第 1 章

数据分析基础

学习目标

(1)知道数据分析的定义,并了解数据分析范式。
(2)了解数据分析所面临的问题并能够思考相应的解决方案。
(3)理解数据分析的相关概念。
(4)了解数据分析的任务和方法。
(5)熟悉不同领域的数据类型与常见数据类型。
(6)熟悉数据分析的生态系统。

学习重点

(1)数据分析的定义。
(2)数据分析范式。
(3)数据分析的任务与方法。
(4)数据分析的生态系统。

随着农业社会和工业社会的衰落,人类社会从 20 世纪中期开始向以计算机为代表的信息社会迅速迈进。在信息社会中,短短数十年间,人们走过了计算机时代和互联网时代,迎来了以海量数据为主要特征的大数据时代。当前,由于信息化系统的迅速普及,以及各式各样数据采集设备的广泛应用,人类的生产生活、行为足迹都在被数字化。同时,数据存储技术的快速进步使得各个组织机构可以轻易实现数据积淀。从而,大数据时代形成了与物理空间、社会空间相对应的第三维空间,即数据空间,其中现实个体能够以数据的形式被充分地刻划和定义。

海量数据捕获了现实世界中个体的静态属性和动态行为特征,使得人们通过数据能够了解、洞悉现实个体的特性与状态,进而极大地影响经济社会发展。因此,数据成为土地、能源之外的新型战略性资源。面对近年来迅猛增长的数据,如何进行有效的数据分析、从海量数据资源中提取有价值的信息和知识已经成为巨大的挑战。同时,我国十分重视大数据产业发展,数据分析在各行业的应用已卓有成效,数据分析技术人才作为驱动数据价值落地的关键因素越

来越被广大企业所重视。

1.1 数据分析的基本概念

在日常生活中，无时无刻不在产生数据。早晨起床，手机通过闹钟会知道你的作息时间；打开电视看新闻，电视会记录你的观看记录；查看新消息、朋友圈，手机会保存你的浏览记录；外出健身，摄像头会记录、保存你的影像；上班工作，企业通过信息化系统进行业务管理。所以，人们每天都会产生与生活、工作相关的大量数据。同样，企业也会不断产生各种类型的数据。以上数据通过遍布各地的通信系统和存储系统汇聚起来便成为海量数据。

理解数据分析的基础是了解数据。从概念而言，大数据(big data)是 1998 年由 SGI(美国硅图公司)首席科学家 John Mashey 在 USENIX 说明大会上提出的。他当时发表了一篇名为 *Big Data and the Next Wave of InfraStress* 的论文，使用了大数据来描述数据爆炸的现象。根据百度百科的定义，大数据是指无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察力和流程优化能力的海量、高增长率和多样化的信息资产。在 2015 年《促进大数据发展行动纲要》中，大数据是指以容量大、类型多、存取速度快、应用价值高为主要特征的数据集合，正快速发展为对数量巨大、来源分散、格式多样的数据进行采集、存储和关联分析，从中发现新知识、创造新价值、提升新能力的新一代信息技术和服务业态。IBM 提出大数据具有 5V 特点：volume(大量)、velocity(高速)、variety(多样)、value(低价值密度)、veracity(真实性)。

1.1.1 数据分析的定义

数据分析(data analysis)是指将各种算法应用于大量的、有噪声的、不完全的实际数据中，提取隐含在其中的、人们事先不知道的有用信息和知识的实践过程。各种算法包括来自统计学中的抽样、估计及假设检验，也包括机器学习、模式识别等领域的建模技术和学习理论。通过数据分析，可以发现个体对象的特征性质、类别分布，也可以预测个体对象未来的观测结果。数据分析为科学研究、社会管理、工业生产等提供了一种新的方法，其目的在于揭示自然现象与人类社会背后隐藏的特性和规律。

对实际数据进行探索、提取知识的过程一般包括数据预处理、数据分析、后处理等步骤，如图 1-1 所示。数据预处理包括数据融合、数据去重、数据去噪等过程。由于数据质量参差不齐、数据格式复杂多样，数据预处理一般较为费时耗力。通过数据预处理，原始数据会被转化为适合分析的形式。后处理主要是为了对数据分析的结果进行解释和展示，使其更易于理解和使用。

