第 3 章

数据结构

第 2 章介绍的字符串是 Python 的一种序列类型。所谓序列，是指一块可存放多个值的连续内存空间，这些值按一定顺序排列，可通过每个值所在位置（称为索引）访问它们。Python 支持多种序列类型，除字符串外，还有列表、元组、集合和字典等。

3.1 列表

列表（list）是 Python 的一个内置对象类型，它具有强大的功能。Python 程序员在处理一组数据时，往往会首先考虑使用 list。

本节首先介绍列表的索引、切片和修改等基本操作。在此基础上，读者可以参照 2.3 节的方法，使用 dir()和 help()函数详细研究列表的使用方法。为了更有针对性，本节还会介绍一些列表的特定功能，如排序、堆栈和队列等。

3.1.1 列表的索引和切片

在 Python 中，列表表现为用中括号（方括号）括起来，用逗号分隔的一系列值。例如，值小于 10 的斐波那契子序列可用列表表示为

```
>>>fib=[1,1,2,3,5,8]
>>>print(fib)
[1, 1, 2, 3, 5, 8]
```

像字符串一样，列表以及其他大多数内建的序列类型，都可以被索引和切片。

```
>>>fib[0]
1
>>>fib[2]
```

```
2
>>>fib[-1]
8
>>>fib[0:3]
[1, 1, 2]
>>>fib[-2:]
[5, 8]
```

列表的切片操作返回一个新列表，而不是原列表的局部，如下的切片操作返回列表一个新的副本。

```
>>>fib[:]
[1, 1, 2, 3, 5, 8]
```

in 和 not in 操作符用于判断一个元素是否在列表中。

```
>>>fib=[1,1,2,3,5,8]
>>>1 in fib
True
>>>3 in fib
True
>>>6 in fib
False
>>>6 not in fib
True
```

需要说明的是，in 和 not in 操作符适用于所有的 Python 序列类型，包括字符串、元组、字典和集体等，在后续介绍相关内容时，对 in 和 not in 操作符不再赘述。

3.1.2　列表的修改

与字符串不同的是，列表是可变的，可以修改它的元素。

1. 修改列表的元素和切片

既可以修改单个元素，也可以对切片进行修改。

```
>>>fib=[1,1,2,3,5,8]
>>>fib[0]=0 #得到一个不正确的斐波那契数列
>>>fib
[0, 1, 2, 3, 5, 8]
>>>fib[0:2]=[1,1] #恢复正确
>>>fib
[1, 1, 2, 3, 5, 8]
```

2. 列表的连接与追加

列表也支持连接操作。

```
>>>fib=fib+[13,21,34]
>>>fib
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34]
```

但只能是列表的连接，不能直接连接元素。

```
>>>fib=fib+55
Traceback(most recent call last):
  File "<pyshell#27>", line 1, in <module>
    fib=fib+55
TypeError: can only concatenate list (not "int") to list
```

可以使用 append 方法在列表的末尾添加新的元素。

```
>>>fib.append(55)
>>>fib
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

append 方法只能追加元素。

```
>>>fib.append([89,144])
>>>fib
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, [89, 144]]
```

上述示例说明两个问题。

(1) 列表的元素可以是不同类型，例如：

```
>>>languages=[1,'Python',2,'C#']
>>>languages
[1, 'Python', 2, 'C#']
```

(2) 列表可以嵌套，即允许一个列表中包含其他列表，列表可以是另一个列表的元素。例如：

```
>>>a=['a', 'b', 'c']
>>>n=[1, 2, 3]
>>>x=[a, n] #x 中有两个元素，每个元素都是一个列表
>>>x
[['a', 'b', 'c'], [1, 2, 3]]
>>>x[0]
['a', 'b', 'c']
>>>x[0][1]
'b'
```

上述 fib.append([89,144])其实是将一个列表作为新的元素追加到原列表的末尾处。要想将一个列表的所有元素追加到另一个列表的末尾，除上述的连接操作外，还可以使用 extend 方法。

```
>>>fib=[1,1,2,3,5,8]
>>>fib.extend([13,21,34])
>>>fib
[1, 1, 2, 3, 5, 8, 13, 21, 34]
```

3. 列表的插入

除在末尾处追加外，还可以使用 insert 方法，向列表的任意位置插入新元素。插入时需要指定新元素的索引和值。

```
>>>fib=[1,1,2,3,5,13,21]
>>>fib.insert(5,8)
>>>fib
[1, 1, 2, 3, 5, 8, 13, 21]
```

4. 列表的删除

从列表中删除元素有多种方法。如果知道要删除的元素在列表中的位置，可以使用 del 语句。

