第一部分
Python 数据可视化基础篇

本篇将循序渐进地介绍 Python 语言的基本使用方式，带领读者走进 Python 数据可视化分析的世界。本篇除了介绍 Python 基础语法的使用外，还会介绍 Python 的 3 个基础包（库）Numpy、Pandas 及 Matplotlib 的使用。通过本篇的学习，读者会对 Python 及其数据可视化功能有一定的认识，并可以获得基本的数据可视化分析能力。

第 1 章　Python 快速入门

Python 是一种简单易学且功能强大的编程语言，具有高效的数据结构，能简单而有效地实现面向对象编程。Python 有简洁的语法和对动态输入的支持，再加上具有解释性语言的特点，因而在大多数平台上和许多领域都是一种理想的脚本语言，特别适用于快速的应用程序开发。

本章主要介绍 Python 入门知识和数据可视化分析，此外还会介绍 Python 的安装和基础语法的使用。

1.1 安装 Python

为了更好地使用 Python，不仅需要安装 Python 本身，还需要安装功能丰富的 Python 库。尤其是针对数据可视化分析，通常还会使用到 Numpy、Pandas、Matplotlib 等第三方库。Anaconda 已经为我们准备好了常用库的封装。在数据分析、数据可视化及机器学习时，可以使用 Anaconda 提供的库的封装。Python 对于新手来说，界面更加友好，环境的配置更加方便。

1.1.1 安装 Anaconda

本节会介绍 Python 的安装与使用 (本书以 Anaconda 为例)，安装 Anaconda 后无须再额外安装 Python。

可从 Anaconda 官方网站选择适合自己计算机设备的 Anaconda 版本进行下载安装。截至本书撰写时，Anaconda 已经更新到 Python3.9 版本。如图 1-1 所示，在 Anaconda 的下载页面中，即可跟随指导安装 Anaconda。Anaconda 安装后的开始界面如图 1-2 所示。该界面的内容会因为计算机所安装 Anconda 应用的版本而有一些小的差异，但主要应用是相同的，其中经常被用来编写 Python 程序的应用有 Spyder、Jupyter Notebook 和 JupyterLab。

图 1-1　Anaconda 的下载页面

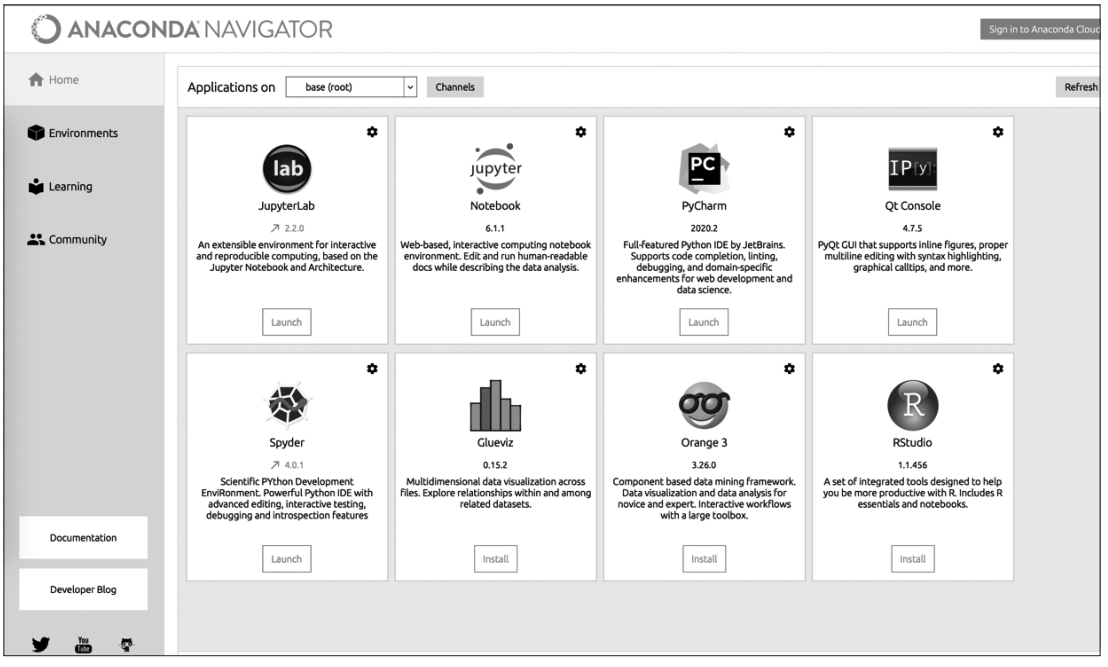

图 1-2　Anaconda 安装后的开始界面

1.Spyder

Spyder 是在 Anaconda 中附带的免费集成开发环境（Integrated Development Environment，IDE）。它包括编辑、交互式测试、调试等功能。Spyder 的操作界面类似于 Matlab。Spyder 的应用界面如图 1-3 所示。

