分析图像的灰度分布是否合理，Photoshop 图像处理软件中提供了“色阶”调整，显示了图像直方图，对于缺乏层次感的图像（该亮的地方不亮，该暗的地方不暗），可直观地调整该直方图的暗部与亮部，达到增强图像对比度的目的。

要绘制指定图像文件的直方图，首先打开图像文件，读取到每一个像素点的数据，保存到一个三维数组 `im` 中，其中第 1 维和第 2 维定位像素点的坐标，第 3 维对应各个颜色通道的值，如 `im[i,j,0]` 保存的是第 `i` 行第 `j` 列的红色通道值，`im[i,j,1]` 保存的是第 `i` 行第 `j` 列的绿色通道值，`im[i,j,2]` 保存的是第 `i` 行第 `j` 列的蓝色通道值。针对 RGB 彩色图像，可以采用下面的公式计算每个像素点的灰度值：

$$\text{Gray} = R \cdot 0.299 + G \cdot 0.587 + B \cdot 0.114$$

注意：灰度值的变化范围为[0,255]，总共有 256 个灰度等级。

参考代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image #导入 Image 库
im=np.array(Image.open("Tulips.jpg"))
n=int(im.size/3 )
hd=n*[0] #hd 列表用于存放每个像素点的灰度值
row=im.shape[0]
col=im.shape[1]
for i in range(row):
    for j in range(col):
        v=im[i,j,0]*0.299+im[i,j,1]*0.587+im[i,j,2]*0.114#计算当前像素点的灰度
        hd[i*col+j] =v #保存每一个灰度值
plt.hist(hd,256,color='k') #绘制直方图
# hd: 各像素点的灰度值列表，256 是横轴灰度等级数，color 设置填充黑色
plt.show()
```

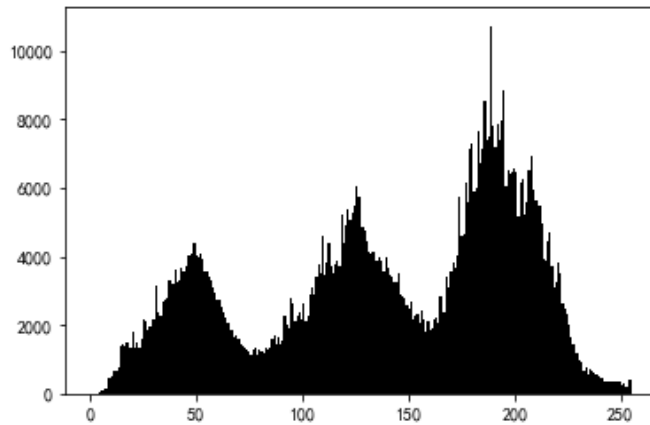


图 1.8.13 图像的直方图

## 8.3 三维图表

三维图表包含了 xyz 三个坐标轴，可以绘制三维立体图形，如三维线条、三维曲面、三维几何体等。

### 8.3.1 三维图表基础

#### 1. 三维绘图子库

Matplotlib 的三维图表主要由 mplot3d 模块实现，由于绘制的三维图表是在二维画布上展示，所以同样需要导入 pyplot 子库。在 mpl\_toolkits.mplot3d.axes3d 子库中包含了实现各种三维图表的类和方法，所以绘制三维图表前，一般用下面语句导入三维绘图子库。

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
或 from mpl_toolkits.mplot3d import Axes3D
```

#### 2. 三维绘图区

与二维图表绘制过程一样，首先调用 figure() 函数创建一个绘图区，然后调用 Axes3D() 函数创建三维坐标系；或者通过 gca 函数获取当前坐标系，并通过参数 projection='3d' 指定绘制的是三维图表。

例如：

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()                    #创建全局绘图区
ax = Axes3D(fig)                     #创建三维坐标系
plt.show()
```

程序执行结果如图 1.8.14 所示。

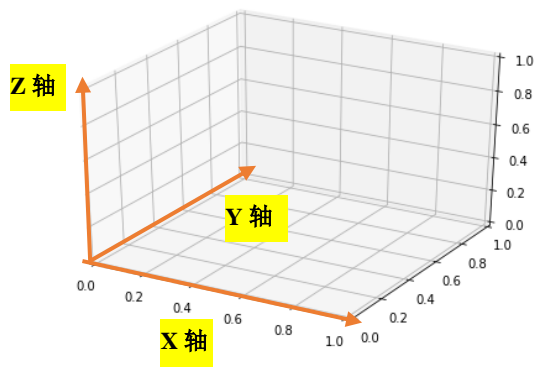


图 1.8.14 三维绘图区及其坐标系

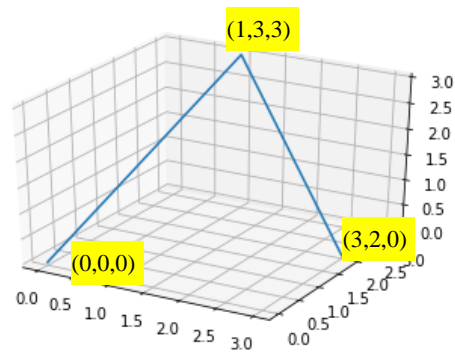


图 1.8.15 绘制三维折线

或者改用 `gca` 函数设置三维坐标系风格，下面程序代码执行效果如图 1.8.15 所示。

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot((0,1,3),(0,3,2),(0,3,0))
plt.show()
```

#创建全局绘图区  
#设置三维坐标轴  
#绘制一条三维折线

### 3. 三维子绘图区

三维绘图区中可以添加多个三维子绘图区，与二维子绘图区的添加方法基本一样，都是调用 `add_subplot` 函数，但是必须增加一个函数参数 `projection='3d'`。

例如：

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(12,5))
ax1 = fig.add_subplot(121,projection='3d')
ax1.plot((0,3),(0,3),(0,3),ls='--',lw=5)
ax2 = fig.add_subplot(122,projection='3d')
ax2.plot((0,3,0),(3,3,0),(3,0,0),c='r',marker="D")
plt.show()
```

#创建全局绘图区  
#添加三维子绘图区 1  
#绘制三维线条  
#添加三维子绘图区 2  
#绘制三维折线与标记

上面代码执行效果如图 1.8.16 所示。

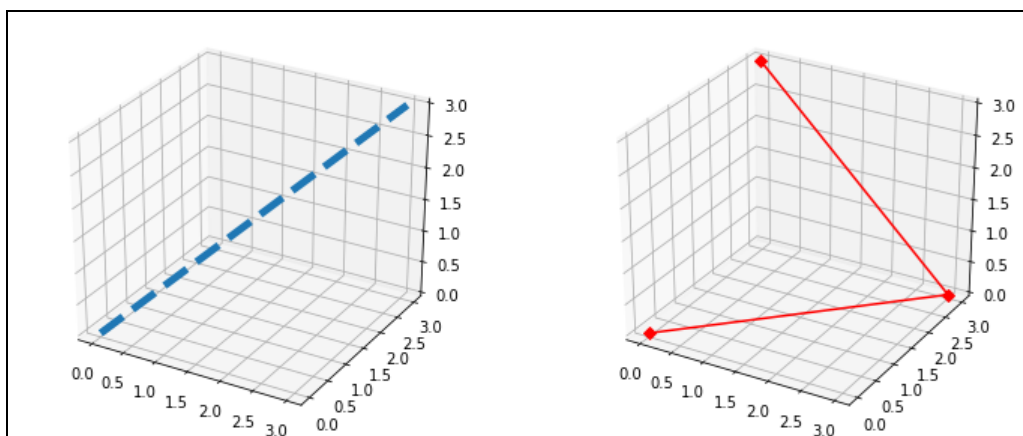


图 1.8.16 三维子绘图区

## 8.3.2 绘制三维图表

### 1. 绘制三维线条

三维线条的绘制方法与二维图表中的线条绘制方法基本一样,调用 `plot` 函数时需要给出 `x`、`y`、`z` 三个维度方向的坐标序列。

【例 8.7】绘制一个三维螺旋线,效果如图 1.8.17 所示。

参考代码如下:

```
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
fig=plt.figure()
ax=Axes3D(fig)
#下面生成螺旋线在 xyz 三个坐标维度上的坐标点序列
z=np.arange(-5*np.pi,5*np.pi,0.1)
x,y=np.sin(z), np.cos(z)
#下面设置三个轴向的刻度范围与刻度标记个数
ax.set_xticks(np.linspace(-1,1,4))
ax.set_yticks(np.linspace(-1,1,3))
ax.set_zticks(np.linspace(-20,20,5))
plt.plot(x,y,z,c='b')          #绘制三维线条
plt.show()
```

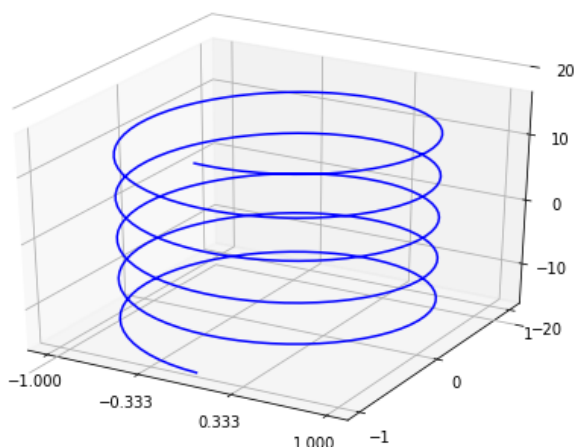


图 1.8.17 三维螺旋线

### 2. 绘制三维对象

Matplotlib 通过不同的填充方式来绘制不同的三维实体对象。如填充三角面、填充四边形或填充网格面等。

【例 8.8】绘制一个四棱锥,效果如图 1.8.18 所示。

分析: 首先给出四棱锥的 5 个顶点的坐标, 然后调用函数 `plot_trisurf()`, 用三角面进行填充即可绘制出一个四棱锥。

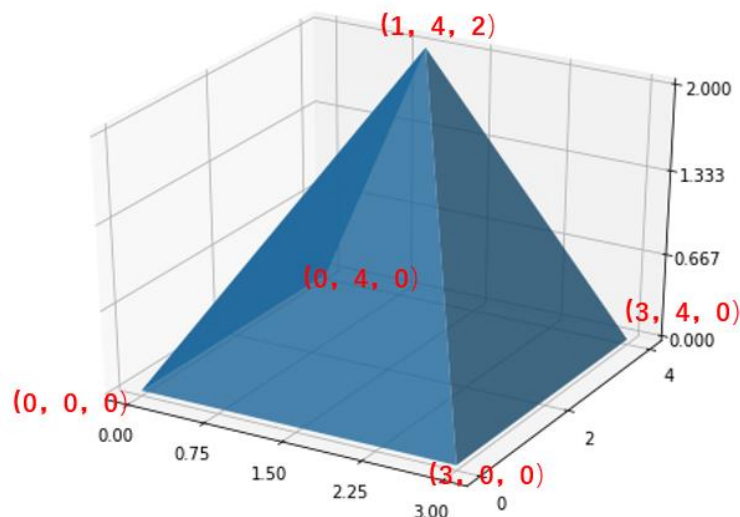


图 1.8.18 四棱锥

参考代码如下：

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x= [0,3,3,0,1]           #指定 XYZ 三个轴向 5 个坐标点的值
y= [0,0,4,4,4]
z= [0,0,0,0,2]
ax.set_xticks(np.linspace(0,3,5))    #设置三个轴向的刻度范围
ax.set_yticks(np.linspace(0,4,3))
ax.set_zticks(np.linspace(0,2,4))
ax.plot_trisurf(x, y, z,alpha=0.8)  #绘制 5 个顶点并以三角面填充
plt.show()
```

**【例 8.9】** 绘制一个空心的立方体，由顶底前后左右六个面组成。

分析：首先需要创建该立方体各个顶点的坐标，然后再调用 `plot_surface()` 将 6 个面绘制出来，最终形成一个空心的立方体。

① 导入必要的库，调用 `numpy` 中的函数生成 `xy` 平面上的 25 个网格坐标点。

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(1,5,5)      #生成的 x 序列为: array([1., 2., 3., 4., 5.])
y=np.linspace(1,5,5)      #生成的 y 序列为: array([1., 2., 3., 4., 5.])
xx,yy=np.meshgrid(x,y)    #调用 meshgrid 生成 xy 平面上的 25 个网格坐标点
plt.plot(xx,yy,"bo")      #用蓝色圆点标注各个坐标点的位置
plt.show()
```

以上代码执行的结果如图 1.8.19 所示。



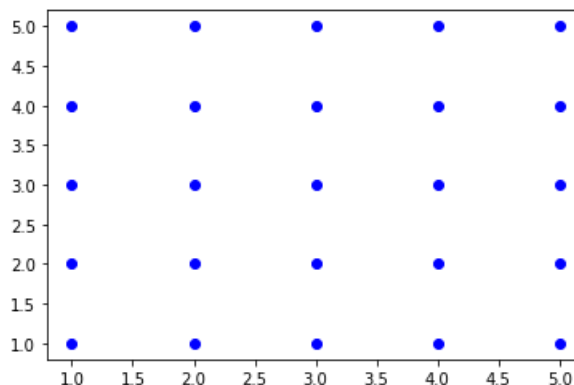


图 1.8.19 网格坐标点

- ② 创建一个三维坐标系的绘图区，设置各个坐标轴的刻度范围

```
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure()
ax=Axes3D(fig)
ax.set_xlim(1,5)          #设置三个坐标轴向的刻度范围
ax.set_ylim(1,5)
ax.set_zlim(1,5)
```

- ③ 生成 xy 平面上 25 个网格点坐标对应的 z 坐标的值

```
z = np.zeros(25)+5      #z 的值是 array([5., 5. .... 5.]), 共 25 个
zz= z.reshape(5, 5)    #将 z 修改为二维向量
```

- ④ 绘制立方体的 6 个面