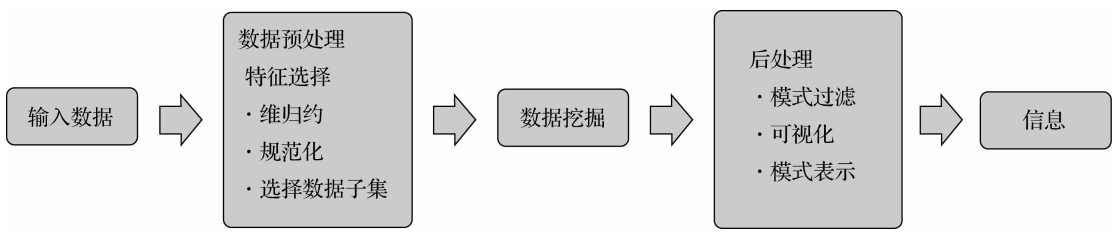

图 1-1 基于数据分析的知识提取过程

数据分析主要涉及三个方面的内容：数据分析理论、数据分析技术和数据分析实践，如图 1-2 所示。其中，数据分析理论是指人们对数据及数据资源利用相关的对象、价值及影响进行思考而产生的认知。通过分析累积的海量数据，能够实现客户群体细分、工作效率提高及业务模式创新等，从而极大地促进经济社会发展。在此过程中，随着数据分析应用的逐渐深入及数据类型的多样化和复杂化，数据分析的实践形态和主要目标也会逐步演化。以上活动需要根据其特点进行抽象提取，从而构建数据分析过程的知识体系。例如，数据分析将来可能不再局限于从数据中提取有价值的信息这一任务，由数据分析引起的数据隐私、数据产权、数据资源共享与管理等将成为新的理论焦点；数据分析技术是指数据分析实施过程依赖的技术方法，包括数据存储、算法模型、分布式平台及云计算等。数据分析技术是落实数据分析理论的保障。根据现实应用的业务场景，需要具备在不同数据分布、分析效率及任务要求条件下进行数据分析的技术方法。例如，除了数据资源、算法模型之外，大规模数据分析实践往往还需要存储环境、计算平台等；数据分析实践是将数据分析技术与累积的海量数据相结合，产生有价值的信息、知识的具体过程。此过程往往与实际数据质量、业务场景等紧密相关，当前的数据分析实践主要可以分为个人数据分析实践、企业数据分析实践、政府数据分析实践和互联网数据分析实践四类。实际上，数据分析理论是数据分析技术和数据分析实践的基础和指导，数据分析技术是数据分析理论在实际业务场景下的实现途径，数据分析实践是基于数据分析理论和数据分析技术体现数据价值的最终过程。

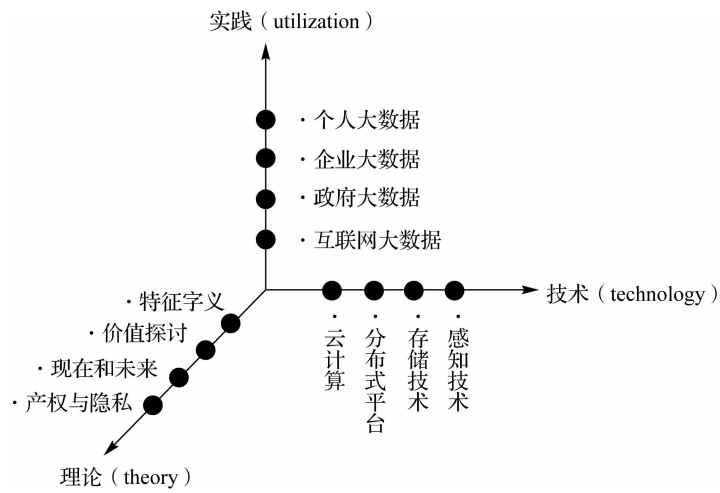

图 1-2　数据分析的内涵

1.1.2　数据分析范式

数据分析的总体目标是在已有数据集的基础上，通过特定的算法提取信息，并将信息转化为可理解的知识，从而辅助决策。任何实际的数据分析过程都是重复这一数据分析的工作范式，如图 1-3 所示。具体来说，数据(data)是使用约定俗成的关键词，对客观事物的数量、属性、位置及其相互关系进行抽象表示，以适合在当前的领域中用人工或自然的方式进行保存、传递和处理。例如，通过高度20 m、宽度 80 m、共 3 层这些数据便描述了客观世界建筑物在人们大脑中的印象；信息(information)来源于数据，但高于数据。信息是指具有时效性的、有一定含义的、有逻辑的、经过加工处理的、对决策有价值的数据流；信息虽给出了数据中一些有一定意义的东西，但它