```
>>>fib=[1,1,2,3,5,8,13,21]
>>>del fib[5]
>>>fib
[1, 1, 2, 3, 5, 13, 21]
```

del 语句还可以从列表中删除切片或清空整个列表。

```
>>>fib=[1,1,2,3,5,8,13,21]
>>>del fib[1:3]
>>>fib
[1, 3, 5, 8, 13, 21]
>>>del fib[:]
>>>fib
[]
```

del 语句也可以删除整个变量。删除整个变量后，再试图引用该变量会引发错误，直到另一个值赋给它为止。

```
>>>del fib
>>>fib
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    fib
NameError: name 'fib' is not defined
```

注意：del 不仅能删除列表，还能删除任何变量。

pop 方法可删除列表末尾的元素，已删除的元素作为 pop 方法的返回值，可以继续使用。

```
>>>fib=[1,1,2,3,5,8,13,21]
>>>a=fib.pop()
>>>a
21
>>>fib
```

```
[1, 1, 2, 3, 5, 8, 13]
```

pop 方法一般是在将列表作为堆栈时使用，此时插入和删除都在列表末尾处进行。一般 pop 方法不带参数。实际上，可以使用 pop 方法来删除列表中任何位置的元素，只要在 pop 方法中带上参数指明元素位置即可。

```
>>>fib=[1,1,2,3,5,8,13,21]
>>>a=fib.pop(5)
>>>a
8
>>>fib
[1, 1, 2, 3, 5, 13, 21]
```

如果不知道要删除元素的位置，只知道要删除元素的值，可以使用 remove 方法。

```
>>>fib=[1,1,2,3,5,8,13,21]
>>>fib.remove(8)
>>>fib
[1, 1, 2, 3, 5, 13, 21]
```

如果列表中有多个等值的元素，remove 方法一次只能删除一个。

```
>>>fib.remove(1)
>>>fib
[1, 2, 3, 5, 13, 21]
```

3.1.3 列表排序

有时需要将列表中无序的元素重新排列使其有序，Python 提供多种方法对列表排序，读者可根据需要选用。

1. 使用 sort 方法对列表进行永久性排序

sort 方法对列表中的元素进行原地排序。

```
>>>cars=['haval','byd','bmw','audi']
>>>cars.sort()
>>>print(cars)
['audi', 'bmw', 'byd', 'haval']
```

可以看出，经过 sort()排序后，列表的元素次序已经永久性地改变。

在调用方法 sort 时如果加上参数 reverse＝True，可以按与字典排序相反的次序排序。

```
>>>cars.sort(reverse=True)
>>>print(cars)
['haval', 'byd', 'bmw', 'audi']
```

2. 使用函数 sorted()对列表进行临时排序

要保留列表元素原来的排列次序，同时用排序后的结果显示，可以使用函数 sorted()。

```
>>>cars=['haval','byd','bmw','audi']
>>>#原始列表
>>>print(cars)
['haval', 'byd', 'bmw', 'audi']
>>>#排序后的结果
>>>print(sorted(cars))
['audi', 'bmw', 'byd', 'haval']
>>>#原始列表并未改变
>>>print(cars)
['haval', 'byd', 'bmw', 'audi']
```

可以看出,执行函数 sorted()以后,原始列表并未改变。同样可以为函数 sorted()加参数 reverse=True。

3. 使列表逆序

要反转列表元素的排列次序,可以使用 reverse 方法。

```
>>>cars=['haval','byd','bmw','audi']
>>>cars.reverse()
>>>print(cars)
['audi', 'bmw', 'byd', 'haval']
```

可以看出,逆序是永久性的。

3.1.4　堆栈和队列

列表的 append 和 pop 方法,使得列表可以很方便地作为堆栈来使用。堆栈中最先进入的元素最后被释放(后进先出),是非常常用的数据结构。用 append 方法可以把一个元素添加到堆栈顶。用不指定索引的 pop 方法可以把一个元素从堆栈顶释放出来。

```
>>>stack=[3, 4, 5]
>>>stack.append(6)
>>>stack.append(7)
>>>stack
[3, 4, 5, 6, 7]
>>>stack.pop()
7
>>>stack
[3, 4, 5, 6]
>>>stack.pop()
6
>>>stack.pop()
5
>>>stack
[3, 4]
```

还可以把列表当作队列使用。队列中最先进入的元素最先释放(先进先出),也是非常

常用的数据结构。不过,列表这样用效率不高。虽然向列表末尾添加元素很快,但在头部弹出元素效率并不高,这是由列表的内部实现决定的(弹出头部元素,要移动整个列表中的所有元素)。

如果要实现队列,可以使用 collections.deque,它是为在首尾两端快速插入和删除而设计的。