在图 1-3 中，最上方是工具栏区域，左侧是代码编辑区域，可以编辑多个 Python 脚本；右上方是变量显示、图像显示等区域；右下方是程序运行和相关结果显示的区域。在代码编辑区域选中要运行的代码，再在工具栏区域单击 Run 按钮或按 F9 键即可运行代码。不同版本的 Anaconda 的快捷键略有不同。

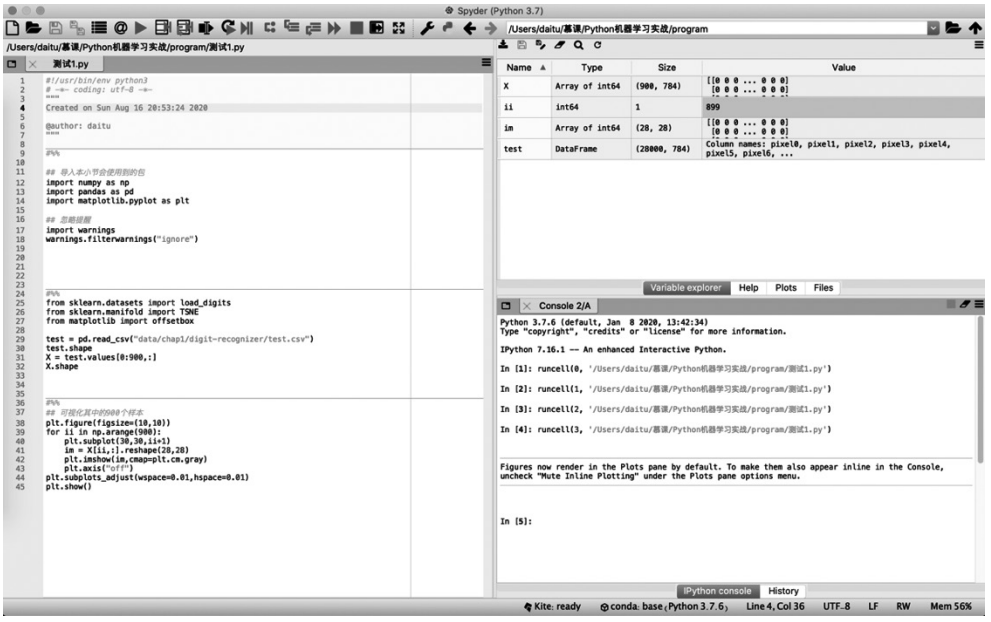

图 1-3　Spyder 的应用界面

2.Jupyter Notebook

Jupyter Notebook 不同于 Spyder。Jupyter Notebook 是一个交互式笔记本，支持运行 40 多种编程语言，并可以使用浏览器打开其中的程序。它的出现是为了方便科研人员随时将把自己的代码和文本生成 pdf 或者网页格式与其他人交流。启动 Jupyter Notebook 后，在合适的位置选择新建 Python3 文件，可获得一个新的 Notebook 文件，每个 Notebook 文件都由许多单元组成，可以在单元中编写程序。Jupyter Notebook 界面如图 1-4 所示。

图 1-4　Jupyter Notebook 界面

3.JupyterLab

JupyterLab 是 Jupyter Notebook 的升级版，在文件管理、程序查看、程序对比等方面，都比 Jupyter Notebook 的功能更加强大。JupyterLab 和 Jupyter Notebook 的程序文件是通用的，可以不进行任何修改就可以运行。打开 JupyterLab 后，JupyterLab 的使用界面如图 1-5 所示。