的价值往往会在时间效用失效后开始衰减，只有通过人们的参与，对信息进行归纳、演绎、比较等处理，使其有价值的部分沉淀下来，并与已存在的人类知识体系相结合，这部分有价值的信息就转变成知识(knowledge)，即知识就是沉淀并与已有人类知识库进行结构化的有价值信息；决策(decision)是指在获取知识的基础上，根据实际业务场景采取有效行动从而提高收益的具体过程。

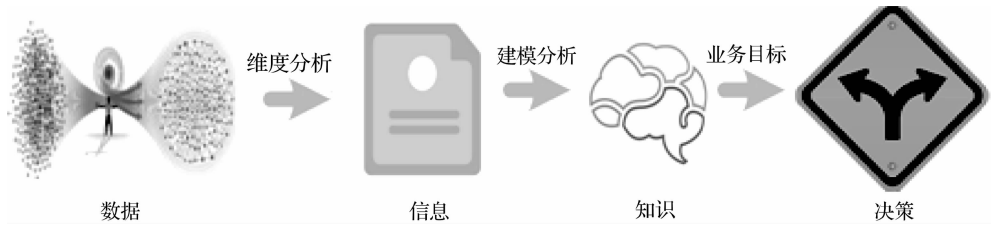

图 1-3　数据分析的工作范式

以商品房销售为例，数据是指销售过程中交易行为的直接记录，包括房屋位置、房屋面积、房屋类型、交易时间、成交价格等；基于以上原始记录，通过对数据进行处理，得出类似“201×年×月北京市朝阳区商品房销售价格同比上涨×%”的结论，即为获得信息；人们通过对以上类似的信息进行归纳、演绎、比较等处理，使其有价值的部分沉淀下来，并与已存在的人类知识体系相结合，这些有价值的信息就转变成知识，即类似“由于土地供应有限，人口迅速增加，北京市商品房交易价格总体上处于并将继续处于上涨态势”；在以上数据分析的基础上，人们基于获得的知识进行最终的投资决策，即“通过购置房产以获得较高的投资回报”。以上即为通过房屋交易记录处理，最终辅助决策的数据分析过程。

1.1.3　数据分析面临的问题

在当前的大数据时代，数据分析面临许多前所未有的困难和挑战。

1. 可伸缩性

由于数据分析任务中实际数据量的巨大差异，要求数据分析的模型算法能够处理大数据集，且时间复杂度不能太高，消耗的内存空间也有限，即必须是可伸缩(scalable)的。为了实现可伸缩的数据分析，研究人员已经进行了许多有益的尝试，包括增量式方法、数据采样方法、数据压缩方法、分布式算法、并行框架等。

2. 高维性

传统的数据集一般只有少量的属性，然而，大数据时代数据集的属性往往是成百上千的。由于数据采集技术的进步，同样的对象可以通过更多传感器从更加丰富的角度进行感知刻划，从而导致数据维度增加。随着数据维度的增加，可能的组合数量也随之上升，样本在属性空间中变得稀疏，从而很可能产生无意义的分析结果。如果持续增加维度，训练数据需要指数级的增加才能避免过拟合。另外，维度增加会导致计算机内存消耗巨大、计算时间延长。而且，在高维空间中，样本距离的度量失去意义，会影响 K-means、KNN(K-nearest neighbor，K 近邻算法)等基于距离计算的方法的精度和性能。

3. 异构及复杂数据

传统数据分析任务中的数据类型主要包括文本、图像、数字、语音信号等，通过聚类、分类等发现其蕴含的模式和语义。随着数量采集技术的发展和数据分析在各个领域的广泛应用，数据类型越来越复杂。例如，由于在线社交网络的兴起，同时包含文本、图片、关系链的多媒体信息对于知识挖掘变得愈发重要。另外，刻划个体及其之间相互关联关系的图数据也成为大数据中的一种重要数据类型。随着定位技术和移动互联网的发展，车辆及行人轨迹数据的量级也在迅猛增长。为了挖掘这些复杂数据对象，需要开发与数据特征相适应的新型数据分析方法。

4. 数据分布

随着数据采集技术的进步，生成的数据量往往十分巨大。所以，构建集中的数据存储空间是极具挑战性的。而且，由于网络数据传输带宽的限制，海量数据传输的耗时也难以接受。例如，当前无处不在的监控摄像头虽然记录了大量视频监控数据，但是实际上是无法进行集中统一分析的，这些数据是在各个区域分散保存的。另外，由于数据所有权的问题，数据分析任务可以拥有数据的使用权，但没有数据的所有权。因此，只能通过远程访问进行数据分析。所以，实际数据常常处于不同地理位置及不同机构中，大数据的资源利用要求开发分布式的数据分析技术。