```
>>>from collections import deque
>>>queue=deque(["Zhang", "Wang", "Li"])
>>>queue.append("Zhao")        #Zhao 来了
>>>queue.append("Qian")        #Qian 来了
>>>queue.popleft()             #最先到达的人离开
'Zhang'
>>>queue.popleft()             #第二个到达的人离开
'Wang'
>>>queue                       #仍然保持人员的到达次序
deque(['Li', 'Zhao', 'Qian'])
```

collections 是 Python 的一个标准库,提供了许多有用的集合类。collections 中的 deque 类似于 list,但在队列头部和尾部添加、删除元素的效率更高。除 deque 之外,collections 还提供元组、字典和集合等序列的特色实现,与 Python 的原生数据结构相比,功能往往更强大,针对性更强,有兴趣的读者可以自行了解。

3.2 元组

列表是可以修改的,非常适合存储在程序运行期间可能变化的数据集。如果需要创建一系列不可修改的元素,如用于参数传递,再如取数据库中的一条记录,可以使用元组。与列表相比,元组的数据结构更简单,处理速度更快。

1. 元组的声明

元组看起来像列表,但使用圆括号而不是方括号来标识。元组的元素类型可以互不相同,元素索引及切片的规定与列表相同。

```
>>>a=(123,321,'Python')
>>>a
(123, 321, 'Python')
>>>a[0]
123
>>>a[2]
'Python'
>>>a[1:]
(321, 'Python')
```

元组在输出时总是有括号,以便正确表现元组的逻辑结构。在输入时可以不用圆括号。

```
>>>b=1,'Zhang',28
>>>b
(1, 'Zhang', 28)
```

但在编程过程中,使用圆括号往往是必需的,例如元组是一个更大的表达式的一部分,不使用圆括号可能会产生混淆。所以,在任何情况下都使用圆括号是好习惯。

可以用 len()函数查看元组的长度(元素数)。

```
>>>a=(123,321,'Python')
>>>len(a)
3
```

同样可以用 in 和 not in 操作符判断一个元素是否在元组中。

```
>>>123 in a
True
>>>222 in a
False
```

2. 尝试修改

元组是不可修改的,不能给元组的一个独立的元素赋值。

```
>>>a=(123,321,'Python')
>>>a[0]=111
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    a[0]=111
TypeError: 'tuple' object does not support item assignment
```

但元组变量可以重新赋值。

```
>>>a=(123,321,'Python')
>>>a=(123,321)
>>>a
(123, 321)
```

3. 元组的封装和拆封

为了方便多值与序列的转换,Python 支持元组的封装和拆封操作。前述示例

```
>>>b=1, 'Zhang', 28
```

其实就是元组的封装,3 个值被自动封装进元组。封装的逆操作称为拆封。

```
>>>c, d, e=b
>>>c
1
>>>d
'Zhang'
>>>e
28
```

在上述示例中，元组的 3 个元素被分别赋值给左侧的 3 个变量。

事实上，等号右边可以是任何线性序列，如列表。

```
>>>b=[1, 'Zhang', 28]
>>>b
[1, 'Zhang', 28]
>>>c, d, e=b
>>>c
1
>>>d
'Zhang'
>>>e
28
```

但实际编程中，元组的封装与拆封使用得最多，如可变参数和多返回值等，详见第 5 章。需要注意的是，拆封要求左侧的变量数量与序列的元素个数严格相同。

```
>>>b=(1, 'Zhang', 28)
>>>c, d=b
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    c, d=b
ValueError: too many values to unpack (expected 2)
```

3.3 字典

另一个非常有用的 Python 序列是字典。字典和列表有相似之处，列表使用[]来定义，而字典使用{}，元素都是用逗号(,)分隔。不同的是，列表的索引是从 0 开始的有序整数，而字典的索引是键(关键字)。字典的键可以是任意不可变类型，如数字、字符串和元组等，一般用字符串，键与值之间用冒号(:)分隔。不能用列表做关键字，因为列表可以用索引、切割或者 append 和 extend 等方法来改变。与前面介绍的序列类型不同，字典以及 3.4 节将要介绍的集合，都不支持索引、切片和相加等操作。

```
>>>dict1={'姓名': '张三', '年龄': 19, '年级': '大二', '成绩': '及格'}
>>>print(dict1)
{'姓名': '张三', '年龄': 19, '年级': '大二', '成绩': '及格'}
```

字典中的键必须是唯一的，且不可变；字典中的值可以不唯一，且可变。字典的最常用操作是依据键来存取值。