```
ax.plot_surface(xx, yy, zz, color='r', alpha=0.7)    #顶面
ax.plot_surface(xx, yy, zz-4,color='g', alpha=1 )    #底面
ax.plot_surface(zz-4,xx, yy, color='b', alpha=1)    #后面
ax.plot_surface(zz, xx, yy, color='c', alpha=0.7)    #前面
ax.plot_surface(yy, zz, xx, color='m', alpha=1 )    #右面
ax.plot_surface(yy, zz-4,xx, color='y', alpha=0.7)    #左面
```

以上代码执行的结果如图 1.8.20 所示。

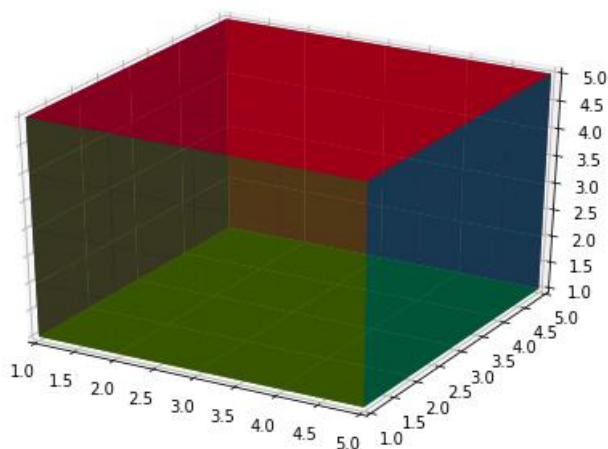


图 1.8.20 空心立方体

【例 8.10】绘制一个由网格面组成的三维几何体对象。

分析：首先生成 XY 平面上的网格点坐标，利用函数  $z=\sin(\sqrt{x^2+y^2})$  获得 Z 轴上对应

的坐标值，再调用 `plot_surface()` 绘制网格面，输出一个三维几何体。

参考代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm #颜色风格控制库
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
ax.set_xlim(-5,5)
ax.set_ylim(-5,5)
ax.set_zlim(-1,1)
X,Y = np.arange(-5, 5, 0.1),np.arange(-5, 5, 0.1)
X,Y = np.meshgrid(X, Y) #形成 XY 平面上的网格点
Z = np.sin(np.sqrt(X**2 + Y**2)) #生成 Z 轴方向对象的坐标值
ax.plot_surface(X,Y,Z,rstride=3,cstride=5,cmap=cm.rainbow)
#rstride 与 cstride 控制每个网格面行列方向的跨度大小
plt.show()
```

以上代码执行的结果如图 1.8.21 所示。

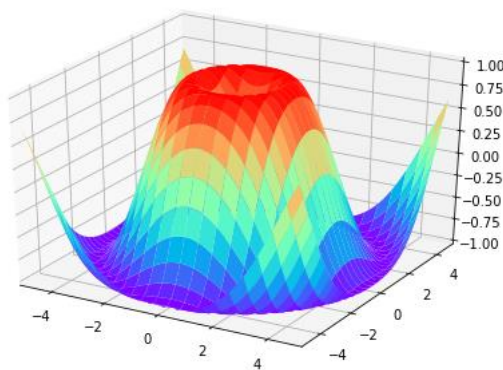


图 1.8.21 三维网格对象

## 8.4 综合应用

本章主要介绍了 Python 语言中 Matplotlib 绘图库的使用，包括图表的组成，绘图区的控制，二维图表和三维图表的绘制方法等，下面综合运用本章所学内容，制作所需图表。

【例 8.11】请分别用 `plot` 函数和 `scatter` 函数绘制如图 1.8.22 所示的图案，生成图案的参数方程为：

$$\begin{cases} r=\sin(a*\theta) \\ x=r*\cos(\theta) \\ y=r*\sin(\theta) \end{cases}$$

其中： $\theta$  是  $[0, 2\pi]$  之间的弧度值， $a$  可以是一个任意实数， $a$  的取值决定了图案的形状。

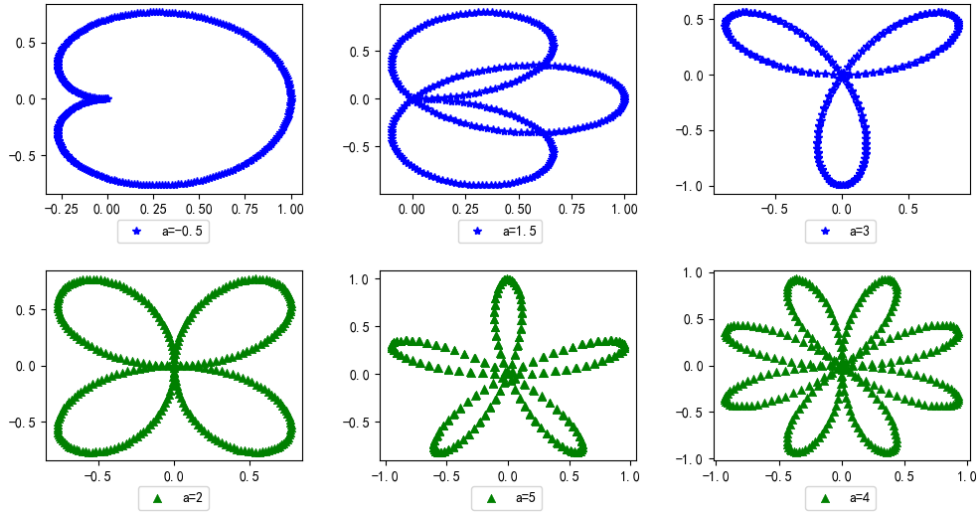


图 1.8.22 绘制图案

方法 1: 用标准的数学库函数来实现, 利用循环控制生成每一个坐标点的值。

参考代码如下:

```
import math
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False #正常显示负号
fig,ax = plt.subplots(nrows=2,ncols=3,figsize=(12,6)) #设置子绘图区
plt.subplots_adjust(wspace=0.3,hspace=0.4) #调整子绘图区的间隔
a= [-0.5,1.5,3] #设置 a 的取值序列, 控制第 1 行的三个图案
for i in range(3): #控制第 1 行的三个子绘图区
    th=0; xx=[]; yy=[]
    while th<=2 * math.pi: #用循环控制生成所有坐标点的取值序列
        r=math.sin(a[i]*th)
        xx.append(r*math.cos(th))
        yy.append(r*math.sin(th))
        th+=0.03
    ax[0][i].plot(xx,yy,'b*',label="a=" + str(a[i])) #用 plot 函数绘制标记
    ax[0][i].legend(bbox_to_anchor=( 0.7,-0.1)) #指定图例位置
a=[2,5,4]
for i in range(3): #控制第 2 行的三个子绘图区
    th=0; xx=[]; yy=[]
    while th<=2* math.pi:
        r=math.sin(a[i]*th)
        xx.append(r*math.cos(th))
        yy.append(r*math.sin(th))
        th+=0.03
    ax[1][i].scatter(xx,yy,c="g",lw=0.5,marker="^",label="a=" + str(a[i]))
    #用 scatter 函数绘制散点图
    ax[1][i].legend(bbox_to_anchor=( 0.7,-0.1))
```

方法 2: 引入 numpy 库, 快速生成数据序列并绘制图表。

参考代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False      #正常显示负号
fig,ax = plt.subplots(nrows=2,ncols=3,figsize=(12,6))
plt.subplots_adjust(wspace=0.3,hspace=0.4)
a= [-0.5,1.5,3]
th=np.arange(0,2*np.pi,0.02)                #快速生成弧度值序列
for i in range(3):
    r=np.sin(a[i]*th)
    x,y=r*np.cos(th),r*np.sin(th)             #计算坐标点序列
    ax[0][i].plot(x,y,'b*',label="a=" + str(a[i])) #绘制坐标点标记
    ax[0][i].legend(bbox_to_anchor=( 0.7,-0.1))   #通过图例显示当前 a 的取值
a=[2,5,4]
for i in range(3):
    r=np.sin(a[i]*th)
    x,y=r*np.cos(th),r*np.sin(th)
    ax[1][i].scatter(x,y,c="g",lw=0.5,marker="^",label="a=" + str(a[i]))
    ax[1][i].legend(bbox_to_anchor=( 0.7,-0.1))
```

【例 8.12】绘制微信好友性别统计柱状图和好友签名的词云图。

分析：词云就是对网络文本中出现频率较高的关键词予以视觉上的突出，渲染出符合指定形状的关键词分布图像，从而过滤掉大量的文本信息，使用户可以快速领略文本的主旨大意，下面一起学习词云图的制作。

本例需要用到以下几个库：itchat（用于登录微信）、jieba（用于分词处理）、wordcloud（用于生成词云图）、PIL（用于图像处理）。

先登录自己的微信账号，获取所有朋友信息，提取需要的签名数据，然后利用结巴库进行分词处理，再用 wordcloud 生成词云图并输出图像。

制作过程：

① 导入必要的库

```
import itchat
import numpy as np
import pandas as pd
import jieba
import os
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import PIL.Image as Image
```

② 登录微信，获取所有朋友的数据

```
itchat.login()          #弹出微信二维码图片，在手机上扫码并确认登录
friends = itchat.get_friends(update=True)    #获取好友信息
```

③ 获取自己的昵称，在当前目录下创建一个用自己昵称命名的文件夹，然后将其设置为当前文件夹

```
NickName = friends[0].NickName#获取自己的昵称
```

```

cdr = os.getcwd()                #获取当前目录
if not(os.path.exists(NickName)): #判别昵称文件夹是否已存在
os.mkdir(NickName)                #则创建一个名字为昵称的新文件夹
os.chdir(cdr+'\\' + NickName)    #切换到昵称文件夹

```

④ 获取好友的性别信息，统计各性别人数，然后绘制各性别好友数量的柱状图

```

df = pd.DataFrame(friends)        #把好友数据处理成 DataFrame
Sex = df.Sex                      #获取好友性别信息
Sex_count = Sex.value_counts()    #统计人数，男：1、女：2、未知：0
plt.figure(figsize=(10,5))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.subplot(121)
x=[0,1,2]
plt.xticks(x,("男",'女','未知'))
barW=0.5
plt.bar(x,Sex_count,barW,color='b',label='性别') #绘制柱状图
plt.legend(loc='best')

```

⑤ 获取好友签名信息，并将其写入文本文件中保存

```

Signatures = df.Signature        #获取好友签名信息
text = ''.join(Signatures)        #将好友签名数据保存到 text 变量中

```

#下面将所有好友的签名信息写入昵称文件夹下的 sig.txt 文件中

```

file_name = NickName+'sig.txt'
with open(file_name,'w',encoding='utf-8') as f:
    f.write(text)
f.close()

```

⑥ 用结巴库对好友签名数据进行分词处理

```

wordlist=jieba.cut(text,cut_all=True) #对好友签名信息进行分词处理
wordSS = ''.join(wordlist)           #将分词结果转换成字符串

```

⑦ 设置词云图的各项参数并生成词云图，保存为图片文件

```

mask = np.array(Image.open('wx.png')) #定义词云图显示的遮罩形状
wcd = WordCloud(
    font_path='C:/Windows/Fonts/simhei.ttf',
    background_color='white',
    mask=mask,                #设置遮罩图片
    max_words=1000,           #最多显示词数
    max_font_size=100 )       #字体最大值
myWD = wcd.generate(wordSS)   #根据签名数据生成词云图
myWD.to_file(NickName+'.jpg') #将词云图保存为图片文件
plt.subplot(122)
plt.imshow(myWD)             #显示词云图
plt.axis('off')               #不显示坐标轴
plt.show()                    #显示图像

```

以上代码执行的结果如图 1.8.23 所示。



## 习题

### 一、选择题

1. 创建并显示 2 行 3 列共 6 个子绘图区，以下代码正确的是\_\_\_\_\_。
  - A. `fig, axes = plt.subplot(nrows=2, ncols=3, figsize=(4,3), dpi=100)`
  - B. `fig, axes = plt.subplots(nrows=2, ncols=3)`
  - C. `fig, axes = plt.subplots(nrows=3, ncols=2)`
  - D. `fig, axes = plt.subplot(nrows=2, ncols=3)`
2. 语句 `plt.plot([1,5], c='g')` 绘制的是\_\_\_\_\_。
  - A. 从点 (0, 0) 到点 (1, 5) 的一条绿色直线
  - B. 从点 (0, 0) 到点 (1, 5) 的一条绿色曲线
  - C. 从点 (1, 5) 到点 (0, 0) 的一条绿色直线
  - D. 从点 (0, 1) 到点 (1, 5) 的一条绿色直线
3. 绘制一条从 (1, 1) 到 (2, 3) 最后到 (3, 2) 的折线，正确的语句是\_\_\_\_\_。
  - A. `plt.plot((3,2),(2,3),(1,1))`
  - B. `plt.plot([1,3,2],[1,2,3])`
  - C. `plt.plot((1,1),(2,3),(3,2))`
  - D. `plt.plot([1,2,3], [1,3,2])`
4. 语句 `plt.plot((3,5),(0,1), c='b')` 绘制的是[ ]。
  - A. 从点 (3, 5) 到点 (0, 1) 的一条蓝色直线
  - B. 从点 (3, 5) 到点 (0, 1) 的一条蓝色曲线
  - C. 从点 (3, 0) 到点 (5, 1) 的一条蓝色直线
  - D. 从点 (0, 3) 到点 (1, 5) 的一条蓝色直线
5. 在 Matplotlib 中，以下\_\_\_\_\_用于绘制饼图。
  - A. `plt.scatter()`
  - B. `plt.hist()`
  - C. `plt.pie()`
  - D. `plt.bar()`
6. 绘制一个散点图，只有 (1, 1)、(2, 3) 和 (3, 5) 三个点，正确的语句是\_\_\_\_\_。
  - A. `plt.scatter([1,2,3], [1,3,5], c="b", marker="o")`
  - B. `plt.scatter((1,1),(2,3),(3,5), c="b", marker="o")`
  - C. `plt.scatter([1,3,5], [1,2,3], c="b", marker="o")`
  - D. `plt.scatter((3,5),(2,3),(1,1), c="b", marker="o")`
7. 在 matplotlib 中，绘制由三角形构成曲面单元的三维图形时，应使用\_\_\_\_\_方法。
  - A. `plot`
  - B. `scatter`
  - C. `plot_surface`
  - D. `plot_trisurf`
8. 在 Matplotlib 中，以下\_\_\_\_\_用于绘制直方图。
  - A. `plt.plot()`
  - B. `plt.hist()`

- C. plt.pie()  
D. plt.polar()

## 二、填空题

- 1.用于创建一个坐标系风格的绘图区域并显示绘图区的函数是\_\_\_\_\_。
- 2.Matplotlib 的图表区包含三个层次：容器层，辅助显示层和\_\_\_\_\_。
- 3.pyplot 是 Matplotlib 的子库，能够绘制出各种常见的图表类型，包括\_\_\_\_\_和三维图表，还可同时绘制多个子图。
- 4.plot 函数用于绘制\_\_\_\_\_和\_\_\_\_\_，不仅可以绘制折线图、曲线图，还可以制作出类似散点图的效果。
- 5.\_\_\_\_\_适合显示一个数据系列中各项数值与其总和的占比关系。
- 6.请补充下面程序中空白处的代码，绘制一个如图 1.8.24 所示的五角星。

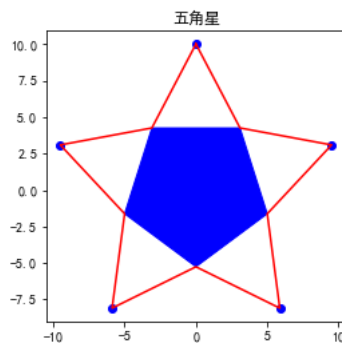


图 1.8.24 五角星