5. 隐私保护

由于实际数据中经常包含用户隐私信息，直接基于这个数据进行分析挖掘会泄露用户隐私。因此，需要在操作、分析这些数据的同时保护用户隐私。当前，研究人员已开始关注密文可搜索机制、支持隐私保护的数据聚类等研究。例如，医疗机构积累了大量医疗数据，通过对这些数据进行分析，可能优化疾病的治疗效果、提高疾病的治愈率。然而，通过专业机构进行医疗数据分析需要考虑患者的隐私问题，需要使用保障数据安全的数据分析方法。

1.1.4 相关概念辨析

为了分析处理来自各个领域、面向不同需求的数据处理任务，研究人员和工程师提出了很多相互联系但又有所差异的概念和定义。首先，数据必须保存在存储空间中才能够进行数据分析，数据库是保存数据的主要形式。由于数据规模大、存储位置比较分散，为了提高计算效率，数据分析常常依赖于并行计算和分布式计算框架。为了能够让计算机拥有人的能力从而实现人工智能，研究人员提出了许多利用经验数据的学习算法，即机器学习。机器学习的理论基础主要是统计学，为了能够从数据中提取有用的知识，数据分析经常需要采用机器学习方法。从而，数据分析可以被视为机器学习与数据库的交叉，利用机器学习领域提供的技术分析海量数据，利用数据库领域的技术进行海量数据的管理。同时，机器学习一般指实现数据建模学习的相关算法，数据分析主要指从数据中提取有用知识的过程。数据分析除包括相关建模学习算法之外，还包括数据清洗、描述性分析等步骤。或者，机器学习注重相关机器学习算法的理论研究和算法提升，数据挖掘注重运用算法或其他某种模式来解决实际问题。机器学习为数据挖掘提供解决实际问题的方法，数据挖掘中算法的成功应用，说明了机器学习对算法的研究具有实际运用价值。在数据分析和机器学习算法的形成过程中，还吸收了来自其他领域的思想，包括最优化、进化计算、信息论和信号处理。信息检索指从数据集合或数据接口中查找个别记录的过程，主要依赖于数据的

明显特征来创建索引结构，从而有效地组织和检索信息。

数据分析与机器学习的发展不是孤立的，是被广泛应用于各个领域、与各个领域的发展共同成长的。例如，机器学习在视觉领域的应用形成了计算机视觉（computational vision）方向。计算机视觉通过采集图片或视频，对图片或视频进行处理分析，从中获取相对应的信息。换而言之，就是运用照相机和计算机来获取我们所需的信息。另外，机器学习在汉语、英语、法语等自然语言领域的应用形成了自然语言处理（natural language processing）方向。自然语言处理，简单来说就是计算机接收用户自然语言形式的输入，并在内部通过人类所定义的算法进行加工、计算等一系列操作，以模拟人类对自然语言的理解，并返回用户所期望的结果。随着互联网、在线社交网络、生物信息网络等的兴起，世界万物变得愈发相互关联，从而形成了以顶点及顶点间的边为结构的大规模图数据。机器学习在图数据分析领域的应用形成了图计算（graph computing）方向。图计算就是主要针对结点关系型数据，研究在这样的数据上进行高效的检索、分类、聚类、预测等任务。图 1-4 展示了数据分析与其他领域之间的关系。