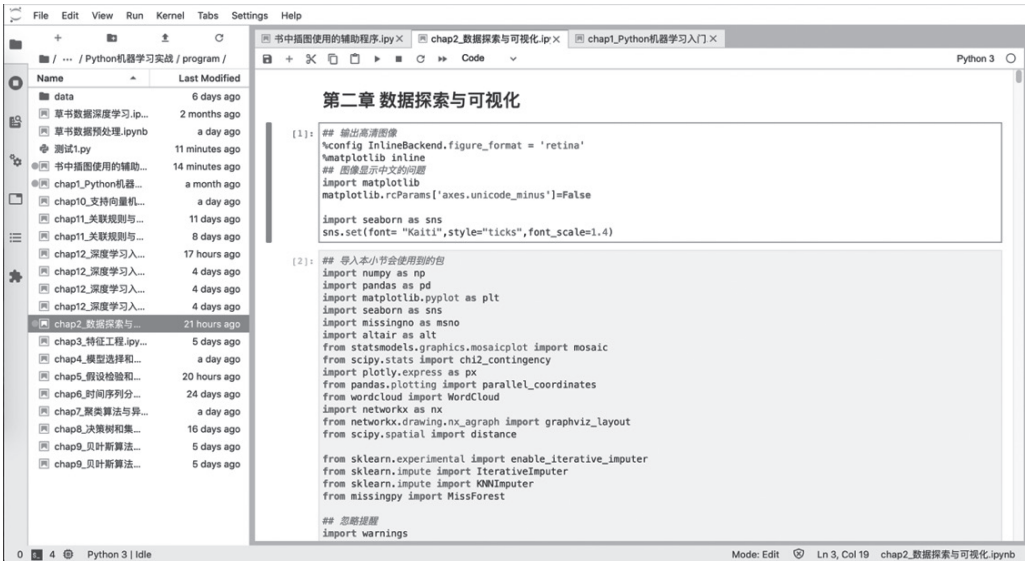

图 1-5　JupyterLab 的使用界面

1.1.2 安装 Python 库

虽然在 Anaconda 中已经提前安装好了常用的 Python 库，但是在使用 Python 进行数据分析与可视化时，会遇到重新安装所需要库的情况。下面介绍一些常用的安装 Python 库的方式。

（1）通过 conda 命令安装，通常使用如下命令。

```
conda install 库的名称
```

（2）通过 pip 命令安装，通常使用如下命令。

```
pip install 库的名称
```

（3）当指定安装库使用的镜像时，可以通过 pip 使用如下命令。

```
pip install -i https://pypi.douban.com/simple 库的名称
```

1.2 Python 的基础知识

本节介绍 Python 的基础知识，包括如何使用 Python 的列表、元组、字典与集合等数据结构，以及 Python 的条件判断语句、循环语句、函数等内容。

1.2.1 列表

列表（list）是 Python 的最基本数据类型之一。列表中的元素按照顺序排列。这个顺序即索引。索引是从 0 开始的。第一个元素索引是 0，第二个元素索引是 1，以此类推。可以通过索引获取列表中的元素。

可以通过 list 函数或者中括号“[ ]”来生成一个列表。例如，下面的程序生成了包含 2~6 这 5 个元素的列表 A，同时使用 len 函数计算列表的长度（列表 A 的长度为 5）。

```
In[1]:# 生成一个列表 A
  A = [2,3,4,5,6]
  A
Out[1]:[2, 3, 4, 5, 6]
In[2]:# 计算列表 A 的长度
  len(A)
Out[2]:5
```

生成列表 A 后，下面的程序可以通过索引获取列表 A 中的元素。其中，从左往右的索引，也就是顺序索引是从 0 开始的；从右往左的索引也叫逆序索引，是从 -1 开始的。

```
In[3]:# 从左往右的索引，从 0 开始
  A[3]
Out[3]:5
In[4]:# 从右往左的索引，从 -1 开始
  A[-2]
Out[4]:5
```

获取列表中一个范围内的元素，可以通过切片索引来完成。例如，使用切片索引“0:3”，获取索引从 0 到 3 的元素，但不包含索引为 3 的元素，即这个范围是一个左闭右开的。下面的程序可以获取列表中多个元素。

```
In[5]:# 获取列表 A 中的一段
  print(A[0:3])          # 输出的结果中不包含索引为 3 的元素
  print(A[1:-1])         # 输出的结果中不包含索引为 -1 的元素
  print(A[-4:])
Out[5]: [2, 3, 4]
        [3, 4, 5]
        [3, 4, 5, 6]
```

针对一个已经生成的列表，可以通过 append 函数在其后面添加一个新元素，并且这个新元素的数据形式可以是多种类型，包括数字、字符串、列表等。例如，下面的程序在列表 A 的末尾添加了新的数字和字符串。

```
In[6]:# 在列表 A 的末尾添加新元素
      A.append(7)              # 添加一个新元素
      A.append("eight")        # 再添加一个新元素
      A
Out[6]: [2, 3, 4, 5, 6, 7, 'eight']
```

在列表的指定位置插入一个新元素可以使用 insert 函数。该函数有两个参数：第一个参数表示插入的位置，第二个参数表示要插入的内容。例如，下面的程序在列表 A 索引为 5 的位置上插入一个字符串 Name。

```
In[7]:# 在列表 A 指定位置添加一个新元素
      A.insert(5,"Name")
      A
Out[7]: [2, 3, 4, 5, 6, 'Name', 7, 'eight']
```

删除列表中的元素可以通过 pop 函数实现。该函数可以删除指定索引号的元素，默认删除列表中的最后一个元素。例如，可以使用下面的程序删除列表 A 中最后一个元素。

```
In[8]:# 删除列表 A 末尾的元素
      A.pop()                          # 删除一个元素
      A.pop(5)                         # 删除索引为 5 的元素
      A
Out[8]: [2, 3, 4, 5, 6, 7]
```