```
import matplotlib.pyplot as plt
import numpy as np
fig=plt.figure(figsize=(4,4),dpi=72)
plt.rcParams["font.sans-serif"]=["SimHei"]
plt.rcParams["axes.unicode_minus"]=False
rw=10                                #假设外接圆的圆心坐标为 (0, 0)，半径为 10
k=np.arange(5)                       #k 设为各个顶点的序号，array([0, 1, 2, 3, 4])
x1=rw*np.sin(2*np.pi/5*k)           #生成外接圆上五个顶点的 x 坐标序列
y1=rw*np.cos(2*np.pi/5*k)           #生成外接圆上五个顶点的 y 坐标序列
plt._____(1)_____                  #用蓝色圆点标记外接圆上的五个顶点
rn=rw*np.sin(np.pi/10)/np.sin(np.pi/5) #计算内接圆的半径
x2=rn*np.sin(2*np.pi/5*k+np.pi/5)    #生成内接圆上五个顶点的 x 坐标序列
y2=rn*np.cos(2*np.pi/5*k+np.pi/5)    #生成内接圆上五个顶点的 y 坐标序列
plt.fill(_____(2)_____)              #用蓝色填充内接五边形
x=[x1[0],x2[0],x1[1],x2[1],x1[2],x2[2],x1[3],x2[3],x1[4],x2[4],x1[0]]
y=[y1[0],y2[0],y1[1],y2[1],y1[2],y2[2],y1[3],y2[3],y1[4],y2[4],y1[0]]
plt._____(3)_____                  #绘制红色的五角星
plt.title("五角星")
plt.show()
```

## 三、简答题

- 1.简述 Matplotlib 可以绘制的常见图表类型。
- 2.简述 Matplotlib 图表的主要组成部分。
- 3.如何在指定的子绘图区中绘制图表？有哪几种方法？



4. 当子绘图区的个数大于 1 时，如何访问指定子绘图区的坐标系？
5. 绘制三维图表时一般用哪些方法生成三维实体对象？
6. 直方图适合分析什么样的数据？图像直方图有什么作用？

## 第九章 访问数据库

数据库技术的应用日益广泛,已成为现代计算机信息系统和计算机应用系统的基础和核心。本章主要介绍通过 Python 编程访问多种类型的关系型数据库,例如 Access、SQLite、MySQL、SQL Server 等。编程时需用到应用程序编程接口(API)连接数据库,并对数据库进行查询、插入、修改和删除等操作。访问 Access 数据库时,可通过 pywin32 模块提供的应用程序接口 ADO(ActiveX Data Objects, ActiveX 数据对象)实现访问;访问 SQLite、MySQL、SQL Server 等数据库时,可通过 Python 标准数据库接口 DB-API 实现,DB-API 为多种数据库系统和数据库接口程序提供一致的访问接口,不同的数据库下载使用对应的不同 DB API 模块,如 sqlite3、pymysql、pymssql 等。

### 9.1 访问 Access 数据库

Access 是开发的小型关系数据库管理系统,操作简便。ADO 是 Microsoft 提出的应用程序接口,可以访问多种数据源,如 Access、MySQL、SQL Server 等。本节主要介绍 Python 通过 ADO 访问 Access 数据库的基本方法和技术。在建立数据库连接之后,不管后端是何种数据库,访问操作方法基本相同。

#### 9.1.1 ADO 基础

ADO 的主要优点是易于使用、高速度、低内存、占用空间少。首先通过引例了解一下 Python 如何通过 ADO 访问 Access 数据库。

##### 1. 引例—数据库的数据显示

**【例 9.1】**编写程序访问 Access 数据库中的 Scores 表,显示表中所有记录。

本章使用 Access 数据库 Class.accdb,包括学生情况表 Students 和成绩表 Scores。其中 Scores 表包括以下字段:学号(短文本型)、课程(短文本型)、成绩(数字型)。通过以下步骤实现数据库中数据表的显示。

```
#① 导入模块
import win32com.client                                # 导入模块
#② 创建 Connection 对象、设置 ConnectionString 属性,并打开连接
conn = win32com.client.Dispatch(r"ADODB.Connection")    # 创建 Connection 对象
#③ 设置 ConnectionString
conn.ConnectionString = 'PROVIDER = Microsoft.ACE.OLEDB.12.0;DATA SOURCE = Class.accdb '
#④ 打开连接
conn.Open()                                             # 打开连接
#⑤ 创建 Recordset 对象,打开数据表遍历读取内容
rs = win32com.client.Dispatch(r'ADODB.Recordset')      # 创建 Recordset 对象
```

```
rs.Open('Scores',conn) # 打开 Scores 表进行内容读取
while not rs.EOF:      # 遍历所有记录
    print(rs.Fields.Item(0).value,rs.Fields.Item(1).value, rs.Fields.Item(2).value) # 读取字段的值并输出
    rs.MoveNext()      # 移动到下一条记录
#⑥ 关闭、删除相关对象
conn.Close()          # 关闭 Connection 对象
rs = None              # 删除 Recordset 对象
```

直接到 pywin32 官网下载安装包，或者通过语句：pip install pywin32 安装 win32com 库后，按照如图 1.9.1 所示的步骤，编写程序访问 Access 数据库：

- ① 导入 win32com.client 模块，利用该模块访问 Access 数据库文件。
- ② 通过 Dispatch()方法创建 Connection 对象。Connection 对象是从数据源获取数据的起点，用于管理一个数据源连接，所有其它的 ADO 对象都是建立在连接对象的基础之上的。
- ③ 设置 Connection 对象的 ConnectionString 属性，其中 Provider 为数据库驱动程序的名称，Data Source 为数据库的路径及文件名、用户名、密码和各种连接选项。
- ④ 使用 Open()方法打开连接，以建立与数据库 Class.accdb 的连接。
- ⑤ 创建 Recordset 对象访问已打开数据库中的 Scores 表，并利用循环配合游标的移动遍历显示其所有记录。
- ⑥ 关闭 Connection 对象，删除 Recordset 对象。

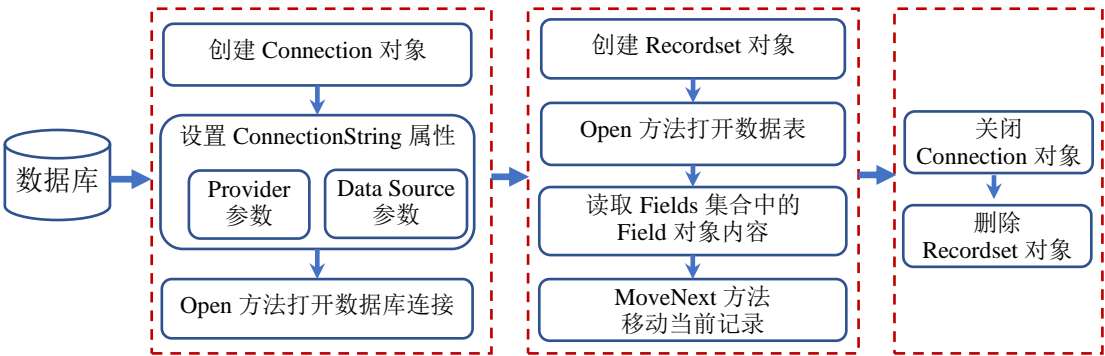


图 1.9.1 例 9.1 访问 Access 数据库的步骤

2. 数据库应用系统的三层结构

数据库应用系统在体系结构上呈现出层次结构，即前台应用程序、中间数据访问层和后台数据库三个层次，之间的关系如图 1.9.2 所示。

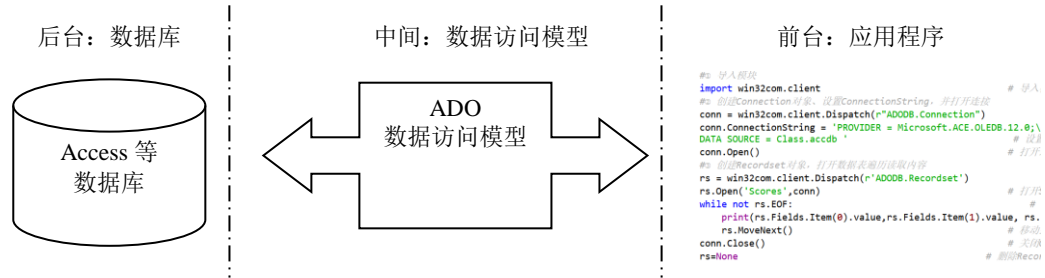


图 1.9.2 数据库应用系统的三层结构

- ① 前台应用程序，如利用 Python 语言所编写的程序，满足一定需求的应用程序功能。
- ② 中间层数据访问层，如 ADO 数据访问模型主要完成前台访问后台数据库的操作，提供两者之间的通信渠道。
- ③ 后台数据库，提供前台应用程序所需要的数据源，执行访问数据源的查询、更新等基本操作。

数据库应用系统的三层体系结构决定了系统的具体设计过程：首先设计数据库，然后使用 ADO 数据访问模型实现前台应用程序功能界面和后台数据库之间通讯，直至所有功能完成。实现数据库访问是我们要解决的关键问题。

### 3. 基本 ADO 对象

ADO 的基本对象有很多种，主要对象如图 1.9.3 所示。

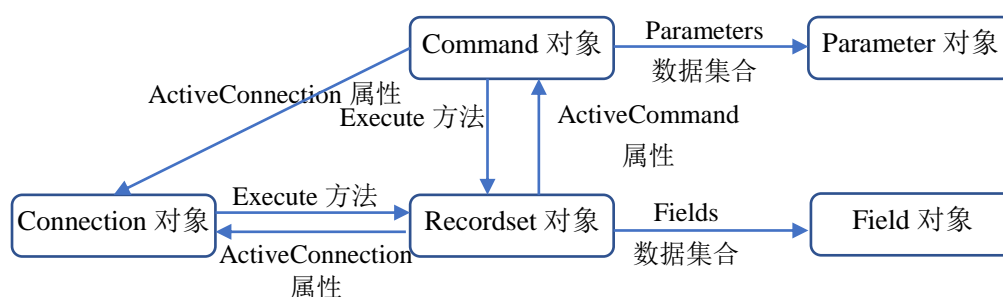


图 1.9.3 ADO 主要对象

1.Connection 对象：用于管理与数据库的连接。在打开 Connection 对象前需要先配置ConnectionString 等属性，以指示它连接到相应类型的数据库。其他 ADO 对象都可以连接到此 Connection 对象，以建立与数据库的连接。另外，也可以使用 Connection 对象的 Execute()方法，通过 SQL 语句进行数据库的查询和维护，返回的查询结果存储在 Recordset 对象中，或无结果时返回关闭的 Recordset 对象。基本过程如图 1.9.4 所示。

2.Recordset 对象：表示一个数据表的记录集合，或者是 SQL 命令的执行结果。Recordset 对象包括了由 Field 对象组成的 Fields 集合，其中每个 Field 对象对应 Recordset 集中的一列。而由于 Recordset 对象所指的当前记录表示集合内的单个记录，所以它通常用于遍历查询或存储过程的结果。使用 Recordset 对象的 ActiveConnection 属性可以说明它关联哪个 Connection 对象。使用其 ActiveCommand 属性可以说明它关联的 Command 对象。另外，Recordset 对象连接到 Connection 对象之后，还可以通过 Open()方法执行查询或存储过程。

3.Field 对象：表示一行结果记录中的列。每个 Field 对象对应于 Recordset 中的一列。使用 Field 对象的 Value 属性可设置或返回当前记录的数据。由于 SQL 查询的结果通常返回多个列，因此 Recordset 对象将包含与列一样多的 Field 对象。Field 对象包含有关它所代表的列的信息，例如列的值、列的类型、字段的最大长度、精度等，使用它的 Value 属性可设置或返回当前记录的数据。

4.Command 对象：定义了对数据源执行的命令。Command 对象通常附加到 Connection

对象，然后执行查询和存储过程。如果要使 SQL 语句具有持久性并用它反复重新执行，或需要使用查询参数的时候，会使用 Command 对象来执行查询及维护。使用 Command 对象运行 SQL 查询时，结果将返回给 Recordset 对象，然后使用该对象进行结果的遍历。

5.Parameter 对象:表示与基于参数化查询的 Command 对象关联的参数或变量。Command 对象可以附加多个 Parameter 对象，并将这些参数的值传送给 Execute()方法，以便反复执行。

### 9.1.2 数据库基本操作

导入 win32com.client 模块后，可以通过 ADO 来访问 Access 数据库。使用 Python 访问 Access 数据库时还要注意位的匹配,即 x86 的 Python 操作 x86 的 Access 文件,x64 的 Python 操作 x64 的 Access 文件。

下面通过实例详细介绍如图 1.9.4 所示的数据库基本查询过程。

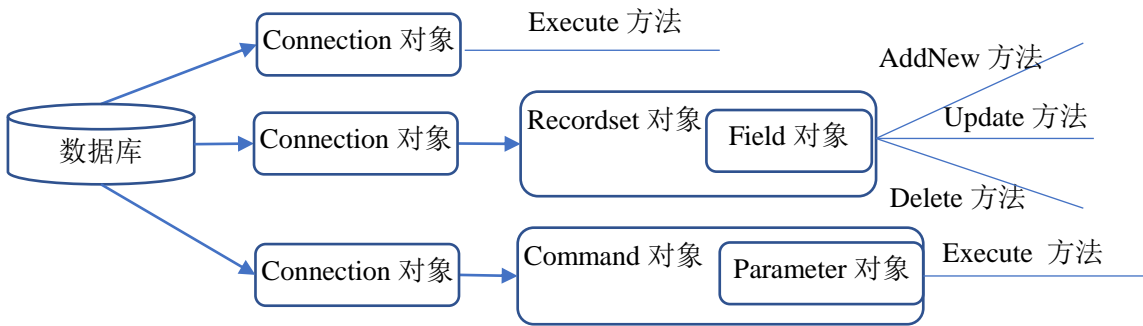


图 1.9.4 ADO 主要对象及数据库查询过程

#### 一、数据库连接

程序操作数据库之前，如何找到数据库？此时需建立程序与数据库之间的连接，可通过创建 Connection 对象来实现。所有其他 ADO 对象都通过 Connection 对象以建立与数据库的连接。

连接数据源时，首先使用 win32com.client 模块的 Dispatch()方法来创建 Connection 对象。然后设置其 ConnectionString 属性，并通过 Open()方法打开数据库连接，完全建立应用程序与数据库之间的连接，并访问 State 属性以判断连接状态，在数据库操作完成后通过 Close()方法关闭连接。

**【例 9.2】**编写程序建立与 Access 数据库 Class.accdb 的连接，并检测连接成功与否且显示其结果，主要代码如下。此处代码中省略了创建并打开 Connection 对象 conn 的语句，以及关闭删除相关对象的语句，参见例 9.1 代码①~④、⑥。后面例 9.3~例 9.8 相同，不再重复说明。