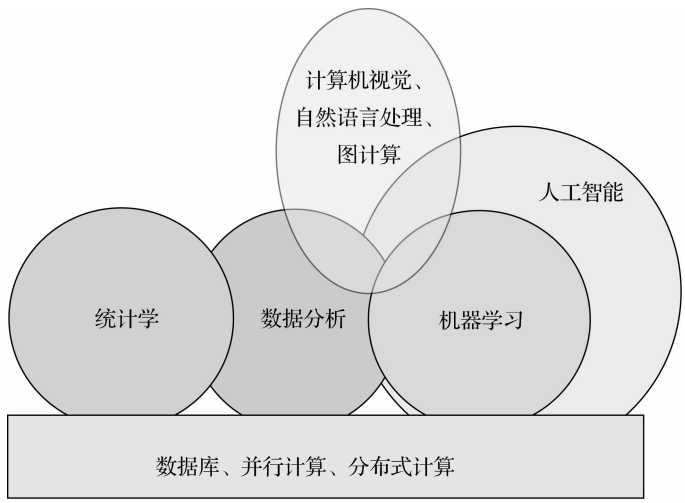

图 1-4　数据分析与其他领域之间的关系

1.2　数据分析的任务与方法

当前现实世界和网络空间都在产生数据，分析挖掘这些数据为人们认识、理解、掌控自己的生产生活环境提供了新的机会和可能。在数据分析挖掘之前，首先需要确定任务目标、理解如何从数据分析结果中受益。然后收集需要的数据，并进行数据清洗、集成和转换等数据准备工作。在进行数据分析建模之前，需要深入地探索理解数据，了解各属性元素之间的相互作用，熟悉数据的分布及特征。最后进行数据分析建模，对数据的分析结果进行展示和应用。所以，数据分析挖掘总体上包括确定目标、数据收集、数据准备、数据探索、数据建模和展示应用六个阶段。

另外，数据分析挖掘过程中数据的建模学习也需要多个步骤，具体包括以下几个方面。

（1）选择数据：将数据分成训练数据、验证数据和测试数据。

（2）模型学习：使用训练数据来构建基于相关特征的模型。

(3)验证模型:使用验证数据直接进行模型评价。

(4)测试模型:使用测试数据检查被验证的模型的性能表现。

(5)调优模型:使用更多数据、不同的特征或不同的参数取值来提升算法的性能表现。

根据数据集中其他属性的取值预测特定属性的取值,或者根据历史样本数据预测当前及未来样本的取值的过程即为预测(prediction)建模。预测建模的目标是训练一个模型,使目标变量预测值与实际值之间的误差达到最小。实际上,用于预测离散目标变量的分类(classification)任务,以及用于预测连续目标变量的回归(regression)任务都属于预测建模。面对各种各样的数据分析需求,当前主要的数据分析建模算法按学习结果可分为分类、回归、聚类和降维四个任务。

1.2.1 分类任务

给定一个样本特征,我们希望预测其对应的属性值。如果此属性值是离散的,那么这就是一个分类问题。面对离散属性预测的分类(classification)任务,当前已经有包括朴素贝叶斯(naive bayes)、支持向量机(support vector machine,SVM)、K 近邻算法(KNN)、人工神经网络(artificial neural network,ANN)、决策树(decision tree)等在内的多种分类算法被提出。

1. 朴素贝叶斯

朴素贝叶斯是基于贝叶斯定理与特征条件独立假设的分类方法,发源于古典数据理论,拥有稳定的数学基础和分类效率。它是一种十分简单的分类算法,通过对给出的待分类项求解各项类别的出现概率大小,来判断此待分类项属于哪个类别。贝叶斯适用于特征之间的相互独立的场景,如利用花瓣的长度和宽度来预测花的类型。“朴素”的内涵可以理解为特征和特征之间独立性强。朴素贝叶斯在文本分类问题中会有很好的效果,常被用于垃圾邮件过滤、医疗诊断中。

2. 支持向量机

支持向量机是一种监督式学习方法,可广泛应用于统计分类及回归分析。支持向量机属于一般化线性分类器,它能够同时最小化经验误差和最大化几何边缘,因此,支持向量机也被称为最大边缘分类器。

3. K 近邻算法

KNN 是一个比较简单的分类、预测算法。首先选取与待分类数据最相似的 K 个训练数据,然后通过对这 K 个数据的结果或分类标号进行取均值、取众数等方法得到待分类、待预测数据的结果或分类标号。

4. 人工神经网络

人工神经网络是一种模仿生物神经网络结构和功能的数学模型或计算模型,用于对函数进行估计或近似。人工神经网络通过大量的人工神经元联结进行计算,通过在外界信息的基础上修正内部结构和连接权值实现学习过程,是一种自适应系统。人工神经网络在语音、图片、视频、游戏等各类应用场景展现出优异的性能,但是存在需要大量的数据进行训练来提高准确性的问题。

5. 决策树

决策树是一个树结构，其中每个非叶结点表示一个特征属性上的判断，每个分支代表这个特征属性在某个值域上的输出，而每个叶结点存放一个类别。使用决策树进行决策的过程就是从根结点开始，判断待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子结点，将叶子结点对应的类别作为决策结果。

1.2.2 回归任务

给定一个样本特征，我们希望预测其对应的属性值。如果此属性值是连续的，那么这就是一个回归问题。回归任务预测目标值最直接的办法是依据输入写出一个目标值的计算公式，该公式就是所谓的回归方程（regression equation），求回归方程中的回归系数的过程就是回归，如图 1-5 所示。与分类任务类似，对于进行连续属性预测的回归（regression）任务，当前已经有包括线性回归（linear regression）、逻辑回归（logistic regression）、岭回归（ridge regression）等在内的多种回归模型被提出。