```
>>>dict1['姓名']
'张三'
```

可以使用 values 方法访问所有值。

```
>>>dict1.values()
```

```
dict_values(['张三', 19, '大二', '及格'])
>>>type(dict1.values())
<class 'dict_values'>
>>>list(dict1.values())
['张三', 19, '大二', '及格']
```

在 Python 的当前版本中，values 方法返回的是一种称为 dict_values 的序列类型，它支持成员测试以及迭代等操作。可以使用 list()函数创建值的列表。

可以使用 keys 方法访问所有键。

```
>>>dict1.keys()
dict_keys(['姓名', '年龄', '年级', '成绩'])
>>>type(dict1.keys())
<class 'dict_keys'>
>>>list(dict1.keys())
['姓名', '年龄', '年级', '成绩']
```

keys 方法返回的是一种称为 dict_keys 的类型，也可以使用 list()函数创建键列表。

可以使用 items 方法来访问字典中的所有键和值。

```
>>>dict1.items()
dict_items([('姓名', '张三'), ('年龄', 19), ('年级', '大二'), ('成绩', '及格')])
>>>type(dict1.items())
<class 'dict_items'>
>>>list(dict1.items())
[('姓名', '张三'), ('年龄', 19), ('年级', '大二'), ('成绩', '及格')]
```

items 方法返回的是一种称为 dict_items 的类型。在 dict_items 中，每个元素是一个由键和值组成的元组，同样可以使用 list()函数创建字典元素列表。

字典是可修改的，可以向字典中添加新的元素，修改字典中原有元素的值，或者删除字典中的某一个元素。

```
>>>dict1['性别']='男'      #添加新元素
>>>print(dict1)
{'姓名': '张三', '年龄': 19, '年级': '大二', '成绩': '及格', '性别': '男'}
>>>dict1['成绩']='优秀'    #修改原有元素的值
>>>print(dict1)
{'姓名': '张三', '年龄': 19, '年级': '大二', '成绩': '优秀', '性别': '男'}
>>>del dict1['成绩']        #删除一个元素
>>>print(dict1)
{'姓名': '张三', '年龄': 19, '年级': '大二', '性别': '男'}
```

从上面的示例能够看出，可以用 del 删除字典项。但是，试图从一个不存在的键中取值，或者删除一个不存在的字典项，都会导致错误。

```
>>>dict1['民族']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
```

```
    dict1['民族']
KeyError: '民族'
```

使用 clear 方法可以删除字典内所有元素。

```
>>>dict1.clear()
>>>print(dict1)
{}
```

一对大括号{}创建一个空的字典。

```
>>>dict2={}
>>>print(dict2)
{}
>>>dict2['语言']='Python'
>>>print(dict2)
{'语言': 'Python'}
```

dict()构造函数可以直接从键值对列表创建字典。

```
>>>dict3=dict([('语言','Python'), ('平台','Windows'), ('数据库','MySQL')])
>>>print(dict3)
{'语言': 'Python', '平台': 'Windows', '数据库': 'MySQL'}
```

如果键是字符串,有时通过关键字参数指定键值对更方便。

```
>>>dict4=dict(语言='Python', 平台='Windows', 数据库='MySQL')
>>>print(dict4)
{'语言': 'Python', '平台': 'Windows', '数据库': 'MySQL'}
```

3.4 集合

集合是一个无序不重复元素的序列,分为以下两种。

(1) set:可变集合。集合中的元素可以动态地增加或删除。

(2) frozenset:不可变集合。集合中的元素不可改变。

可以说 frozenset 是 set 的不可变版本,本节介绍 set。

集合可以用大括号{}或 set()函数来创建。如果想要创建空集合,只能用 set()函数,因为{}用来创建空字典。

```
>>>set1={'Python','C++','Java'}
>>>print(set1)
{'Java', 'Python', 'C++'}
>>>set2=set(['Python','C++','Java','Java'])
>>>print(set2)
{'Java', 'Python', 'C++'}
>>>set3=set('abracadabra')
```

```
>>>print(set3)
{'c', 'a', 'r', 'd', 'b'}
>>>set4=set('alacazam')
>>>print(set4)
{'c', 'a', 'l', 'z', 'm'}
>>>set5=set()
>>>print(set5)
set()
```

上面的示例演示了集合的创建方法，同时演示了集合的去重复特性。试图通过索引访问集合元素会引发错误。

```
>>>set1={'Python','C++','Java'}
>>>set1[0]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    set1[0]
TypeError: 'set' object is not subscriptable
```

集合中的元素可以动态地增加或删除。

```
>>>set1={'Python','C++','Java'}
>>>set1.add('C#')
>>>set1
{'C#', 'Python', 'C++', 'Java'}
>>>set1.remove('Java')
>>>set1
{'C#', 'Python', 'C++'}
```

集合支持并(运算符|)、交(运算符 &)、差(运算符－)和异或(运算符^)等数学运算。