```
if conn.State == 1:                                     # 通过 State 属性判断连接状态
    print("Successfully connected! ")
else:
```

```
print("Failed.")
```

其中，Connection 对象的主要属性和方法如表 1.9.1 和表 1.9.2。

表 1.9.1 Connection 对象的主要属性

属 性	说 明
ConnectionString	设置或返回建立连接数据库的信息。主要包括：Provider：连接的提供者名称；Data Source：连接的数据文件名称（含路径）。
ConnectionTimeout	设置或返回等待命令完成的最大秒数。
State	返回 Connection 对象的连接状态，1 表示打开，0 表示关闭。

表 1.9.2 Connection 对象的主要方法

方 法	说 明
Open([ConnectionString,UserID>Password,Options])	打开与数据源的连接。
Close()	关闭连接。
Execute(CommandText[,RecordsAffected,Options])	执行指定的 SQL 语句等。

注意，使用 Close 方法关闭 Connection 对象时，并非将它从内存中删除；此时仍然可以更改它的属性设置并在以后再次使用 Open 方法打开它。要将对象从内存中完全删除，可将对象变量设置为 None。例如：

```
conn = None
```

Connection 对象的 ConnectionString 属性由 Provider 和 Data Source 两部分组成。对于不同类型的数据，提供者 Provider 也不同，例如 Access 2007 及以上的数据库 Provider 为“Microsoft.ACE.OLEDB.12.0”，而 Access 2007 以前版本的数据库 Provider 为“Microsoft.Jet.OLEDB.4.0”。Data Source 为 Access 文件的路径及文件名，还可以包含数据库的用户名、密码以及其他连接选项。

## 二、记录集操作

使用 ADO 时，可以通过 Recordset 对象对数据进行操作。Recordset 对象表示来自于基本表或命令执行结果的记录全集。所有 Recordset 对象均使用记录（行）和字段（列）进行构造。Recordset 对象一次可以访问一行记录，并可通过 Fields 对象表示该行中的每一列。Recordset 对象还可以通过其具有的方法移动到结果集中的下一行，前一行，第一行和最后一行记录。另外，Recordset 对象连接到 Connection 对象之后，还可以通过 Open()方法执行查询或存储过程。对象使用结束后通过 Close()方法关闭 Recordset 对象以释放关联的系统资源，或者将其设置为 None 完全删除对象。

**【例 9.3】** 编写程序创建 Recordset 对象访问已打开数据库中的 Students 表并遍历显示其所有记录的学号、姓名和出生年月字段。

Students 表包括以下字段：学号（短文本型）、姓名（短文本型）、性别（短文本型）、专业（短文本型）、出生年月（日期/时间型）、党员（是/否型）、助学金（货币型）。

```
rs = win32com.client.Dispatch(r'ADODB.Recordset')      # 创建 Recordset 对象
rs.Open('Students',conn)                                # 打开 Students 表进行内容读取
while not rs.EOF:                                         # 遍历所有记录
```

```
print(rs.Fields("学号"), rs.Fields("姓名"), rs.Fields("出生年月")) # 读取字段的值并输出
rs.MoveNext() # 移动到下一条记录
```

200101	李小明	建筑学
200102	陈晓华	建筑学
200103	丁保华	建筑学
200201	姜沛棋	交通工程
190202	朱克良	交通工程

图 1.9.5 查询结果

部分运行结果如图 1.9.5 所示。首先使用 win32com.client 模块的 Dispatch 方法来创建 RecordSet 对象，再使用 Open()方法打开 Students 数据表，其形式为：  
Open(Source,ActiveConnection,CursorType,LockType,Options)，各参数的含义见表 1.9.3。

表 1.9.3 Recordset 对象 Open()方法参数说明

参 数	说 明
Source	可选参数。表名、SQL 语句、Command 对象变量名、Recordset 文件名等。
ActiveConnection	可选参数。有效的 Connection 对象变量名；或包含ConnectionString 参数的字符串。
CursorType	打开 Recordset 时应使用的游标类型。
LockType	打开 Recordset 时应使用的数据锁定类型。
Options	指示如何使用 Source 参数。

其中，游标类型(CursorType)和数据锁定类型(LockType)是常用的两个参数，表 1.9.4 和表 1.9.5 说明了它们的不同值/常量代表的不同含义。当仅读取 RecordSet 对象时，这两个参数可以取默认值，需要对记录集进行维护时，需要采用适当的参数值。

表 1.9.4 CursorType 参数值的说明

CursorType 值/ CursorTypeEnum 常量	说 明
0 (默认) / AdOpenForwardOnly	仅向前游标。只读，只能向前浏览记录。当只需要在记录集中单向移动时经常使用。
1/ AdOpenKeyset	键集游标。只读，用户对记录做的更改和删除通过游标可体现，但用户对记录做的添加通过游标不体现。
2/ AdOpenDynamic	动态游标。可读写，支持所有移动。用户对记录做的添加、更改和删除通过游标均可体现。
3/ AdOpenStatic	静态游标。可读写，可用来查找数据的静态副本。用户对记录做的修改通过游标均不体现。

表 1.9.5 LockType 参数值的说明

LockType 值/ LockTypeEnum 常量	说 明
1 (默认) / AdLockReadOnly	只读，不能修改数据。
2/ AdLockPessimistic	以保守方式逐个锁定记录，在编辑之后立即锁定记录。
3/ AdLockOptimistic	以开放方式逐个锁定记录，只在调用 Update 方法提交更新后才锁定记录。
4/ AdLockBatchOptimistic	开放式批量锁定，允许修改多个记录，只有调用 UpdateBatch 方法后才锁定记录。

在打开 RecordSet 对象时，对象 rs 其实只是一个游标，游标指向对应数据表中的一条记

录，称为当前记录。接下来，可以按字段名称或者字段位置访问当前记录各字段的值。由于“学号”是表中索引为 0 的字段，以下几种方法中，如果将“学号”与索引 0 替换，作用都等价。

```
print(rs.Fields.Item("学号"))      # 以下几个方法等价
print(rs.Fields("学号"))
print(rs.Fields(0).value)
print(rs.Fields.Item(0).value)
```

也可以通过一些函数来移动游标以改变当前记录：使用 `rs.MoveFirst()`、`rs.MoveLast()`、`rs.MovePrevious()`、`rs.MoveNext()` 分别表示移动到首记录、末记录、前一个记录、后一个记录。另外还可以通过属性来判断记录的位置。属性 `rs.BOF` 用于判断当前记录是否位于第一个记录的前面；属性 `rs.EOF` 判断当前记录是否位于最后一个记录之后，常用于判断记录是否遍历完。如果 `rs.BOF` 和 `rs.EOF` 同时为 `True` 时说明当前 `rs` 中没有记录，那么也不能进行上面几个 `Move` 操作。例 9.1 代码中，最后关闭 `Connection` 对象并删除 `Recordset` 对象，后面的例子中将省略这些代码。

**【例 9.4】** 编写程序创建 `Recordset` 对象查询和维护已打开数据库中的 `Scores` 表。

```
rs = win32com.client.Dispatch(r'ADODB.Recordset')
rs.Open('Select 成绩 from Scores where 学号="200102" and 课程="大学计算机"',conn)
print(rs.Fields.Item(0).value)      # 查询当前结果记录的第 1 列值
rs.Close()
rs.Open('Scores', conn, 1, 3)       # 以开放锁定方式打开 RecordSet 对象，使得表可以编辑更新
rs.AddNew()                         # 添加一条新记录
rs.Fields.Item(0).Value = "200808"  # 设置新记录的各个字段值
rs.Fields.Item(1).Value = "Python"
rs.Fields.Item(2).Value = "98"
rs.Update()                         # 更新
for i in range(0,3):
    print(rs.Fields.Item(i).value,end = " ")
```

程序执行结果

如图 1.9.6 所示。

90.0
200808 Python 98.0

图 1.9.6 使用 `Recordset` 查询结果

从以上代码可以看到，使用 `Recordset` 对象对表进行查询时，只需要把例 9.3 代码中的 `Open` 语句中的数据表名称改成 SQL 语句，执行后即可直接查看查询结果。如果为表增加记录，先以开放锁定的方式打开 `Recordset` 对象，使得表可以编辑更新，然后调用 `Recordset` 对象的 `AddNew()` 方法，并设置新记录的各个字段值，最后通过 `Update()` 方法将内容更新到表中。而且，在调用 `AddNew()` 方法后，新记录将成为当前记录。所以可通过输出 `rs` 的当前项观察新记录。

### 三、基本查询和维护

直接使用 `Connection` 对象的 `Execute()` 方法，可通过 SQL 语句进行数据库的查询和维护。



执行查询时，Connection 对象的 Execute()方法将返回 Recordset 对象，注意此时无需创建新的 Recordset 对象。

**【例 9.5】** 编写程序使用 Connection 对象查询和维护数据库，并可以根据用户输入的信息创建条件变化的 SQL 语句进行查询。

```
conn.Execute("Update Scores set 成绩 = 90 where 学号 = '200102' and 课程 = '高等数学'") #语句①
(rs1,Result1)= conn.Execute("Select 学号,课程,成绩 from Scores Order by 学号") #语句②
while not rs1.EOF:
    print(rs1.Fields.Item(0).value,rs1.Fields(1),rs1.Fields('成绩'))
    rs1.MoveNext()
while 1:
    stuNo=input("Input 学号:")
    if stuNo=="":
        break
    sql="Select 课程,成绩 from Scores where 学号 = " + stuNo + " " #语句③
    (rs,Result)= conn.Execute(sql)
    while not rs.EOF:
        print(rs.Fields.Item(0).value,rs.Fields.Item(1).value)
        rs.MoveNext()
```

程序执行时，不断输入不同的学号，将显示其每门课程成绩，直到输入为空为止。结果如图 1.9.7 所示。

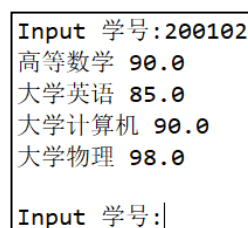


图 1.9.7 例 9.5 使用 Connection 执行查询结果

上面的程序中，反复要求用户输入，然后生成适当的 SQL 语句并执行。但是不足之处在于每次输入不同值之后需要重建 SQL 语句，虽然仅仅改变了某个字段的值。而这一点可以通过 Command 对象配合 Parameter 对象来解决。

## 四、参数化查询和维护

Command 对象定义对数据源执行的命令。通过设置 ActiveConnection 属性与已打开的 Connection 对象进行关联；并将要执行的 SQL 语句设置为 CommandText 属性，使用 Execute()方法执行查询过程。使用 Command 对象执行查询时，返回结果传给 Recordset 对象，然后使用它进行结果的遍历。Command 对象执行查询的一个主要特性在于它能够为查询过程使用参数，即 Parameter 对象。不需要使用参数时，也可以将 SQL 语句传送给 Connection 对象的 Execute()方法或 Recordset 对象的 Open()方法。

**【例 9.6】** 编写程序使用 Command 对象进行数据库查询。

```
cmd = win32com.client.Dispatch('ADODB.Command')
cmd.ActiveConnection = conn # 将 ActiveConnection 设置为 Connection 对象
```

```
cmd.CommandText="Select 课程,成绩 from Scores where 学号 = '201101'"
(rs,Result) = cmd.Execute()          # 执行查询, 并在 RecordSet 对象 rs 收集结果
while not rs.EOF:
    print(rs.Fields.Item(0).value,rs.Fields.Item(1).value)
    rs.MoveNext()
cmd = None
```

程序运行结果如图 1.9.8 所示。

大学物理	48.0
高等数学	82.0
大学英语	81.0
大学计算机	82.0

图 1.9.8 例 9.6 使用 Command 执行查询结果

如何使得查询中的参数反复发生变化时无需每次重新构建 SQL 语句? 可以通过 Command 对象结合 Parameter 对象。如例 9.7 所示, 当创建好 Command 对象后, 创建一个接受若干参数的查询语句, 并将通过 Command 对象的 CreateParameter()方法创建 Parameter 对象, 然后通过 Append()方法附加到命令对象。当参数反复发生变化时, 程序要做的就是更改 Parameter 对象的值并再次调用 Execute()方法。

**【例 9.7】** 编写程序使用 Command 对象和 Parameter 对象进行数据库查询。