针对列表，还可以通过 del 函数删除列表中指定位置的元素。例如，可以使用下面的程序删除列表 A 中索引为 2 的元素。

```
In[9]:# 通过 del 函数删除指定的元素
      del A[2]
      A
Out[9]: [2, 3, 5, 6, 7]
```

列表中的元素可以使用 Python 中的任何数据类型。例如，下面的程序生成列表 B，其中包含字符串和列表。

```
In[10]:# 列表中的元素还可以是列表
       B = ["A","B",A,[7,8]]
       B
Out[10]: ['A', 'B', [2, 3, 5, 6, 7], [7, 8]]
```

下面的程序可以通过加号“+”将多个列表进行元素合并（列表合并），通过星号“*”将列表的元素进行重复（列表重复），生成新的列表。

```
In[11]:# 列表合并
       [1,2,3] + [4,5,6]
Out[11]: [1, 2, 3, 4, 5, 6]
In[12]:# 列表重复
       [1,2,"three"] * 2
Out[12]: [1, 2, 'three', 1, 2, 'three']
```

下面的程序可以通过 reverse 函数获取列表的逆序；可以通过 count 函数统计列表中某元素出现的次数；可以通过 sort 函数对列表中的元素进行排序；还可以通过 min 函数和 max 函数找出列表中的最小值和最大值。

```
In[12]:# 输出列表 A 的内容
       A = [15,2,31,10,12,9,2]
       # 获取列表的逆序
       A.reverse()
```

```
        A
Out[12] [2, 9, 12, 10, 31, 2, 15]
In[13]:# 计算列表 A 中元素出现的次数
        A.count(2)
Out[13]:2
In[14]:# 对列表 A 进行排序
        A.sort()
        A
Out[14]: [2, 2, 9, 10, 12, 15, 31]
In[15]:# 获取列表 A 中的最大值和最小值
        print("A 最小值 :",min(A))
        print("A 最大值 :",max(A))
Out[15]:A 最小值 : 2
         A 最大值 : 31
```

1.2.2 元组

元组（tuple）和列表非常类似，也是 Python 中最常用的一种序列。但是，元组一旦被初始化就不能被修改。可以使用小括号或者 tuple 函数创建元组。在使用小括号创建元组时，只有一个元素的元组在定义时必须在第一个元素后面加一个逗号，例如：

```
In[16]:# 初始化一个元组
        C = (1,2,3,4,5,6)
        C
Out[16]: (1, 2, 3, 4, 5, 6)
In[17]:# 定义只有一个元素的元组
        C1 = (1,)
        C1
Out[17]: (1,)
```

和列表一样，针对元组中的元素，同样可以使用索引获取元素，通过 len 函数计算元组的长度，例如：

```
In[18]:# 通过索引获取元组中的元素
        print(C[1])
        print(C[-1])
        print(C[1:5])
Out[18]:2
         6
         (2, 3, 4, 5)
In[19]:# 计算元组的长度
        len(C)
Out[19]:6
```

可以使用加号“+”将多个元组进行组合。例如，拼接元组 C 和 ("A","B","C")，可获得新元组 D。可以使用乘号“*”获取重复的元组。例如，下面的程序将元组 (1,2,"A","B") 重复两次，可使用 (1,2,"A","B") * 2。

```
In[20]:# 将元组进行组合获得新的元组
       D = C + ("A","B","C")
       D
Out[20]: (1, 2, 3, 4, 5, 6, 'A', 'B', 'C')
In[21]:# 获取重复的元组
       (1,2,"A","B") * 2
Out[21]: (1, 2, 'A', 'B', 1, 2, 'A', 'B')
```

1.2.3 字典

字典是 Python 的最重要数据类型之一。其中，字典的每个元素的键值对（key：value）使用冒号“:”分割；键值对之间用逗号“,”分割；整个字典包括在大括号“{}”中。计算字典中键值对的数量可以使用 len 函数。例如，可使用下面的程序初始化一个字典 D。

```
In[22]:#  初始化一个字典
       D = {"A":1, "B":2,"C":3,"D":4,"E":5}
       D
Out[22]:{'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5}
In[23]:# 计算字典中元素的数量
       len(D)
Out[23]:5
```

可以通过字典的 keys 函数查看字典的键，通过字典的 values 函数查看字典的值，并且可以通过字典的键获取对应的值。除此之外，还可以使用字典的 get 函数获取字典中的内容，如果没有对应的元素则输出 None。例如：

```
In[24]:# 查看字典中的键
       D.keys()
Out[24]:dict_keys(['A', 'B', 'C', 'D', 'E'])
In[25]:# 查看字典中的值
       D.values()
Out[25]:dict_values([1, 2, 3, 4, 5])
In[26]:# 通过字典中的键获取对应的值
       print('D["B"]:',D["B"])
       print('D["D"]:',D["D"])
Out[26]:D["B"]: 2
        D["D"]: 4
In[27]:# 通过 get 函数获取字典中的内容，如果没有对应元素则输出 None
       print('D.get("C"):',D.get("C"))
       print('D.get("F"):',D.get("F"))
Out[27]:D.get("C"): 3
        D.get("F"): None
```