```
>>>set3=set('abracadabra')
>>>set3
{'c', 'a', 'r', 'd', 'b'}
>>>set4=set('alacazam')
>>>set4
{'c', 'a', 'l', 'z', 'm'}
>>>set3 | set4
{'c', 'a', 'r', 'd', 'l', 'z', 'm', 'b'}
>>>set3 & set4
{'c', 'a'}
>>>set3-set4
{'d', 'b', 'r'}
>>>set3 ^ set4
{'z', 'm', 'b', 'r', 'd', 'l'}
```

3.5 Python 集成开发环境

本书前面的示例，每个命令都只有一行，而真正的 Python 程序会大量使用多行命令。在命令行状态下（无论是 Python 解释器还是 IDLE），输入多行命令都需要一定技巧，见 1.3 节。在命令行状态下，输入多行命令比较麻烦，更不用说调试和纠错了。

使用 IDLE 或者 Python 解释器非常适合简单程序，但对于大型的编程项目则力不从心，这时就需要选择一个合适的集成开发环境。

集成开发环境（Integrated Development Environment，IDE）是专用于软件开发的程序。IDE 将支持软件开发的一系列工具集成到一起，这些工具至少包括一个专门处理代码的编辑器（如提供语法高亮和自动补全等功能），还应该包括构建、执行、调试和版本控制工具等。

3.5.1 集成开发环境介绍

能够进行 Python 程序开发的集成开发环境很多，本节介绍几款 Windows 环境下常用的 Python 语言集成开发环境。

1. Visual Studio

由 Microsoft 建立的 Visual Studio 是一款全功能集成开发平台。Visual Studio 仅兼容 Windows 和 Mac OS 操作系统，它既提供免费版（社区版）也提供付费版（专业版和企业版）。随着 Visual Studio 版本的升级，对 Python 的支持度越来越高。

2. Visual Studio Code

与完全版的 Visual Studio 不同，Visual Studio Code（也称作 VS Code）是一款兼容 Linux、Mac OS X 和 Windows 平台的全功能代码编辑器。在 Anaconda 中，VS Code 是选装组件，使用非常方便，Anaconda 的介绍见 6.5 节。

3. PyCharm

PyCharm 是专门面向 Python 的全功能集成开发环境。拥有付费版（专业版）和免费开源版（社区版），无论在 Windows、Mac OS X 还是 Linux 操作系统中都支持快速安装和使用。在 PyCharm 中可以直接运行和调试 Python 程序，并且支持源码管理和项目。

4. Spyder

Spyder 是一款为数据科学工作做了优化的开源 Python 集成开发环境。它附在 Anaconda 发行版中，随 Anaconda 自动安装。Spyder 的目标受众是使用 Python 的数据科学家，它很好地集成了一些诸如 SciPy、NumPy 和 matplotlib 这样的公共 Python 数据科学库。

5. Eric6

Eric6 是一个用 Python 语言编写的 Python IDE。Eric6 是用 PyQt 开发的，因此它的运行重度依赖几个 Python 包，如 PyQt 等，这意味着使用 Eric6 要先理解 Python 的包管理，对初学者稍嫌麻烦。但 Eric6 本身是用 PyQt 开发的，使用它特别适合开发 PyQt 程序。