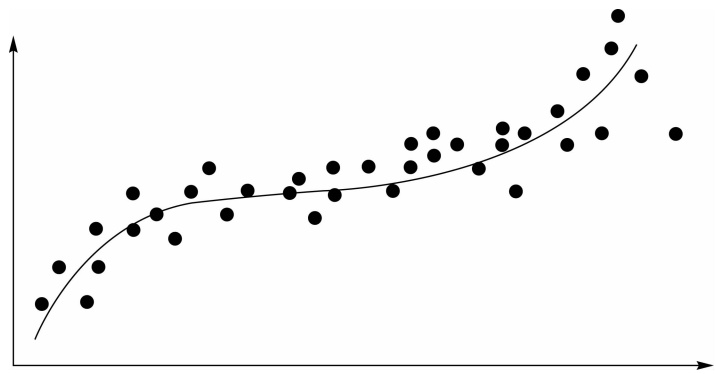

图 1-5 回归算法示意图

1. 线性回归

线性回归是利用数理统计中的回归分析来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。其中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示的回归分析称为一元线性回归分析。如果包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。

2. 逻辑回归

逻辑回归在线性回归的基础上通过使用其固有的逻辑函数估计概率，来衡量因变量（想要预测的标签）与一个或多个自变量（特征）之间的关系。通常，逻辑回归会将概率值压缩到某一特定范围。逻辑回归一般用于需要明确输出的场景，如预测是否会发生降雨。

3. 岭回归

为了应对线性回归模型中基于最小二乘法的系数求解方法对输入变量噪声敏感及解不稳定问题，对原目标函数添加惩罚项以限制模型参数的数值大小。

1.2.3 聚类任务

聚类(clustering)任务是在没有给定划分类别的情况下,根据样本相似度进行样本分组的一种方法。聚类的输入是针对一组未被标记的样本,聚类算法根据数据自身的距离或相似度将其划分为若干组,划分的原则是组内距离最小化而组间距离最大化,如图 1-6 所示。聚类和分类都是分析样本的属性,不同的是,分类在预测之前知道属性的范围,或者说知道到底有几个类别,而聚类是不知道属性的范围的。所以,分类也常常被称为监督学习(supervised learning),而聚类就被称为无监督学习(unsupervised learning)。对于聚类任务,当前已有 K-means 聚类、DBSCAN 聚类、谱聚类(spectral clustering)等方法被提出。

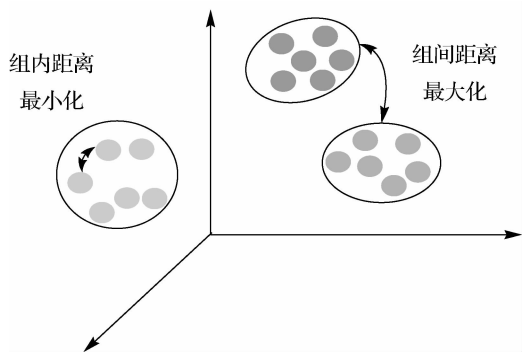

图 1-6 聚类算法示意图

1. K-means 聚类

K-means 聚类是一种无监督学习算法,此算法把 N 个点(可以是样本的一次观察或一个实例)划分到 K 个集群(cluster),使得每个点都属于离它最近的均值(聚类中心,centroid)对应的集群。

2. DBSCAN 聚类

具有噪声的基于密度的聚类方法(density based spatial clustering of applications with noise,DBSCAN)是一种基于密度的空间聚类算法。该算法将具有足够密度的区域划分为簇,并在具有噪声的空间数据库中发现任意形状的簇。

3. 谱聚类

谱聚类是从图论中演化出来的算法,后来在聚类中得到了广泛的应用。它的主要思想是把所有的数据看作空间中的点,这些点之间可以用边连接起来。距离较远的两个点之间的边权重值较低,而距离较近的两个点之间的边权重值较高,通过对所有数据点组成的图进行切图,让切图后不同的子图间边权重和尽可能低,而子图内的边权重和尽可能高,从而达到聚类的目的。

1.2.4 降维任务

降维(dimensional reduction)任务的目标就是采用某种映射方法,将原高维空间中的数据点映射到低维空间中。之所以进行数据降维,是因为在原始的高维空间中包含冗余信息及噪声信息,在实际应用中会造成误差,降低准确率;希望通过降维减少冗余信息所造成的误差,提高识别