字典的 pop 函数可以利用键名删除键值对。针对字典中的键值对，也可以将相应键赋予新的值。例如：

```
In[28]:# 删除对应的键值对
       D.pop("A")
Out[28]:D
```

```
        {'B': 2, 'C': 3, 'D': 4, 'E': 5}
In[29]:# 更新字典中的取值
       D["B"] = 10
       D
Out[29]:{'B': 10, 'C': 3, 'D': 4, 'E': 5}
In[30]:# 往字典中添加新的内容
       D["F"] = 11
       D
Out[30]:{'B': 10, 'C': 3, 'D': 4, 'E': 5, 'F': 11}
```

1.2.4 集合

集合（set）是一个无序的无重复元素的序列。可以使用大括号“{ }”或者 set 函数创建集合。需要注意的是，创建一个空集合必须用 set 函数而不是大括号“{ }”，因为大括号“{ }”是用来创建一个空字典的。例如：

```
In[31]:# 创建一个集合
       A = {"A","B","C",4,5,6}
       A
Out[31]:{4, 5, 6, 'A', 'B', 'C'}
In[32]:# 判断元素是否在集合内
       "A" in A
Out[32]:True
In[33]:# 计算集合元素的数量
       len(A)
Out[33]:6
```

集合之间也可以进行相互的运算。例如，集合的差集可以使用减号“-”或者 difference 函数；集合的并集可以使用“|”或者 union 函数；集合的交集可以使用“&”或者 intersection 函数；集合的并集减去交集可以使用“^”或者 symmetric_difference 函数。例如：

```
In[34]:# 集合之间的运算
       A = {"A","B","C",4,5,6}
       B = {"A","B","D","E",1,2,4,5}
       print("A - B:",A - B)  # 存在集合 A 中但不存在集合 B 中的元素
       print("A - B:",A.difference(B))  # 存在集合 A 中但不存在集合 B 中的元素
       print("A | B:",A | B)  # 集合的并集
       print("A | B:",A.union(B))  # 集合的并集
       print("A & B:",A & B)  # 集合的交集
       print("A & B:",A.intersection(B))  # 集合的交集
       print("A ^ B:",A ^ B)  # AB 集合不同时存在的元素
       print("A ^ B:",A.symmetric_difference(B))  # AB 集合不同时存在的元素
Out[34]:A - B: {6, 'C'}
        A - B: {6, 'C'}
        A | B: {1, 2, 4, 5, 6, 'C', 'B', 'A', 'D', 'E'}
        A | B: {1, 2, 4, 5, 6, 'C', 'B', 'A', 'D', 'E'}
        A & B: {'B', 4, 5, 'A'}
```

```
A & B: {'B', 4, 5, 'A'}
A ^ B: {1, 2, 6, 'C', 'D', 'E'}
A ^ B: {1, 2, 6, 'C', 'D', 'E'}
```

1.2.5 字符串

字符串是 Python 的最常用数据类型。可以使用引号“'”或“"”来创建字符串。字符串的基础使用方式和列表很相似。例如，可以通过索引进行字符串内容的提取，通过 len 函数计算字符串的长度，通过“+”拼接字符串，通过“*”进行字符串的重复输出等。

```
In[35]:# 创建一个字符串 A
       A = "Hello World!"
       print(A)
       print(" 字符串的长度 :",len(A))
Out[35]:Hello World!
         字符串的长度 : 12
In[36]:# 通过索引获取字符串内容
       print("A[2]:",A[2])
       print("A[1:10]",A[1:10])
Out[36]:A[2]: l
         A[1:10] ello Worl
In[37]:# 字符串的拼接
       A + A + "Hello World!"
Out[37]:'Hello World!Hello World!Hello World!'
In[38]:# 字符串的重复输出
       "Hello World!" * 4
Out[38]:'Hello World!Hello World!Hello World!Hello World!'
```