3.5.2　PyCharm 的安装与使用

PyCharm 是 Python 语言开发中一个很受欢迎的 IDE，界面与主流大型 IDE（如 Visual Studio 等）相类似，集成的功能也很多。无论是有经验的程序员还是初学者，将 PyCharm 作为 Python 开发环境都是不错的选择。

1. 下载安装

到 JetBrains 的官网 https://www.jetbrains.com/下载 PyCharm。PyCharm 分为专业版（Professional）和社区版（Community），下载免费的社区版即可。写作本书时的最新版本是 2020.1.2，下载得到可执行文件 pycharm-community-2020.1.2.exe。

执行该文件进行安装。安装是向导式风格，本书未提示的界面，单击 Next 按钮即可。在 Choose Install Location 界面，可以改变安装路径，如图 3-1 所示。

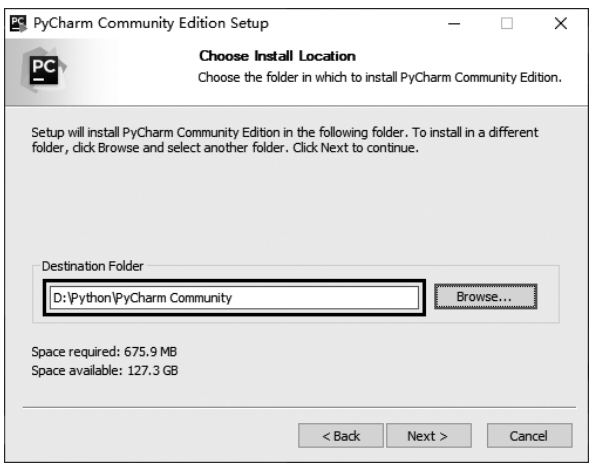

图 3-1　PyCharm 安装界面（一）

在 Installation Options 界面，选中所有复选框，如图 3-2 所示。

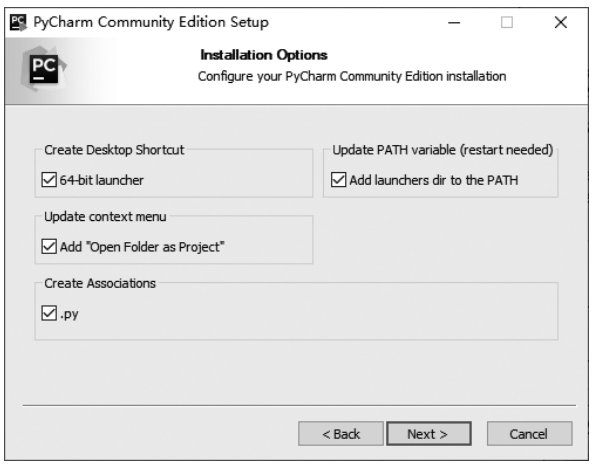

图 3-2　PyCharm 安装界面（二）

安装完成后重启计算机。

2. 首次运行配置

第一次启动 PyCharm 时需要进行一些配置，是分步骤进行的。

第一步：设置 IDE 的配置信息，如果以前没有保存，选择 Do not import settings 即可，如图 3-3 所示。

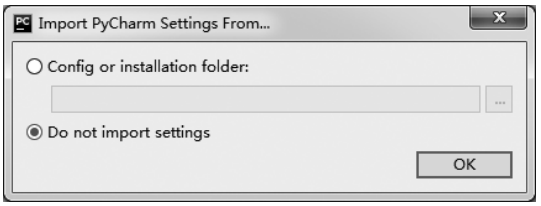

图 3-3　PyCharm 首次运行

第二步：界面主题的选择，有 Darcula 和 Light 两个选择，可以看到效果图，请读者根据自己的喜好选择。

第三步：Customize PyCharm 页，不需要做任何改变。

3. 编程示例

启动 PyCharm，首先进入欢迎界面，如图 3-4 所示。

图 3-4　PyCharm 欢迎界面

PyCharm 以目录为项目进行管理。预先创建一个目录，如 E:\PyCharmProjects，用于保存所有的 PyCharm 项目。

在图 3-4 所示的界面中选择 Create New Project 创建一个新项目，弹出如图 3-5 所示的界面。

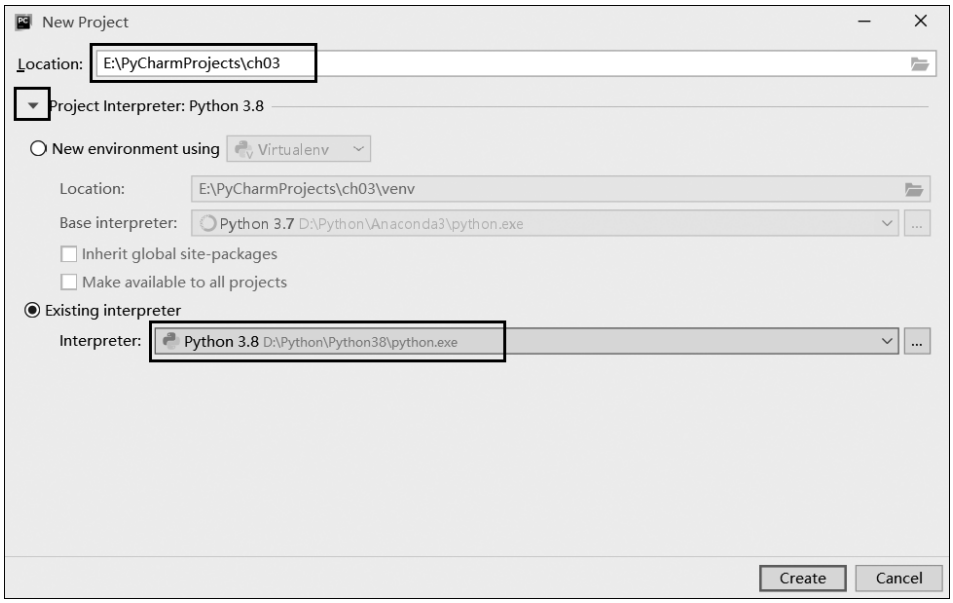

图 3-5　创建新项目

在 Location 处输入或选择想要保存项目的目录。该目录不一定要事先存在，PyCharm 会自动创建。为新项目选择一个解释程序(Interpreter)。

新项目创建成功后，自动进入 PyCharm 主界面，如图 3-6 所示。

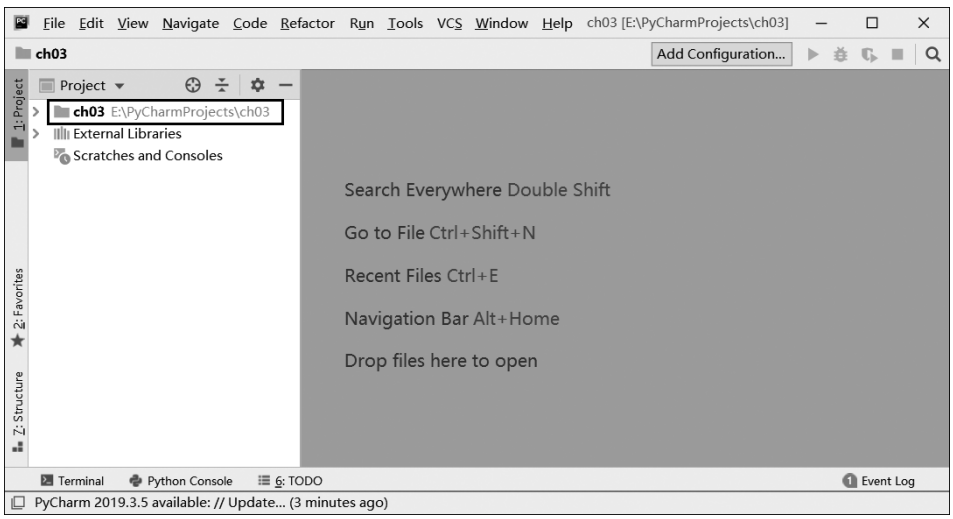

图 3-6　PyCharm 主界面

在 PyCharm 主界面上，右击项目名称，在弹出的菜单中选择 New→Python File，创建一个文件 test_while.py。在主界面上可以看到，该文件增加到了项目下，并已在右侧的编辑器中自动打开，如图 3-7 所示。在编辑器中输入代码：