```
# 此处省略创建 Command 对象并设置 ActiveConnection 属性的代码, 参见例 9.6
cmd.CommandText ="SELECT 学号,课程,成绩 FROM Scores WHERE 成绩 >= ? AND 成绩 <=?"
param1=cmd.CreateParameter('minS',3)      # 创建整型输入参数
param2=cmd.CreateParameter('maxS',3)
cmd.Parameters.Append(param1)              # 将参数附加到 Command 对象
cmd.Parameters.Append(param2)
while 1:
    param1.Value = eval(input("Enter min score:"))
    if param1.Value<0:
        break
    param2.Value = eval(input("Enter max score:"))
    (rs, result) = cmd.Execute()            # 执行 CommandText, 结果返回到 RecordSet 对象 rs
    # 此处省略遍历查看数据库的代码, 参见例 9.3
cmd = None
```

Command 对象的 CreateParameter 方法形式为:

CreateParameter (Name,Type,Direction,Size,Value)

参数的描述如表 1.9.6 所示。

表 1.9.6 Create Parameter()方法参数描述

参 数	说 明
Name	可选, Parameter 对象名称。
Type	可选, Parameter 对象数据类型。
Direction	可选, Parameter 对象类型。默认为 1, 表示输入参数。
Size	可选, 参数值最大长度 (以字符或字节数为单位)。
Value	可选, 变体型, Parameter 对象的值。

其中 Type 属性常用值如表 1.9.7 所示。

表 1.9.7 属性常用值

属性值	说 明	属性值	说 明
3	4 字节带符号整型	7	日期值
5	双精度浮点数	200	字符串值
6	货币值	11	布尔型值

需要注意，例 9.7 代码中创建了两个整型参数，但不同的参数类型在创建时需要采用不同的参数，或者可能需要设置其他等属性，如 **Size**。例如，创建字符串值类型的参数要求使用四个参数，见例 9.8。如果没有正确设置参数所需属性，当执行 **Append()** 方法时将出现错误。

在例 9.7 的代码中，创建了一个 **Command** 对象并为其分配一个包含两个参数的 SQL 语句。之后创建的两个 **Parameter** 对象对应了 SQL 语句中的两个变量。当反复请求用户输入时，将其设置为参数值并执行该语句。此时当用户使用不同的值时，程序重复地读取用户的输入值作为参数执行查询。所以，利用 **Command** 结合 **Parameter** 对象，无需为不同的输入值重新生成 SQL 语句，可提供显著的性能优势。

接下来，使用 **Command** 对象结合参数来执行数据库维护。由于使用的是 **Parameter** 对象，因此代码会自动转义任何引号字符和其他特殊字符本身。

**【例 9.8】**编写程序使用 **Command** 对象和 **Parameter** 对象进行数据库维护。

```
# 创建一个接受 3 个参数的 SQL 语句
cmd.CommandText = "INSERT INTO Scores(学号,课程,成绩) VALUES(?,?,?)"
# 创建 3 个 Parameter 对象并附加到 Command 对象 cmd
param1 = cmd.CreateParameter('学号',200,1,50) # 创建字符串值类型参数，设置最长为 50 个字符
param2 = cmd.CreateParameter('课程',200,1,50)
param3 = cmd.CreateParameter('成绩',3)
cmd.Parameters.Append(param1)
cmd.Parameters.Append(param2)
cmd.Parameters.Append(param3)
while 1:
    param1.Value = input("Enter 学号:")
    if param1.Value=="":
        break
    param2.Value = input("Enter 课程:")
    param3.Value = eval(input("Enter 成绩:"))
    (rs, result) = cmd.Execute()
cmd = None
```

同样，还可以用相同的方法来使用参数处理 **UPDATE** 和 **DELETE** 语句。例如，可以使用下列 **UPDATE** 语句来更新数据：

```
cmd.CommandText = "UPDATE Scores SET 成绩 = 成绩+? WHERE 课程=?"
```

或者使用 **DELETE** 语句删除数据：

```
cmd.CommandText = "DELETE FROM Scores WHERE 成绩<?"
```

创建各种类型的参数，利用 **Command** 对象执行数据库查询更加方便和灵活，也更加高效，体现了明显的性能优势。

## 9.1.3 综合应用

**【例 9.9】**设计 Access 数据库应用程序，显示、添加、删除、修改、查询和统计数据。

为了方便经常查看记录集中的所有记录的字段内容，首先自定义函数 show(r)，遍历显示 Recordset 对象 r 中所有记录，并显示所包含的总记录数。

```
def show(r):
    r.MoveFirst() #光标移到首条记录
    count = 0
    while not r.EOF:
        for i in range(r.Fields.Count):
            print(r.Fields[i].Name, ": ", r.Fields[i].Value)          #字段名：字段内容
        count += 1
        r.MoveNext()
    print("Records Count:%d" % (count,))
```

1. 显示 Students 表所有内容，部分结果如图 1.9.9 所示。

```
#导入模块
import win32com.client
#①使用 win32com.client 模块的 Dispatch()方法来创建 Connection 对象
conn = win32com.client.Dispatch(r'ADODB.Connection')
#②设置其 ConnectionString 属性，并通过 Open 方法打开数据库连接
DSN= 'Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Class.accdb;'
conn.Open(DSN)
#③使用 Dispatch()方法来创建 Recordset 对象 rs，并使用 Open()方法打开数据表 rs_name
rs = win32com.client.Dispatch(r'ADODB.Recordset')
rs_name = 'Students'
rs.Open(rs_name,conn,1,3)          # 打开数据表进行内容读取
show(rs)                          # 自定义函数，显示 rs 中记录
```

首先导入 win32com.client 模块，使用模块的 Dispatch()方法创建 Connection 对象 conn 并设置其属性，然后通过 Open()方法打开与 Class.accdb 的数据库连接。接下来，创建 RecordSet 对象 rs，通过 Open()方法打开数据表 rs\_name，再调用自定义函数 show(rs)来显示记录集中的内容。注意此处以开放锁定的方式打开 Recordset 对象，使其在随后的代码中可进行编辑更新。

2. 使用 Recordset 对象添加一条记录，学号为 201201，姓名为李建国。

```
#④调用 RecordSet 对象的 AddNew()方法，设置新记录的各个字段值，最后通过 Update()方法将内容更新到表中
rs.AddNew()          # 添加一条新记录
rs.Fields.Item(0).Value = "201201"          # 为新记录的字段设置值
rs.Fields.Item(1).Value = "李建国"
rs.Update()          # 更新
show(rs)
```

为表添加记录时，可以直接调用 Recordset 对象的 AddNew()方法，此时新记录将成为当前记录，设置新记录的各字段值，最后通过 Update()方法将内容更新到表中。

3. 使用 Connection 对象增加、删除和更改记录，部分运行结果如图 1.9.10 所示。

```
#⑤调用 Connection 对象的 Execute()方法执行记录的增删改
sql = "INSERT INTO "+ rs_name + "(学号,姓名) VALUES ('201205','陈为民)"          # 创建 SQL 语句
(rs,result1) = conn.Execute(sql)          # 执行 SQL 语句
```

```

sql = "Delete * FROM " + rs_name + " WHERE 学号 = '201201'"           # 创建 SQL 语句
conn.Execute(sql)                                                     # 执行 SQL 语句
sql = "UPDATE " + rs_name + " Set 助学金=200 WHERE 学号 = '201205'"   # 创建 SQL 语句
conn.Execute(sql)                                                     # 执行 SQL 语句
rs = win32com.client.Dispatch(r'ADODB.Recordset')                    # 创建 Recordset 对象
rs.Open(rs_name,conn)                                                 # 打开表进行内容读取
show(rs)

```

数据库的查询或维护也可以通过调用 Connection 对象的 Execute()方法, 此处设置增、删、改记录的 SQL 语句作为 Execute()方法的参数。执行维护数据库的 SQL 语句命令时, 将返回关闭的 Recordset 对象 rs。此时在调用自定义函数 show(rs)显示记录集中内容之前, 应该创建 Recordset 对象, 并在执行 Open()方法时传送相应的参数, 打开数据表进行内容读取与显示。

#### 4. 使用 Command 对象结合 Parameter 对象查询统计以下结果。

```

学号 : 200101
姓名 : 李小明
性别 : 男
专业 : 建筑学
出生年月 : 2002-01-21 00:00:00+00:00
党员 : True
助学金 : 220
学号 : 200102
姓名 : 陈晓华
性别 : 女
专业 : 建筑学
出生年月 : 2001-09-21 00:00:00+00:00
党员 : False
助学金 : 210

```

图 1.9.9 数据显示部分结果

```

学号 : 201205
姓名 : 陈为民
性别 : None
专业 : None
出生年月 : None
党员 : False
助学金 : 200

```

图 1.9.10 数据维护后部分结果

首先根据输入的助学金上限和入党情况查找符合条件的学生学号和姓名, 直到输入的助学金值小于 0 时结束查询。运行结果如图 1.9.11 (1) 所示。

```

#⑥创建 Command 对象, 并利用其进行参数化查询
cmd = win32com.client.Dispatch('ADODB.Command')
cmd.ActiveConnection = conn           # 将 ActiveConnection 设置为 Connection 对象
#利用 cmd 进行参数化查询
cmd.CommandText="SELECT 学号,姓名 FROM " + rs_name + " WHERE 助学金<= ? AND 党员=?"
param1=cmd.CreateParameter('grants',3)           # 创建整型输入参数
param2=cmd.CreateParameter('member',11)          # 创建布尔型输入参数
cmd.Parameters.Append(param1)                   # 将参数附加到 Command 对象
cmd.Parameters.Append(param2)
while 1:
    param1.Value = eval(input("Enter student grants:")) # 输入助学金参数
    if param1.Value<0: break
    param2.Value = eval(input("Enter party member:"))   # 输入党员参数
    (rs, result) = cmd.Execute()                       # 执行 CommandText, 结果返回到 Recordset 对象 rs
    show(rs)

```

```

Enter student grants:190

Enter party member:True
学号 : 200103
姓名 : 丁保华
Records Count:1

Enter student grants:200

Enter party member:False
学号 : 190202
姓名 : 朱克良
学号 : 201101
姓名 : 黎敏艳
学号 : 201205
姓名 : 陈为民
学号 : 201201
姓名 : 李建国
Records Count:4

Enter student grants:-5

```

(1)

```

Enter ID:200102
姓名 : 陈晓华
平均成绩 : 90.75
最高成绩 : 98.0
Records Count:1

Enter ID:190202
姓名 : 朱克良
平均成绩 : 80.5
最高成绩 : 90.0
Records Count:1

Enter ID:|

```

(2)

图 1.9.11 使用 Command 对象进行查询统计结果

Command 对象可以执行增删改数据维护操作，也可以执行数据的查询。Command 对象执行命令的主要优点之一是可以使用参数。将 SQL 语句中的参数用问号代替后，使用 CreateParameter()方法指定参数的名称、类型、长度、值等创建新的 Parameter 对象。随后利用 Append()方法将它们添加到 Parameters 集合，并当读入用户输入的参数值后，赋给参数的 Value 属性。执行 Execute()方法时，将使用 Command 对象中参数的当前值，或者可通过 Execute()调用传入的参数覆盖它们。

接下来再根据学号查询学生的姓名、平均成绩和最高成绩，查询结果如图 1.9.11 (2) 所示。

```

#⑦利用 Command 对象进行参数化查询
cmd = win32com.client.Dispatch('ADODB.Command')
cmd.ActiveConnection = conn
cmd.CommandText = "SELECT 姓名,Avg(成绩) AS 平均成绩,Max(成绩) as 最高成绩 from
Students,Scores where Scores.学号= ? and Students.学号=Scores.学号 group by 姓名";
param1=cmd.CreateParameter('ID',200,1,8)          # 创建字符串型输入参数
cmd.Parameters.Append(param1)                      # 将参数附加到 Command 对象
while 1:
    param1.Value = input("Enter ID:")              # 输入学号参数
    if len(param1.Value)==0: break
    (rs, result) = cmd.Execute()                    # 执行 CommandText, 结果返回到 Recordset 对象 rs
    show2(rs)
conn.Close()
rs = None
cmd = None

```

## 9.2 访问其他数据库

Python 还可以支持访问很多不同的数据库。Python 的标准数据库接口 DB-API 为很多数据库操作提供了标准的编程接口，并在连接数据库后，使用基本一致的流程访问各种数据

库。不过，对于不同数据库，由于导入不同的标准库，创建 Connection 对象的语句也略有不同。

### 9.2.1 访问 SQLite 数据库

SQLite 是一款轻型、开源的关系型数据库，整个数据库存储在一个文件中，在很多设备中 SQLite 数据库已成为本地存储的最佳方案。Python 通过内置的 sqlite3 模块访问 SQLite 数据库，无需安装直接导入。创建 SQLite 数据库时，可在 Python 程序中通过执行 SQL 语句直接创建，或使用 Navicat for SQLite、SQLite Studio 等工具进行可视化创建与管理。

Python 操作 SQLite 数据库的基本流程如图 1.9.12 所示。

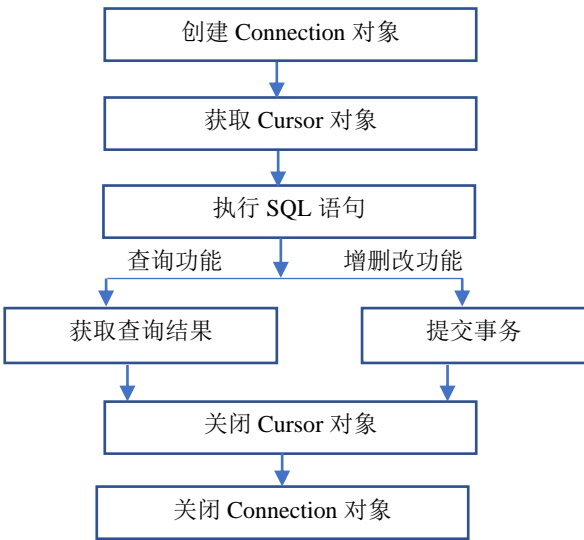


图 1.9.12 访问数据库的基本流程

访问 SQLite 数据时，首先导入 sqlite3 模块；然后执行以下基本步骤：

- ①通过 sqlite3 模块的 connect()方法，创建一个与数据库文件相关联的 Connection 对象。
- ②直接调用 Connection 对象的 execute()或 executemany()方法，执行指定的 SQL 语句；或者先用 Connection 对象的 cursor()方法创建 Cursor 对象，然后调用 Cursor 对象的 execute()或 executemany()方法，执行 SQL 语句。
- ③如果 SQL 语句仅执行查询，可以通过 Cursor 对象的多种方法获取查询结果，也可以直接遍历 execute()方法的返回结果；如果 SQL 语句执行维护（增、删、改）操作，则要通过 Connection 对象的 Commit()方法，提交事务以保存对数据库的更新。
- ④关闭 Cursor 对象和 Connection 对象，释放资源。

### 一、数据库创建

**【例 9.10】**创建 SQLite 数据库 Class.db，并在其中创建 Scores 表，包括以下字段：学号（短文本型）、课程（短文本型）、成绩（数字型）。通过以下步骤实现数据库和表的创建，具体代码如下：