除了可以对字符串进行上述基本操作之外，还可以通过 find 函数查找字符串中的子串，通过 join 函数拼接字符串，通过 split 函数拆分字符串，通过 replace 函数替换字符串中的指定内容，通过 strip 函数删除字符串首尾的空格等。

```
In[39]:# 查找字符串中的子串，找到了就返回第一个字符的索引
       A = "Python 数据分析与可视化实战 "
       print("A.find() : ",A.find(" 分析 "))
Out[39]:A.find() :  8
In[40]:# 拼接字符串
       print("+".join("ABCDE"))
       print("+".join(["A","B","C","D","E"]))
Out[40]:A+B+C+D+E
         A+B+C+D+E
In[41]:# 拆分字符串
       print("Python 数据分析与可视化实战 ".split(" 与 "))
       print("A+B+C+D+E".split("+"))
Out[41]: ['Python 数据分析 ', ' 可视化实战 ']
          ['A', 'B', 'C', 'D', 'E']
In[42]:# 替换字符串中的指定内容
       print("Python 数据分析与可视化实战 ".replace(" 与 ","+"))
       print("A+B+C+D+E".replace("+","-->"))
```

```
Out[42]:Python 数据分析 + 可视化实战
         A-->B-->C-->D-->E
In[43]:# 删除字符串首尾的空格
       print("   Python 数据分析    可视化实战   ".strip())
Out[43]:Python 数据分析    可视化实战
```

1.3 Python 的语法结构

Python 的重要且常用的语法结构，主要有条件判断语句、循环语句等。本节将会对相关的常用内容进行简单介绍，帮助读者快速了解 Python 的语法结构。

1.3.1 条件判断语句

条件判断语句是通过条件语句的执行结果(True 或者 False)来决定是否执行代码块。常用的条件判断语句是 if 语句。例如，判断一个数字 A 是否为偶数，可以使用下面的程序。

```
In[44]:# if 语句
       A = 10
       if A % 2 == 0:                              # 判断是否为偶数
           print("A 是偶数 ")
Out[44]:A 是偶数
```

针对 if / else 语句，其常用的结构如下：

```
if 判断条件:
    执行代码块 1……
else :
    执行代码块 2……
```

如果满足判断条件，则执行代码块 1，否则执行代码块 2。上面的程序判断 A 是偶数，则输出“A 是偶数”。下面的程序判断 A 不是偶数，则输出“A 是奇数”。

```
In[45]:# if  else 语句
       A = 9
       if A % 2 == 0:
           print("A 是偶数 ")
       else:
           print("A 是奇数 ")
Out[45]:A 是奇数
```

In[45] 程序片段的判断结果是二选一的。有时候，判断结果有多种形式。例如，判断学生成绩可以是及格和不及格，也可以是优、良、中。这种多条件判断可以通过增加多个 elif 语句实现。例如，下面的程序判断一个数能否同时被 2 和 3 整除。

```
In[46]:# 多条件判断增加 elif 语句
       A = 1
```

```
        if A % 2 == 0:                          # 能否被 2 整除
            print("A 能被 2 整除 ")
        elif A % 3 == 0 :                       # 能否被 3 整除
            print("A 能被 3 整除 ")
        else:                                   # 其他情况
            print("A 不能被 2、3 整除 ")
Out[46]:A 不能被 2、3 整除
```

多条件判断语句除了使用 if 和 else 外，一般还使用 elif。if 和 elif 语句所列条件之外的情况放到 else 下输出。

1.3.2 循环语句

下面分别介绍利用 for 循环语句和 while 循环语句的示例。for 循环语句主要是遍历一个序列，即将一个序列（如列表、元组等）中的所有元素逐个地取出，每取出一个元素执行一次 for 循环体内的代码块。while 循环语句则是对给定的条件进行判断，当判断结果为真时则执行 while 循环体内的代码块，执行完成后再次对给定的条件进行判断，直到判断结果为假才退出循环。

例如，下面的程序使用 for 循环语句依次从 1 ~ 100 中取出每一个数进行相加，计算 1 ~ 100 的累加和。

```
In[47]:# 通过 for 循环语句计算 1 ~ 100 的累加和
       A = range(1,101)                         # 生成 1 ~ 100 的向量
       Asum = 0
       for i in A:
           Asum += i                            # 等价于“Asum = Asum + i”
       Asum
Out[47]:5050
```

针对计算 1 ~ 100 的累加和，还可以使用 while 循环语句来完成。例如，下面的程序从 100 开始逆向加到 1。

```
In[48]:# 使用 while 循环语句计算 1 ~ 100 的累加和
       A = 100
       Asum = 0
       while A > 0:
           Asum = Asum + A
           A = A - 1
       Asum
Out[48]:5050
```

在循环语句中，还可以通过 break 语句提前跳出循环。例如，下面的程序使用了条件判断语句，如果累加和大于 2000，则通过 break 语句跳出当前的 while 循环语句。

```
In[49]:# 通过 break 语句跳出循环
       A = 100
       Asum = 0
       while A > 0:
           Asum = Asum + A
```

```
            if Asum > 2000:# 如果累加和大于 2000，则执行下面的 break 语句跳出循环
                break
            A = A - 1
        print("Asum:",Asum)
        print("A:",A)
Out[49]:Asum: 2047
         A: 78
```