```
a, b=0, 1
```

```
while b<10:
    print(b)
    a, b=b, a+b
```

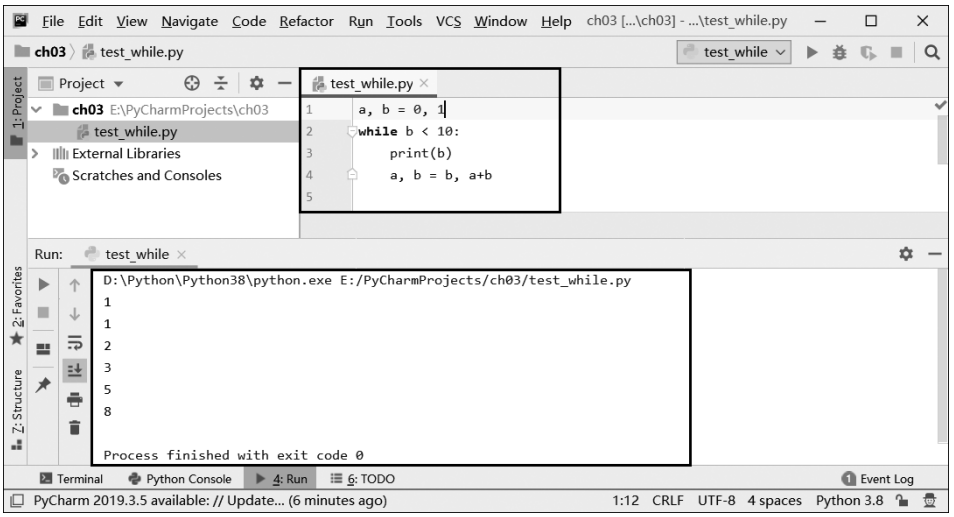

图 3-7　用 PyCharm 编程

注意：在代码结束处不需要有空行，编辑后的文件会自动保存。

在左侧该文件名称上右击，在弹出的菜单中选择“Run 'test_while'”执行程序，界面下部会显示程序执行的结果。

上面讲解了 Python 项目的开发过程。在实际开发中：

- 再次启动 PyCharm 时，自动打开上次的项目。想要切换或创建新的项目，请先在菜单中选择 File→Close Project 关闭当前项目，并自动进入欢迎界面。
- 图形化的编辑界面，程序代码可以反复修改执行，比在命令行工具中方便。
- 必要时可以使用调试(Debug)模式，跟踪程序的执行并检错纠错，这是使用 IDE 的最大优势。

在本书的后续内容中，将根据需要使用 IDLE 或者 PyCharm 来验证代码。两者各有利弊，使用 PyCharm 程序结构更清晰，使用 IDLE 更容易理解程序的执行步骤。使用 IDLE 时，代码中包含提示符；使用 PyCharm 时，会指明项目和文件名称，所以不会混淆。

3.6 习题

1. 使用 dir()和 help()函数详细研究列表的使用方法。

2. 用一个列表表示平方序列