```

#①导入模块
import sqlite3
#②创建 Connection 对象
conn=sqlite3.connect("Class.db")
#③获取 Cursor 对象
cursor=conn.cursor()
#④执行 SQL 语句，创建 Scores 表并添加一条记录
cursor.execute('CREATE TABLE Scores(学号 text,课程 text,成绩 real)')
cursor.execute("INSERT INTO Scores VALUES ('200101','大学计算机',82)")
#⑤关闭 Cursor 对象
cursor.close()
#⑥提交事务
conn.commit()
#⑦关闭 Connection 对象
conn.close()

```

由于 SQLite 的驱动内置在 Python 标准库中，首先直接导入 sqlite3 模块，随后使用它操作 SQLite 数据库。使用该模块之前，通过 sqlite3.connect()方法连接数据库"Class.db"，创建与数据库相关联的 Connection 对象 conn。若该数据库文件不存在，则将在当前目录新建一个数据库；也可以使用带路径的文件名，则会在任意位置创建数据库；还可以使用特定的名称":memory:"，则会在 RAM 中创建一个数据库。

创建了 Connection 对象之后，在第③句中使用其 cursor()方法创建一个 Cursor 对象；然后，在第④句中分两次调用其 execute()方法执行 SQL 语句，分别实现创建数据表 Scores、在表 Scores 中添加数据记录的操作；随后通过第⑤句关闭 Cursor 对象以释放资源。

第⑥句中，由于前面通过 insert 语句对数据库新增数据，此时需要使用 Connection 对象的 commit()方法提交事务。如果不调用 commit()方法，那么自上一次调用 commit() 以来对该数据库做的修改将不被保存到数据库。

要注意的是，再次运行该实例时，由于 Scores 表已经存在，会提示错误信息：OperationalError: table Scores already exists。

将语句④修改为使用 Connection 对象的 execute()方法，也可以执行同样的操作，例如：

```

conn.execute("CREATE TABLE S2(学号 text,课程 text,成绩 real)")
conn.execute("INSERT INTO S2 VALUES ('300101','大学计算机',82)")

```

此处使用 Connection 对象的 execute()方法，实际上是通过先调用 cursor()方法创建了一个 Cursor 对象，然后根据给定的参数再调用 Cursor 对象的 execute()方法。所以，它其实是由 Cursor 对象提供的 execute()方法的快捷方式。

第⑦句代码关闭数据库连接。要注意，如果前面对数据库做了更改，要在关闭数据库连接前调用 commit()方法提交事务，否则所做的更改将全部丢失。

Connection 和 Cursor 都是 sqlite3 模块中重要的类，表 1.9.8 中列出了一些常用的方法。

表 1.9.8 sqlite3 模块中类的常用方法

方法	说明
sqlite3.connect(database)	打开一个到 SQLite 数据库文件 database 的连接。
connection.cursor()	创建一个 Cursor 对象



connection.execute(sql [, optional parameters])	自动创建 Cursor 对象并调用其 execute () 方法
connection.executemany(sql[, parameters])	自动创建 Cursor 对象并调用其 executemany() 方法
connection.commit()	提交当前的事务
connection.rollback()	回滚到当前事务开始前的状态,即恢复到上次调用 commit() 后数据库的状态
connection.close()	关闭数据库连接
cursor.execute(sql [,optional parameters])	执行一个 SQL 语句
cursor.executemany(sql, seq_of_parameters)	对 seq_of_parameters 中的所有参数或映射执行一个 SQL 语句。

**【例 9.11】**通过用户输入方式，在 Class.db 数据库的 Scores 表中添加多条记录。

```
import sqlite3
conn=sqlite3.connect("Class.db")
cursor=conn.cursor()
#执行 SQL 语句，通过输入添加多条记录
while True:
    no=input("请输入学号(输 0 退出):\n")
    if no=='0':
        break
    course=input("请输入课程:\n")
    mark=int(input("请输入成绩:\n"))
    cursor.execute("INSERT INTO Scores VALUES (?,?,"),(no,course,mark))
cursor.close()
conn.commit()
conn.close()
```

通过用户输入创建 SQL 语句，execute()方法中调用的 SQL 语句已被参数化，使用了占位符"?"代替 SQL 文本。

**【例 9.12】**通过 executemany()方法，在 Class.db 数据库的 Scores 表中添加多条记录。

方法一：使用直接创建的列表作为 executemany()方法中 SQL 语句的参数。

```
import sqlite3
conn=sqlite3.connect("Class.db")
cursor=conn.cursor()
#①创建列表
persons=[ ("200103","大学英语",88),
          ("200103","高等数学",89),
          ("200103","大学计算机",72),
          ("200103","大学物理",90) ]
#②调用 executemany()方法
cursor.executemany("INSERT INTO Scores VALUES (?,?,"persons)
cursor.close()
conn.commit()
conn.close()
```

语句①中列表中包含 4 个元组元素，每个元素包含了一条记录的若干字段值。语句②调

用 `executemany()` 方法，使用给定的列表参数，执行同一个 SQL 语句，把列表中 4 个元素所代表的记录逐一添加到 `Scores` 表中。

方法二：用生成器产生 SQL 语句的参数。将上方的语句①改成一个生成器。

```
def data_gen():
    for no in range(2000112,2000120):
        for course in {"大学计算机","高等数学","大学物理","大学英语"}:
            yield str(no),course,random.randint(80,100)
```

语句②依然调用 `executemany()` 方法，但参数改为用生成器产生：

```
cursor.executemany("INSERT INTO Scores VALUES(?,?,?)",data_gen())
```

当然，此处也可以调用 `Connection` 对象的 `executemany()` 方法，它实际上是 `Cursor` 对象的 `executemany()` 方法的快捷方式。

## 二、数据库查询

通过实例实现对数据库的查询操作，并显示查询结果。

【例 9.13】查询 `Class.db` 中 `Scores` 表的所有记录，然后分别使用不同的方法读取查询结果。具体代码如下：

```
import sqlite3
conn=sqlite3.connect('Class.db')
cursor=conn.cursor()
#执行 SQL 语句，查询 Scores 表的所有记录
cursor.execute("SELECT * FROM Scores")          #语句①
#获取查询结果
result=cursor.fetchone()                        #语句②
#显示结果
print(result)
cursor.close()
conn.close()
```

对于查询得到的结果，通常可以使用表 1.9.9 中列出的 3 种方法来读取。

表 1.9.9 `Cursor` 对象获取查询结果集的常用方法

方法	说明
<code>cursor.fetchone()</code>	获取查询结果集中的下一条记录，返回一个单一的序列。当没有更多可用的数据时，则返回 <code>None</code> 。
<code>cursor.fetchmany([size])</code>	获取查询结果集中由 <code>size</code> 参数（默认为 1）指定数量的记录，返回一个列表。
<code>cursor.fetchall()</code>	获取查询结果集中所有（剩余的）记录，返回一个列表。当没有可用的记录时，则返回一个空的列表。

1. 从例 9.12 代码可以看到，使用 `fetchone()` 方法返回查询结果集中的一条记录。结果 `result` 为一个元组，执行结果如下：

`('200101', '大学计算机', 82.0)`

如果继续执行 `fetchone()` 方法，将返回结果集中的下一条记录。

2. 将语句①修改为 `result=cursor.fetchmany(2)`，获取结果集中的 2 条记录。返回的 `result`

为一个列表，包括 2 个元组，执行结果如下：

```
[('200101', '大学计算机', 82.0), ('200101', '大学物理', 81.0)]
```

若 size 缺省，则默认获取 1 条记录；若 size 大于实际可用记录，则仅返回由可用记录组成的列表。

3. 将语句②修改为 `result=cursor.fetchall()`，返回的 `result` 为一个列表，包含了查询结果集中所有记录。

4. 将语句①修改为 `cursor.execute("SELECT * FROM Scores WHERE 学号=?",('200102',))`，将查询到学号 200102 的所有记录。在 `execute()` 方法的第一个参数 `SELECT` 语句中，使用了问号？作为占位符，代替了具体的数值，参数为元组形式，包含了将要替换问号的元素（注意，元组中只有一个元素时，不要忽略最后的逗号）。这个语句的作用等价于 `cursor.execute("SELECT * FROM Scores WHERE 学号='200102'")`。但是，使用占位符可以避免 SQL 注入的风险，所以建议使用占位符。

5. 将语句①和②一起替换成以下语句，也可以直接遍历显示查询结果。

```
for row in cursor.execute("SELECT * FROM Scores"):
    print(row)
```

### 三、数据库维护

通过实例来实现对数据库的修改、删除操作，并浏览显示维护后的查询结果。

【例 9.14】对 Scores 表中所有的“大学计算机”课程成绩增加 5 分；删除“大学物理”课程的所有记录；查询学号 200101 的大学计算机课程成绩；浏览维护后的数据库所有记录。具体代码如下：

```
import sqlite3
conn=sqlite3.connect('Class.db')
cursor=conn.cursor()
#执行修改和查询语句
cursor.execute("UPDATE Scores SET 成绩 = 成绩+? WHERE 课程=?", (5, '大学计算机'))
cursor.execute("DELETE FROM Scores WHERE 课程=?", ('大学物理',))
cursor.execute("SELECT * FROM Scores WHERE 学号=:no and 课程=:course", {"no": '200101', "course": '大学计算机'})
print(cursor.fetchone())
cursor.execute("SELECT * FROM Scores")
result=cursor.fetchall()
print(result)
cursor.close()
conn.commit()
conn.close()
```

代码中，对数据库进行了删除和修改等维护操作，采用了 `Cursor` 对象的 `execute()` 方法执行 SQL 语句。在 `UPDATE` 和 `DELETE` 语句中使用了常见的问号占位符；而在第一个 `SELECT` 查询语句中使用了命名占位符：采用冒号加 key 的形式作为占位符，参数采用字典形式，每个 value 将替换 key 在语句中所占位置。

## 9.2.2 访问 MySQL 数据库

MySQL 是一种功能强大的网络关系型数据库，开源且支持大型数据库。pymysql 是从 Python 连接到 MySQL 数据库服务器的接口。在 Python 中可通过 `pip install pymysql` 安装模块，然后使用 pymysql 模块来访问 MySQL 数据库。

【例 9.15】连接数据库 ClassDB，创建表 Scores，字段包括学号 no、课程 course 和成绩 score；在 Scores 表添加记录，查询和显示 no 为 200101 的记录。

```
import pymysql #语句①
conn = pymysql.connect(host="localhost",user="root",passwd="u123",db="ClassDB") #语句②
cursor = conn.cursor()
cursor.execute("DROP TABLE IF EXISTS Scores") #如果表 Scores 存在则删除
cursor.execute("CREATE TABLE Scores(no varchar(20), course varchar(20), score int)")
sql="INSERT INTO Scores VALUES('200101','CS',82)"
try:
    cursor.execute(sql)
    conn.commit()
except:
    conn.rollback() #若发生错误则回滚
cursor.execute("SELECT * FROM Scores WHERE no=%s"%(200101))
result = cursor.fetchall()
print(result)
cursor.close()
conn.close()
```

利用 pymysql 库访问 MySQL 数据库时，操作过程和 9.2.1 节访问 SQLite 数据库基本一致。本例中，同样先采用 pymysql 的 connect() 方法连接到指定的数据库，再通过创建的 Cursor 对象通过 execute() 方法执行 SQL 语句，并在修改数据库后通过 commit() 方法提交事务。当发生错误时，可以通过 rollback() 方法回滚到当前事务开始前的状态，也就是上次调用 commit() 之后数据库当时的状态。

### 9.2.3 访问 SQL Server 数据库

SQL Server 是 Microsoft 公司开发的关系型数据库管理系统，pymssql 是从 Python 连接到 SQL Server 数据库的接口，可通过 `pip install pymssql` 安装，或者下载与本身 python 版本所对应的 pymssql 安装文件(.whl)，再通过 `pip install *.whl` 进行安装。

将例 9.15 中语句①和②修改为下列语句，导入 pymssql 模块并连接到 ClassDB 数据库。