在 Python 中，还可以在列表中使用循环和判断等语句，这样的列表称为列表表达式。例如，下面的程序生成列表 A 后，通过 for 循环语句将列表 A 中元素使用 int 函数转化为整型，并作为新列表中的元素。

```
In[50]:# 通过列表表达式生成新列表
       A = [15,"2",31,"10",12,"9",2]
       A = [int(i) for i in A]
       A
Out[50]: [15, 2, 31, 10, 12, 9, 2]
```

1.3.3 try/except 语句

“异常”是一个事件，会在程序执行过程中发生，影响了程序的正常执行。一般情况下，Python 在无法正常处理程序时就会发生一个“异常”。捕获“异常”信息可以使用 try/except 语句。try/except 语句用来检测 try 语句中的错误，从而让 except 语句捕获“异常”信息并处理。下面的程序则是通过 for 循环计算一个列表中元素的和，由于列表中有些元素是字符串，无法被直接相加，即直接相加会出错。这里针对这个出错（“异常”）利用 try/except 语句进行捕获并处理。

```
In[51]:# 计算列表 A 中元素的和时，因为数值与字符串不能相加，所以会出错
       A = [0,1,2,3,4,5,6,7,"8","9",10]
       Asum = 0
       for ii in A:
           Asum = Asum + ii
```

执行 In[51] 程序片段会报错，并且会终止程序的执行。

In[52] 程序片段捕获并处理“异常”后继续求和。

```
In[52]:# 计算 A 中数值的和
      A = [0,1,2,3,4,5,6,7,"8","9",10]
      Asum = 0

      for ii in A:
          try:
              Asum = Asum + ii   # 将有可能出现“异常”信息的代码放在 try 下执行
          except:                # 如果发生“异常”，捕获“异常”信息的代码并执行其下
                                   的代码块
              Asum = Asum + int(ii)      # 将列表 A 中每个元素转化为整型再相加
          else:                  # 如果不发生“异常”，则执行“异常”信息的代码下的代
                                   码块
```

```
            print(" 没有出错！ ")
    print(Asum)
    Out[52]:
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    没有出错!
    55
```

关于 try/except/else 语句说明如下。

（1）try：其下的代码块检验是否有“异常”存在。

（2）except：其下的代码块在出现“异常”时执行。

（3）else：其下的代码块是在不出现“异常”时才执行。

1.4 Python 函数

在编程过程中经常会使用到函数。函数是已经组织好的、可重复使用的、实现单一功能的代码段。函数能提高应用程序的可读性，增强代码的重复利用率。Python 提供了许多内建函数，如 print、len 等函数。本节将简单介绍如何自定义函数以及 lambad 函数的使用。

1.4.1 函数

Python 可以自己定义新的函数。自定义函数的结构如下。

```
def functionname( parameters ):
   """
    函数 _ 文档字符串，对函数进行功能说明
   """
   function_suite                  # 函数体，即函数要实现的功能代码
   return expression               # 函数的输出
```

其中，functionname 表示函数的名称；parameters 指定函数需要传入的参数。自定义一个计算累加和的函数，程序如下。

```
In[52]:# 定义一个计算累加和的函数
       def sumx(x):
           """
            该函数对 1~x 的所有数据求和
            x 表示终止数字，如果 x 为 5，则表示该函数求 1+2+3+4+5 的和
```

```
        """
        x = range(1,x+1)      # 生成 1 ~ x 的向量
        xsum = 0
        for i in x:
            xsum = xsum + i
        return xsum

    # 调用上面的函数
    sumx(200)

Out[52]:20100
```

上面定义的函数中 sumx 是函数名，x 是使用函数时需要输入的参数，调用函数可使用 sumx(x) 来完成。

1.4.2 lambda 函数

lambda 函数也叫匿名函数，即没有具体名称的函数，它可以快速定义单行函数，完成一些简单的计算功能。可以使用下面的程序定义 lambda 函数。

```
In[53]:# 一个参数的 lambda 函数
       f = lambda x: x**2        # 参数为 x，函数的功能为 x**2，即计算 x 的平方
       f(5)
Out[53]:25
In[54]:# 多个参数的 lambda 函数
       f = lambda x,y,z: (x+y)*z # 参数为 x、y、z，函数的功能为 (x+y)*z
       f(5,6,7)
Out[54]:77
```

在 lambda 函数（表达式）中，冒号前面可以有多个参数，参数之间要用逗号分隔；冒号后面是函数的功能以及返回的计算结果。

1.5 数据可视化分析

数据可视化是关于数据视觉表现形式的科学技术研究，它旨在借助图形化手段，清晰有效地传达与沟通信息，是科学可视化与信息可视化的统一。当前，数据可视化在教学、科学研究等方面极为活跃，已成为人工智能和大数据分析的基础内容之一。俗话说“一图胜千言”，相对于复杂难懂且体量庞大的数据，图表所传达的信息量要大得多，并且更有效。