```
sq=[1,4,9,16,25]
```

要求完成以下功能：

(1) 取第 1 个元素值。

(2) 取倒数第一个元素值。

(3) 取包含前三个元素的子列表。

(4) 检查数据 6 和 9 是否在列表中。

(5) 在列表的末尾追加一个值 38。

(6) 将刚追加的值改为 36。

(7) 将列表[49,64,81]的元素追加到原列表的末尾。

(8) 在列表的首部插入一个值 0。

(9) 删除列表中的第 5 个元素。

(10) 删除列表中值为 9 的元素。

3. 创建一个字符串列表,包含一周中每天的英文简称,如 Mon、Tue 等。要求:

(1) 输出该列表的逆序。

(2) 对该列表进行永久性排序。

(3) 用 str()函数将列表转换为字符串,观察其结果。

4. 实现字符串反转:输入 str="string",输出'gnirts'。

5. 有如下列表:

```
li=['alex','eric','rain']
```

要求完成以下功能:

(1) 计算列表长度并输出。

(2) 列表中追加元素"seven",并输出添加后的列表。

(3) 在列表的第 1 个位置插入元素 "Tony",并输出添加后的列表。

(4) 将列表第 2 个位置的元素修改为 "Kelly",并输出修改后的列表。

(5) 删除列表中的元素"rain",并输出修改后的列表。

(6) 删除列表中的第 2 个元素,并输出删除的元素的值和删除元素后的列表。

(7) 删除列表中的第 0 个元素,并输出删除元素后的列表。

(8) 删除列表中的第 2～4 个元素,并输出删除元素后的列表。

(9) 将列表所有的元素反转,并输出反转后的列表。

6. 假设有两个列表,a=[1,2,3],b=[4,5,6],请将两个列表合并。

7. 创建一个存储学生信息的元组,有 4 个字段,分别为:学号=101、姓名='xiaoming'、年龄=19、成绩=59.5。显示该生的学号和姓名。将该元组拆封到 a、b、c 和 d 的 4 个变量中,并显示。

8. 创建一个存储学生信息的字典,有 4 个元素,分别为:学号=101、姓名='xiaoming'、年龄=19、成绩=59.5。显示该生的学号和姓名;将该生的成绩改为 60;显示该字典的键列表;显示该字典的值列表;显示该字典的元素列表。

9. 将字符串"k:1|k1:2|k2:3|k3:4",处理成 Python 字典{'k':'1', 'k1':'2', 'k2':'3', 'k3':'4'}。提示:读者可自行学习 Python 内置函数 eval()的用法。

10. 定义一个篮子集合

```
basket={'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
```

要求完成以下功能:

(1) 显示 basket。

(2) 检查'crabgrass'是否在 basket 中。

(3) 将'crabgrass'添加到 basket,显示 basket。

(4) 向 basket 中再添加一个'apple',显示 basket。

(5) 删除 basket 中的'orange'。

11.有列表 li=[1,1,6,3,1,5,2],删除其中的重复元素。

提示:可以预习第 4 章内容,并使用循环语句完成;也可以使用集合作为转换的中间结果。

12. 比较列表、元组、字典和集合的特点及用法。

13. 有如下字典:

```
dic={'k1':"v1","k2":"v2","k3":[11,22,33]}
```

要求完成以下功能:

(1) 输出所有的 key。

(2) 输出所有的 value。

(3) 输出所有的 key 和 value。

(4) 在字典中添加一个键值对"k4":"v4",输出添加后的字典。

(5) 修改字典中"k1"对应的值为"alex",输出修改后的字典。

(6) 在 k3 对应的值中追加一个元素 44,输出修改后的字典。

(7) 在 k3 对应的值的第 1 个位置插入一个元素 18,输出修改后的字典。

14. 完成以下转换:

(1) 将字符串 s="alex"转换成列表。

(2) 将字符串 s="alex"转换成元组。

(3) 将列表 li=["alex","seven"]转换成元组。

(4) 将元组 tu=('alex',"seven")转换成列表。