精度，后者通过降维算法来寻找数据内部的本质结构特征。对于降维任务，当前已有主成分分析（principal component analysis，PCA）、线性判别分析（linear discriminant analysis，LDA）、多维尺度变换（multiple dimensional scaling，MDS）等方法被提出。

1. 主成分分析

PCA 降维是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大，以此使用较少的数据维度，同时保留住较多的原数据点的特性。也就是说，PCA 追求的是在降维之后能够最大化保持数据的内在信息，并通过衡量在投影方向上的数据方差的大小来衡量该方向的重要性。

2. 线性判别分析

LDA 方法也称 fisher linear discriminant，是一种有监督的线性降维算法。它通过将高维空间数据投影到低维空间来确定每个样本所属的类，这里考虑 K 个类的情况。它的目标是将样本能尽可能正确地分成 K 类，体现为同类样本投影点尽可能近，不同类样本点尽可能远。

3. 多维尺度变换

MDS 算法是一种经典的降维方法，可以缓解在高维情形下出现的数据样本稀疏和距离计算困难（维数灾难）等问题。MDS 算法寻找高维数据到低维空间的投影，要求原始空间中样本之间的距离在低维投影空间中得以保持。

1.3 典型数据领域与常见数据类型

几乎所有的商业和非商业机构都在产生数据，这些数据能够帮助企业、机构为用户提供更好的用户体验和产品服务。

1.3.1 典型数据领域

典型的数据领域包括以下几个方面。

1. 医疗大数据

医疗大数据主要包括患者就医过程产生的数据，如患者的电子病历、体征数据、化验数据、住院数据，医学影像数据，医生对患者的问诊数据，医生对患者的临床诊治、用药、手术等数据；制药企业和生命科学产生的数据，如与用药相关的用药量、用药时间、用药成分、实验对象反应时间、症状改善表象等数据；可穿戴设备带来的健康数据，如通过各种穿戴设备（手环、起搏器、眼镜等）收集的人体的各种体征数据；医疗研究和实验室数据，如在药物研制过程中进行各种实验及不同适应症状、副作用等数据。通过对以上数据的分析应用，可以为患者提供更好的、个性化的医疗方案，可以提供良好的疾病管理服务，可以设计更高效的药物研发方案。

2. 金融大数据

金融大数据主要包括金融机构在日常经营中各种应用系统产生的数据，如银行卡数据、存贷款数据、客户资产数据、个人信用数据等；金融机构在网上采集的企业舆情数据和个人行为数据，如企业相关的司法、行政处罚及诉讼数据，资产重组、投融资及产品数据，个人学历、职业数据等；

从第三方购买的数据，如从政府购买的公积金、社保和税务等公共数据。通过对以上数据的分析应用，可以进行贷款风险评估和交易欺诈检测，可以进行理财推荐和精准营销，可以进行股价预测，可以基于客户细分开发符合客户个性化需求的金融产品。

3. 交通大数据

交通大数据包括基于公共交通部门发行交通卡收集的乘客出行数据，如各时段、各路段的乘车人数；交通管理部门在道路上预埋或预设物联网传感器收集的数据，如车流量、客流量信息；通过卫星地图对城市道路的交通情况进行分析得到道路交通的实时数据；通过出租车车载终端或数据采集系统提供的实时数据；基于智能手机中的地图应用产生的交通信息。通过对以上数据的分析应用，可以感知城市道路拥堵情况，可以制定疏散和管制措施预案，从而提高道路管理能力，可以为用户提供线路推荐服务。

4. 旅游大数据

旅游大数据包括城市的旅游景点数据、经济数据、气象数据、人文数据等，景区的票务数据、旅游人数数据、监控数据等，酒店的在线预订和入住数据、客户信息及行为偏好数据等，旅游网站的旅游线路数据、旅游攻略数据、点评数据等，游客画像、消费数据、紧急调度数据等。通过对以上数据的分析应用，可以进行旅游推荐、安全提醒、旅游产品规划、旅游广告投放等服务。

5. 在线网络大数据

在线网络大数据包括社交平台产生的社会关系数据、用户生成数据、位置数据、个人偏好数据等，电商平台产生的商品销售数据、客户消费数据、促销活动数据等，视频网站产生的视频内容数据、用户观看数据等，媒体平台产生的新闻数据、用户评论数据等，求职招聘平台产生的企业招聘数据、个人简介数据、岗位薪酬数据等。通过对以上数据的分析应用，可以进行社会关系推荐、工作职位推荐、商业广告投放、新闻消息推荐等服务。