```
import pymssql
conn = pymssql.connect(host='localhost',user='root',password='u123',database='ClassDB')
```

数据 Connection 对象建立后，之后对数据库的访问操作类似于 9.2.1 节对 SQLite 数据库的操作，不再赘述。

## 习题

### 一、选择题

1. 下列关于 Connection 对象说法错误的是：

A. 使用 Open 方法可建立到数据源的连接，在其成功完成后，连接是活跃的，可以对

它发出命令并且处理结果。

- B. 使用 Close 方法关闭对象并非将它从内存中删除，但也无法再次打开。
  - C. 要将对象从内存中完全删除，可将对象变量设置为 None。
  - D. 可以直接使用其 Execute 方法，通过 SQL 语句进行数据库查询。
2. 下列关于 Recordset 对象，说法错误的是：
- A. 任何时候，Recordset 对象所指的当前记录均为集合内的第一条记录。
  - B. 所有 Recordset 对象均使用记录（行）和字段（列）进行构造。
  - C. 可以将查询 SQL 语句传递给 Recordset 对象的 Open 方法执行数据库查询。
  - D. 打开至少包含一条记录的 Recordset 对象时，第一条记录为当前记录，而 BOF 和 EOF 属性为 False。
3. 下列关于 Command 对象，说法正确的是：
- A. 可使用其 CommandType 属性指定要执行的 SQL 语句。
  - B. 可使用其 Execute 方法执行 SQL 命令并在适当的时候返回 Recordset 对象。
  - C. Command 对象的一个优点是能够使用 Parameter 参数，当参数反复发生变化时，程序会为不同的输入值重新生成不同的 SQL 语句，并再次调用 Execute 方法。
  - D. 使用 CreateParameter 方法可用指定的属性创建新的 Parameter 对象，并会自动地追加到 Command 对象的 Parameter 集合。
4. 下列关于 Field 对象，说法错误的是：
- A. 每个 Field 对象对应于 Recordset 对象中的一条记录。
  - B. Field 对象包含列的值、类型、字段最大长度信息等。
  - C. 使用其 Name 属性可返回字段名，使用其 Value 属性可设置或返回当前记录的数据。
  - D. Recordset 对象含有由 Field 对象组成的 Fields 集合。
5. 下列关于 sqlite3 模块中方法的描述，错误的是：
- A. sqlite3.connect()方法用于打开一个到 SQLite 数据库的连接
  - B. sqlite3.cursor()方法用于创建一个 cursor 对象
  - C. 由于 SQLite 的驱动内置在 Python 标准库，编程时无需安装可直接导入 sqlite3 模块
  - D. connect()方法的参数中可以设置为指定路径下的数据库文件，也可以使用特定名称 ":memory:"在 RAM 中创建一个数据库。
6. 下列关于维护 SQLite 数据库的说法中，错误的是：
- A. 进行增删改时，不一定要使用 commit()方法提交事务也可以保存对数据库的修改
  - B. 可以通过调用 Connection 对象 execute()方法，执行 SQL 语句来维护数据库
  - C. 可以调用 Connection 或 Cursor 对象的 executemany()方法，对指定的所有参数执行 SQL 语句
  - D. rollback()方法可以撤销当前事务，恢复至上一次调用 commit()后的状态
7. 查询 SQLite 数据库时，错误的是：
- A. Connection 对象和 Cursor 对象都可以发起对数据库的查询操作
  - B. Cursor 对象的 fetchone()方法返回查询结果集中的第一条记录
  - C. Cursor 对象的 fetchmany(size)方法返回查询结果集中的多条记录，记录数可能少于 size。
  - D. fetchmany()方法和 fetchall()方法，返回的结果都是列表。
8. 关于 Python 支持的多种数据库描述中，错误的是：
- A. 当使用 Python DB-API 连接各种数据库时，可以导入不同的标准库连接不同的数据库，访问数据库的基本流程也大不相同。
  - B. 访问 MySQL 数据库时使用的 pymysql 库，和 sqlite3 库不同的是，需要先安装才可

导入。

C. pymysql 库中也可以用 Cursor 对象的 execute()方法执行 SQL 语句,并在维护的时候通过 commit()方法提交事务

D. 访问 SQL Server 数据库时,可以导入 pymssql 库,并用其 connect()方法建立对数据库的连接。

## 二、 填空题

1. 在 ADO 基本的对象中, \_\_\_\_\_对象用于管理与数据库的连接; \_\_\_\_\_对象定义了将对数据源执行的命令。
2. Connection 对象的 \_\_\_\_\_属性由 Provider 和 Data Source 两部分组成。
3. Recordset 对象使用结束后通过 \_\_\_\_\_方法关闭对象以释放关联的系统资源,或者将其设置为 \_\_\_\_\_完全删除对象。
4. Command 对象能够为查询过程使用参数,即附加多个 \_\_\_\_\_对象,并将这些参数的值传送给 Execute 方法以进行反复执行。
5. Recordset 对象的 \_\_\_\_\_属性用于判断当前记录是否位于第一个记录的前面; \_\_\_\_\_属性判断当前记录是否位于最后一个记录之后,常用于判断记录是否遍历完。

## 三、 简答题

1. 叙述使用 Python 操作 Access 数据库的基本步骤。
2. 说明如何使用 Connection 对象执行数据库查询和维护。
3. 说明如何使用 Recordset 对象执行数据库查询和维护。
4. 说明如何使用 Command 对象执行数据库查询和维护。
5. 说明当使用查询参数的时候,如何通过 Command 对象来执行查询及维护。
6. 叙述使用 Python 操作 SQLite 数据库的基本步骤。

## 四、 程序填空题

下载数据库 BookInfo.accdb, 数据库包含一些书籍的基本数据表“Book”和读者评分表“Record”。Book 表结构包括: ISBN 编号(文本型)、书名(文本型)、出版社(文本型)、售价(数字型)、作者(文本型);Record 表结构为: ISBN 编号(文本型)、读者(文本型)、评分(数字型)。填充代码,完成以下功能。

1. 用 Recordset 对象打开数据库中的 Book 表,遍历显示其所有记录;逐一设置一条新记录的各个字段值并添加更新到 Book 表中: ("9787532175468", "走出唯一真理观", "上海文艺出版社", 58, "陈嘉映")。

```
import win32com.client
conn = win32com.client.Dispatch(r"ADODB.Connection")
conn.①_____ = 'PROVIDER = Microsoft.ACE.OLEDB.12.0;DATA SOURCE = BookInfo.accdb'
conn.Open()
rs = win32com.client.Dispatch(r'ADODB. ②_____')
rs.Open('SELECT 作者,售价 FROM Book WHERE 书名="一见你就好心情",conn)
#显示查询结果当前记录的第 1 列值
print(rs.Fields.Item(0).value,rs.Fields.Item(1).value)
rs.Close()
#以开放锁定的方式打开 RecordSet 对象
rs.Open('Book', ③_____, 1, 3)
#设置新记录的各个字段值
rs.AddNew() #添加一条新记录
```

```

rs.Fields.Item(0).Value = "9787532175468"
rs.Fields.Item(1).Value = "走出唯一真理观"
rs.Fields.Item(2).Value = "上海文艺出版社"
rs.Fields.Item(3).Value = 58
rs.Fields.Item(4).Value = "陈嘉映"
#更新到表中
rs. _____④_____
rs.Close()
conn.Close()

```

2. 使用 Connection 对象的 Execute()方法, 将《一见你就好心情》的价格更新为 9 折优惠价; 根据用户输入的书名, 查询并输出该书的作者、售价, 重复此过程, 直到书名为空。

#省略连接对象 conn 的创建、打开等语句

```

conn.Execute("Update Book _____①_____ 售价 = 售价*0.9 WHERE 书名 = '一见你就好心情'")
while 1:
    stuNo=input("Input 书名:")
    if stuNo=="":
        break
    sql="Select 作者,售价 _____②_____ Book WHERE 书名='"+stuNo+"'"
    (rs,Result)= conn. _____③_____ (sql)
    print(rs.Fields.Item(0).value,rs.Fields.Item(1).value)
#省略释放对象等语句

```

3. 输入最低价和最高价, 找到该价格区间书籍的书名和多个读者评分, 并一一列出查询结果。重复此过程, 直到输入的最低价<0 时结束循环。要求使用 Command 对象和 Parameter 对象,使用参数查询语句。

#省略连接对象 conn 的创建、打开等语句

```

cmd = win32com.client.Dispatch(r'ADODB.Command')
cmd.ActiveConnection = conn
cmd. _____①_____ ="SELECT 书名,评分 FROM Book,Record WHERE 售价 >= ?
AND 售价 <=? AND Book.ISBN 编号=Record.ISBN 编号"
param1=cmd.CreateParameter('minP',3)
param2=cmd. _____②_____ ('maxP',3)
cmd.Parameters.Append(param1)
cmd.Parameters.Append(param2)
while 1:
    param1. _____③_____ = eval(input("The lowest price:"))
    if param1.Value<0:
        break
    param2.Value = eval(input("The highest price:"))
    (_____④_____, result) = cmd.Execute()
    while not rs.EOF:
        print(rs.Fields.Item(0).value,rs.Fields.Item(1).value)
        rs. _____⑤_____
#省略释放对象等语句

```

# 实验六 列表、元组、字典、集合

## 一、实验目的

- 1、掌握不同数据序列的基本概念、定义、操作及应用场合
- 2、掌握列表、元组的分片、列表对象方法的使用
- 3、掌握内置函数 `sum`、`len` 对列表、元组的操作
- 4、掌握列表、元组推导式的用法
- 5、掌握字典的创建、遍历

## 二、实验内容

- 1、用 `random` 库，生成一个数值在-1~1 之间的 10\*5 的矩阵，再用列表推导式将其转置。计算矩阵的转置与矩阵本身的乘积。
- 2、生成 10 行 5 列的随机数矩阵。每个元素的值范围为：-1~1 之间。（提示：`random` 函数产生的值在 0-1 之间），再编写代码实现矩阵转置。
- 3、输入一个英文句子，统计其中每个字母出现的次数，组合成一个字典。例如 `{"A":5,"B":0}`。要求：大小写字母相同处理。
- 4、请编写程序，从键盘上一次输入 10 个数，数间以空格间隔，用 `map` 函数将数据整理成一个数值列表输出。
- 5、建立一个学生姓名与手机号的字典，要求输入格式为：张斌 13401279012，数据自行定义。输入-1 时结束输入过程。然后再通过一个循环，实现通过输入姓名查找手机的功能，输入“xxx”则查找结束
- 6、生成一个含有"A"~"Z"26 个字母的列表，再生成一个列表：`[1,2,...,26]`，将两个列表结合生成以字母为键、数字为值的字典。
- 7、用字典完成程序,提示用户输入电话号码,并用英文单词形式显示数字。例如:输入 138 显示: one three eight



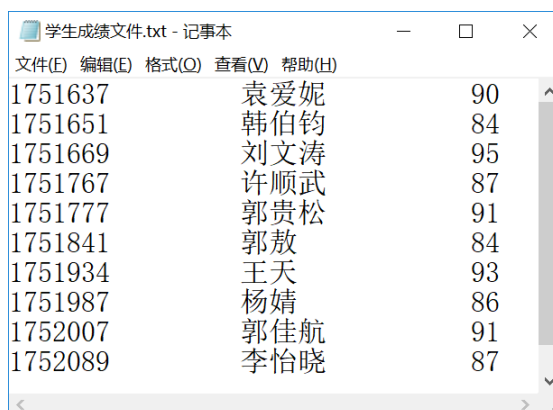
# 实验七 文件

## 一、实验目的

1. 掌握文件文件的特点和使用。
2. 掌握文件的打开、关闭和读写操作。
3. 学会在应用程序使用文件。

## 二、实验内容

1. 打开一个文本文件 T1.txt，读出文件的内容，然后进行如下处理：
  - ① 统计并且输出文件的行数；
  - ② 将文件的大小写字母互相转换后写入文件 T2.txt，即大写的转成小写的，小写的转换成大写的，其余不变
2. 打开如下图 2.7.1 所示的学生成绩文件.txt，统计平均分、最低分、平均分、最高分。



1751637	袁爱妮	90
1751651	韩伯钧	84
1751669	刘文涛	95
1751767	许顺武	87
1751777	郭贵松	91
1751841	郭敖	84
1751934	王天	93
1751987	杨婧	86
1752007	郭佳航	91
1752089	李怡晓	87

图 2.7.1 所示的学生成绩文件.txt

3. 从互联网上下载《红楼梦》的某一回组成文本文件 hlm.txt，然后设计一程序统计林黛玉和贾宝玉两个人名在文件中出现的次数。
4. 编写一个职工奖金处理程序。要求如下：
  - ① 输入不超过 10 个职工的工号、姓名和奖金，保存在文件 JJ.txt。文件中的数据格式：  
90813,张大海,1080.50 元
  - ② 从 JJ.txt 中读出数据，然后按奖金从低到高排序，保存在文件 NewJJ.txt。
5. 编写一个将两个文本文件的内容合并的程序。

# 实验八 面向对象程序设计基础

## 一、实验目的

1. 掌握类的定义和实例化的方法。
2. 学会设计简单的类。
3. 掌握利用类的继承关系定义派生类的方法。

## 二、实验内容

1. 设计一个楼房类 (**Building**)，包含楼的长、宽、层数及每平方米单价属性，并具有求楼房的面积及总价等功能。最后编写利用该类的主程序。

2. 定义一个 **identifier** 类，具有一个属性 **id**，存放身份证号，以及如下函数：

(1) **getyear** 函数用于从身份证号码中提取出生年份；

(2) **Disp** 方法用于输出身份证号码。

编写利用该类的主程序。

3. 设计一个矩形类 (**Rect**)，具有长、宽属性，类还具有求解并显示矩形的周长和面积的功能以及求两个矩形面积和的功能。

提示：假定有两个矩形 **r1** 和 **r2**，求两矩形面积之和的方法是 **Sum**，则调用方法为：

**r1.Sum(r2)**                      **r2.Sum(r1)**

4. 设计一个 **Person** 类，具有如下的成员：

(1) 属性：**Name** 和 **Age**，表示一个人姓名和年龄；

(2) 方法：**Disp**，输出各数据成员

最后编写一个使用该类的测试程序。

5. 从题 4 派生类 **Teacher**，并添加下列成员：

(1) 属性：**No** 和 **Ta**，分别表示教师的工号和教龄；

(2) 方法：**NewDisp**，输出所有成员的值。

最后编写利用该类的主程序。

# 实验九 数据可视化

## 一、实验目的

1. 掌握 Python 的绘图区和子绘图区的使用。
2. 熟悉常用图表的绘制方法。
3. 掌握图表常用组成元素的显示输出。

## 二、实验内容

1. 创建一个 2 行 1 列的绘图区并在第 1 行第 1 列绘制函数  $f(x)=x^2$  的曲线图 ( $x$  的取值范围  $[-1, 1]$ )，在第 2 行第 1 列绘制函数  $f(x)=1/x$  的曲线图 ( $x$  的取值范围  $[0, 1]$ )，效果如图 2.9.1 所示。

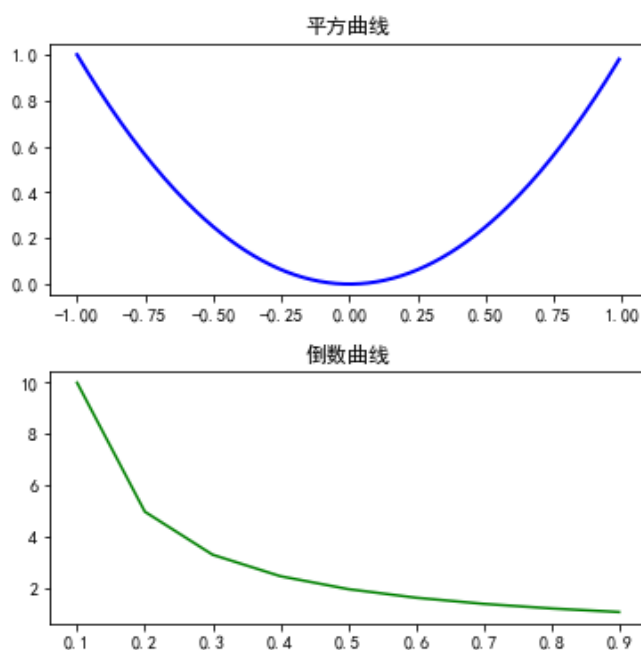


图 2.9.1 平方曲线与倒数曲线

2. 调用 `scatter` 函数绘制正弦函数的曲线，请在曲线中添加一个表示 XY 的轴线，并在 X 轴方向输出刻度标记文本，效果如图 2.9.2 所示。

提示：利用 `plot` 函数绘制直线，然后在合适位置显示标记字符。

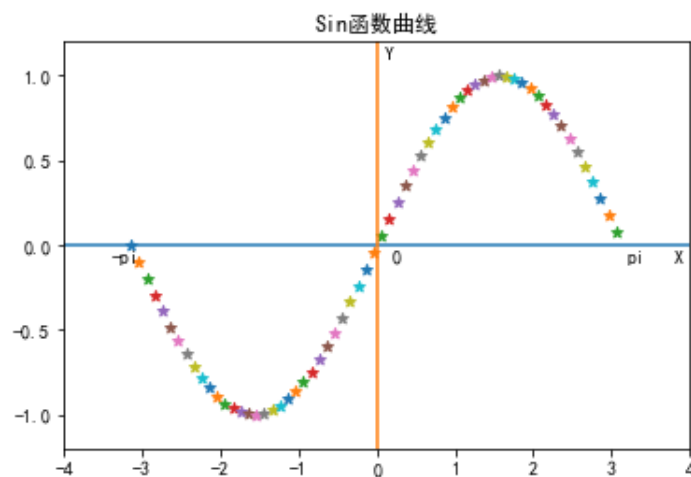


图 2.9.2 sin 函数的曲线图

3. 调用 `bar()` 函数绘制教材部分例 8.6 中的图像直方图效果，图像文件可以自己任意指定。

4. 绘制一个心形图案，参考数学函数为  $\rho=a(1-\sin\theta)$  ( $\theta$  的取值范围  $[0,2\pi]$ )， $x=\rho\cos\theta$ ， $y=\rho\sin\theta$ ，在直角坐标系上的绘制图案效果如图 2.9.3 (a) 所示，在极坐标中的绘制效果如图 2.9.3 (b) 所示。

提示：极坐标绘制函数参考格式：`plt.polar(theta,p)`

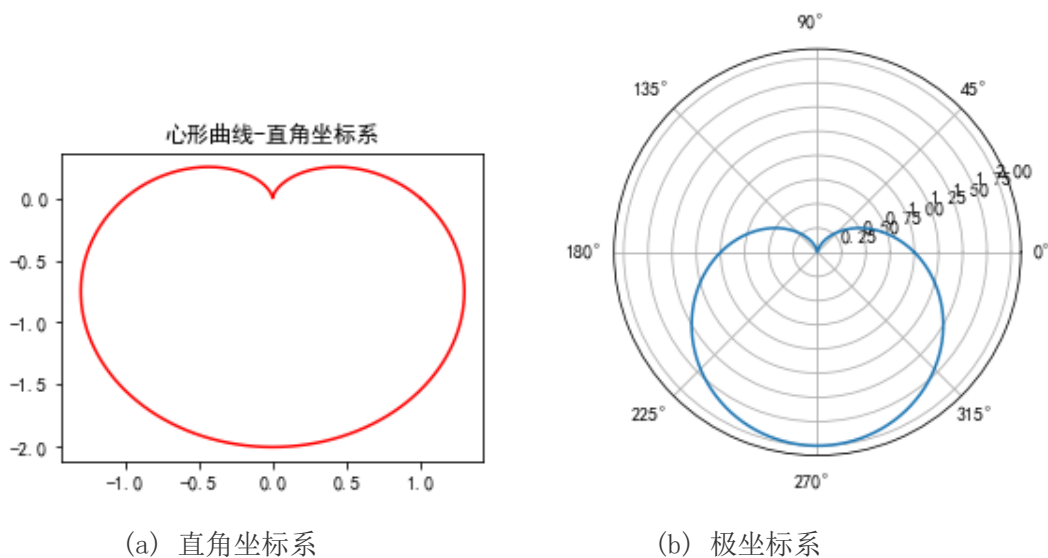


图 2.9.3 心形图案

5. 绘制一个饼图，显示你每个月各项消费支出的比例，消费支出主要包括：学习用品、日常用品、伙食费、通讯费、娱乐费和其他开支。

6. 读取“三国人名汇总.txt”中的人物名字，读取“三国演义.txt”的全部内容，先统计所有人物的名字在书本中出现的次数，并对出现次数超过 100 次的人物绘制一个柱状图，然后根据人物的词频大小绘制一个三国人名的词云图，效果如图 2.9.4 所示。（数据文件下载链接：<https://pan.baidu.com/s/1b81lMIx3In-V64vs7bsHkw>，提取码：jg41）

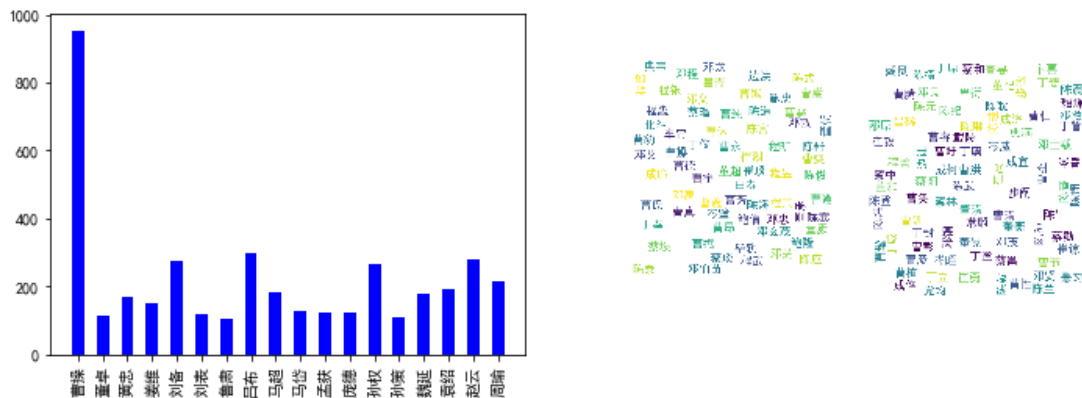


图 2.9.4 柱状图和词云图

7. 神经网络模型在训练过程中将显示每一个训练回合所用的时间、训练集上的损失值、训练集上的准确率、验证集上的损失值和验证集上的准确率，把这些数据保存到变量 history, 训练结束后对这些数据进行分析，图 2.9.5 显示了前面 5 个回合的训练数据。