1.5.1 什么是好的数据可视化

数据可视化与信息可视化、科学可视化以及统计图形密切相关。好的数据可视化图像并不是看上去绚丽多彩而极端复杂的，而是能有效地传达数据信息、思想概念的，其美学

形式与功能等需要统筹考虑。

数据可视化的目的是通过对数据进行可视化处理，从而以更简单、精确、有效的方式传递信息。相对于枯燥乏味的数值、复杂的数据结构和类型，人们对图像的形状、位置、大小、色彩等信息能够更好、更快地识别。因此，通过数据可视化得到的图像能够加深对数据的认识、理解与记忆，使信息更容易准确地传播。数据可视化的作用是表达观点，通过设计合适的可视化图像，使得没有背景知识的大众也能读懂图像中隐藏的重要信息。

好的数据可视化案例一般会具有以下特点。

（1）快速传递有用的信息。数据可视化的最主要目的是帮助我们快速从数据中读取想要的信息，因此可视化图像对信息传递的快速、准确非常重要。

（2）充分显示数据的多维性。数据可视化图像应兼顾全局，从多个角度和维度刻画数据，避免陷入数据的局部细节。通过数据对每一维度值的分类、排序、组合等，观察数据的多个属性，从而获取更加全面的信息。

（3）直观地展示信息。我们可以通过好的可视化图像获得复杂的信息，甚至只要简单的图像就能将信息展示，从而使决策者轻松地使用数据。

本书主要讲述的是基于 Python 的数据可视化分析与实战，并将数据可视化与分析工具相结合，在充分传达有用信息的同时获得较为简单的可视化图像。因此，本书在介绍数据可视化分析和通过 Python 程序获得可视化图像时，将会尽可能地使用简单的 Python 程序获取相同的可视化效果。

1.5.2 数据可视化图像的基本类型

数据图像可以分为几大类，如趋势型图像、类别比较型图像、数据关系型图像、数据分布型图像、关联型图像、高维数据型图像、地理空间型图像等。针对这些类型的数据图像，常用的数据可视化图像如图 1-6 所示。

图 1-6　常用的数据可视化图像

1.5.3 数据可视化分析基本流程

在利用 Python 进行数据可视化分析时，可以参考图 1-7 所示的数据可视化分析基本流程示意图。在图 1-7 中，将数据可视化分析分为了 5 个步骤，分别是确定数据可视化主题、收集数据可视化需要的数据，对数据进行预处理和特征变换，确定展示数据的数据可视化图像及利用 Python 中合适的库进行数据可视化。在实际的数据可视化分析时，可以灵活调整相应的步骤。

图 1-7　数据可视化分析的基本流程示意图

1.5.4 Python 进行数据可视化分析的优势

相比于其他常见的统计分析与绘图软件，Python 在数据分析、图形绘制、数据挖掘、机器学习等数据可视化方面具有诸多优势。

（1）Python 有大量的数据可视化库。Python 在数据分析与数据可视化领域有很多优秀的第三方库可供使用。例如，针对静态图像的数据可视化，可以使用 Pandas、Matplotlib、Seaborn、plotnine 等库，快速方便地得到自己需要的数据可视化图像，进行信息的快速传递；针对可交互图像的数据可视化，可以使用 Plotly、Bokeh、Pyecharts 等库；针对网络图像的数据可视化可以使用 Networkx、igraph 库等。同时，Python 还拥有针对其他类型图像的数据可视化库，如针对地图的数据可视化库、针对三维图像进行渲染的数据可视化库等。

（2）Python 除了具有强大的数据可视化功能外，还具有数据预处理、数据分析、数据挖掘及机器学习的能力，从而可以在大数据分析的过程中，将这些功能和数据可视化相结合，帮助使用者快速分析数据、解读结果及验证模型效果等，从而提升数据分析和信息获取的效率。

（3）Python 是开源的编程语言，其使用是完全免费的；拥有数量庞大的志愿者，对

使用者提出的各种问题进行答疑解惑；具有大量功能丰富的库可以使用；简明易懂，利于初学者学习掌握；可以直接通过 Python 来绘制论文所需要的图像。

1.6 本章小结

本章主要介绍了 Python 的入门内容，并且对数据可视化分析进行了简单的介绍。主要介绍了相关环境的安装和使用，以及 Python 的列表、元组和字典等基础的数据结构，同时还介绍了 Python 中的条件判断、循环与函数等基础的语法结构。针对数据可视化分析，不同类型的可视化图像均有其所使用的数据场景。针对这些图像的数据可视化方式及使用，会在后面更详细的数据可视化实战案例中进行讲解。