```
In [*]: history=sjbModel.fit(train_images1, train_labels1, epochs=50, batch_size=128, validation_data=(partial_train_images, partial_train_labels))

Epoch 1/50
157/157 [=====] - 168s 1s/step - loss: 0.7225 - accuracy: 0.6906 - val_loss: 0.4364 - val_accuracy: 0.8307
Epoch 2/50
157/157 [=====] - 165s 1s/step - loss: 0.3391 - accuracy: 0.8667 - val_loss: 0.3285 - val_accuracy: 0.8570
Epoch 3/50
157/157 [=====] - 167s 1s/step - loss: 0.2395 - accuracy: 0.9071 - val_loss: 0.3291 - val_accuracy: 0.8577
Epoch 4/50
157/157 [=====] - 169s 1s/step - loss: 0.1840 - accuracy: 0.9301 - val_loss: 0.1635 - val_accuracy: 0.9405
Epoch 5/50
157/157 [=====] - 168s 1s/step - loss: 0.1465 - accuracy: 0.9449 - val_loss: 0.1102 - val_accuracy: 0.9615
```

图 2.9.5 神经网络模型训练过程

现将 50 个训练回合产生的数据写入到文件 Epoch.txt 中。

```
#写入数据文件的参考代码如下：
f=open(r'.\epoch.txt', "w")
loss = history.history['loss'] #训练集的损失
acc= history.history['accuracy'] #训练集的准确率
val_loss = history.history['val_loss'] #验证集上的损失
val_acc = history.history['val_accuracy'] #验证集上的准确率
for i in range(len(acc)):
    f.write(str(loss[i]) + '\t')
    f.write(str(acc[i])+ '\t')
    f.write(str(val_loss[i])+ '\t')
    f.write(str(val_acc[i])+ '\t')
    f.write('\n')
f.close()
```

用记事本中打开 Epoch.txt, 前面 5 行数据的显示结果如图 2.9.6 所示，其中第 1 列为训练集的损失值，第 2 列为训练集的准确率，第 3 列为验证集上的损失值，第 4 列为验证集上的准确率。

```
epoch.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0.7225214242935181      0.6905999779701233      0.4364478290081024      0.8307499885559082
0.33910927176475525    0.8666999936103821      0.32848525047302246      0.8569999933242798
0.2395269274711609     0.9071499705314636      0.32913365960121155      0.8577499985694885
0.18396182358264923    0.9301499724388123      0.1634775698184967       0.940500020980835
0.146547332406044      0.9449499845504761      0.11018671840429306      0.9614999890327454
```

图 2.9.6 训练产生的部分数据

下面请读入 Epoch.txt 文件中的数据,可视化显示训练集与验证集的准确率及损失值的变化趋势,可视化显示的结果将有助于判别本次训练的神经网络模型其拟合情况,输出结果如图 2.9.7 所示。(Epoch.txt 文件的下载链接为

<https://pan.baidu.com/s/1KukeCMtSkAhzgDsRVbtwzg>, 提取码为: iqjy)

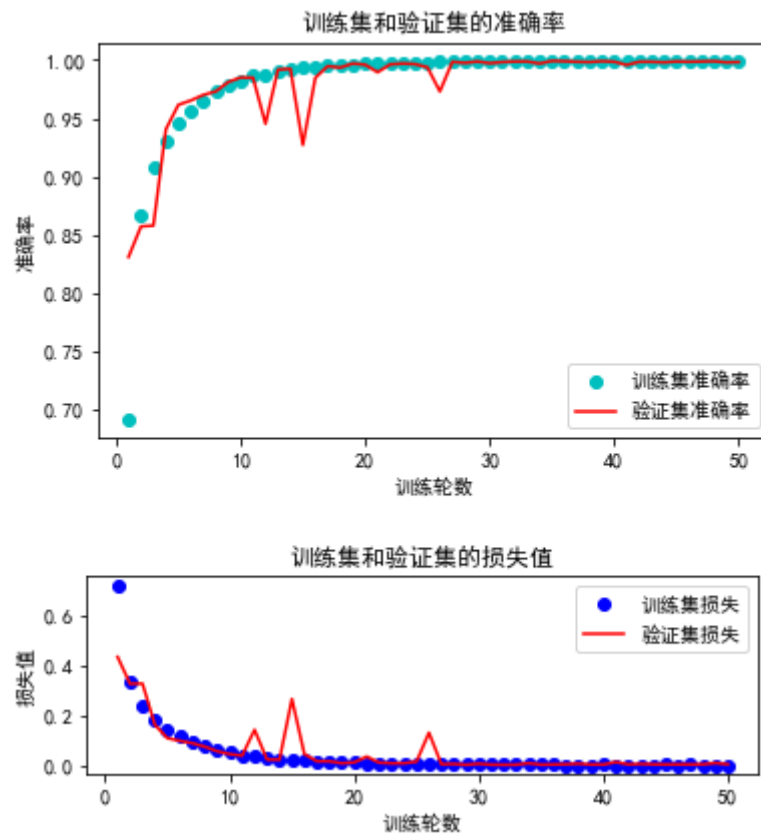


图 2.9.7 训练集与验证集的准确率与损失值可视化结果

# 实验十 数据库应用

## 一、实验目的

1. 理解访问数据库的基本过程。
2. 掌握实现 Python 对 Access 数据库访问的主要对象的创建和使用。
3. 掌握实现数据库查询的主要方法。
4. 掌握实现数据库维护的主要方法。
5. 掌握实现 Python 对 SQLite 数据库的访问。

## 二、实验内容

1. 下载数据库 plant.accdb, 数据库包含一些公园在 2021 年 3 月引入植物的信息表“Info”和种类表“Cat”。 Info 为详细植物信息表, 表结构包括: 编号(文本型)、中文名(文本型)、公园(文本型)、引入日期(日期/时间型)、高度(数字型)。 Cat 为植物类型及养护情况表, 表结构包括: 类型(文本型)、中文名(文本型)、养护(文本型)。编写程序, 用 Recordset 对象打开数据库中的 Info 表, 并遍历显示其所有记录。

2. 编写程序, 用 Recordset 对象向 plant 数据库 Info 表中添加一条记录(CSH0030,垂丝海棠,闸北公园, 2021.03.20,1350), 并查询 2021 年 3 月 20 日(包含)之后的每个公园引入的植物总数量, 显示查询结果。

3. 编写程序, 用 Connection 对象执行下列查询和维护语句。

(1) 查询所有落叶植物(养护属性为落叶)的中文名、平均高度, 结果按平均高度降序排列;

(2) Info 表中添加一条记录(BJJ0052,八角金盘,顾村公园, 2021.03.25,450);

(3) 在 Info 表中将 2021 年 3 月 18 日(不包含)之前引入的植物高度提高 10%。

4. 编写程序, 用 Command 对象执行下列查询语句, 并显示结果。

(1) 从信息表查询每天入库的植物总数量;

(2) 在 Info 表中删除编号为 LME0033 的记录;

(3) 将 Info 表中将引入日期在 2021 年 3 月 12 日(包含)之前, 高度小于 500(不包含)的植物记录的高度提高 5%。

5. 编写程序, 反复由用户输入不同的参数, 通过 Command 对象和 Parameter 对象执行查询语句, 并显示结果。

(1) 查询所有高度大于(不包含)n 的植物编号、引入日期和养护信息, 并按引入日期降序排列;

(2) 查询某年某月某日(不包含)之后引入的植物中文名、公园、引入日期、养护, 并按引入日期升序排列;

- (3) 在信息表中将某公园引入的植物高度增高 10%。
6. 编写程序，创建 SQLite 数据库 plantDB，新建 Info 表和 Cat 表，表中字段名使用相应的英文字符串代替。将第 1 题中下载的 plant.accdb 数据逐一添加到 plantDB 的两个表中。
7. 编写程序，通过 sqlite3 模块的方法，实现第 4 题的 3 个功能。