1.3.2 常见数据类型

以上来自各个领域的数据具有不同的类型，从数据类型上大致可以分为结构化数据、半结构化数据和非结构化数据三大类。

1. 结构化数据

结构化数据是能够用二维表结构来表达实现的数据，如数字、符号等。在项目中，这些数据一般保存在数据库中，其具有的明确关系使得这些数据运用起来十分方便，不过，它们在商业上的可挖掘价值比较差。

2. 半结构化数据

半结构化数据不符合关系型数据库或其他数据模型结构，但包含用来分隔语义元素及对记录和字段进行分层的相关标记。因此，它也被称为自描述的结构。在半结构化数据中，属性的个数、顺序等都是不重要的，它们可以通过相关标记进行解释，从而具有良好的扩展性。常见的半结构化数据有网页、XML 和 JSON。

3. 非结构化数据

非结构化数据是数据结构不规则或不完整，没有预定义的数据模型，不方便用数据库二维逻

辑表来表现的数据。非结构化数据包括所有格式的办公文档、文本、图片、各类报表、图像和音频、视频信息等。非结构化数据的格式非常多样，标准也是多样性的，而且在技术上非结构化信息比结构化信息更难标准化。所以，存储、检索、发布及利用非结构化数据需要更加智能化的 IT 技术，如海量存储、智能检索、知识挖掘、内容保护等。

1.4 数据分析的生态系统

目前，数据分析方面的工具和框架已经有很多，极大地方便了数据的分析和挖掘工作，本节将按功能和目标介绍数据分析的相关工具，这将有助于读者了解数据分析的生态系统。

1.4.1 分布式文件系统

1. 谷歌文件系统(Google file system，GFS)

GFS 是 Google 为了满足本公司需求而开发的基于 Linux 的专有分布式文件系统。尽管 Google 公布了该系统的一些技术细节，但 Google 并没有将该系统的软件部分作为开源软件发布。

2. Hadoop 分布式文件系统(Hadoop distributed file system，HDFS)

HDFS 是 Hadoop 的核心子项目，是一个可以运行在普通硬件设备上的分布式文件系统，是分布式计算中数据存储和管理的基础，是基于流数据模式访问和处理超大文件的需求而开发的。它所具有的高容错、高可靠性、高可扩展性、高吞吐率等特征为海量数据提供了不怕故障的存储，给超大数据集的应用处理带来了很多便利。

3. Ceph 分布式文件系统

Ceph 是一个统一的分布式存储系统，设计初衷是提供较好的性能、可靠性和可扩展性。Ceph 项目最早起源于 Sage Weil 就读博士期间的工作(最早的成果于 2004 年发表)，并随后贡献给开源社区。在经过数年的发展之后，目前已得到众多云计算厂商的支持并被广泛应用。Red Hat 及 OpenStack 都可与 Ceph 整合以支持虚拟机镜像的后端存储。

4. Lustre 分布式文件系统

Lustre 是一款开源的、基于对象存储的集群并行分布式文件系统，具有很高的扩展性、可用性、易用性等，在高性能计算中应用很广泛，世界十大超级计算中心中的 7 个及超过 50%的全球排名前 50 的超级计算机都在使用 Lustre。Lustre 可以支持上万个结点，数以 PB 数量的存储系统。

1.4.2 分布式编程框架

1. MapReduce 编程框架

MapReduce 是一种编程模型，被广泛应用于大规模数据的处理中。2004 年，Google 发表了一篇论文，介绍了 MapReduce 编程框架。利用这款软件框架，开发人员可以快速地编写分布式应用程序。现今，该框架已被广泛地应用到日志分析、海量数据排序等任务中。MapReduce 编程模式实际上是基于一种传统的分治法而实现的。分治法将复杂问题分成多个

类似的子问题，直到子问题的规模小到能直接得出结果，再聚合中间数据所得到的最终结果就是原问题的解。

2. Spark 编程框架

Spark 是 UC Berkeley AMP 实验室开源的类 Hadoop MapReduce 的通用的并行计算框架，Spark 基于 MapReduce 算法实现分布式计算，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是，Spark 中任务的输出和结果可以保存在内存中，从而不再需要读写 HDFS，因此，Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

3. Storm 编程框架

Storm 是一款最早由 BackType 公司（现已被 Twitter 公司收购）开发的分布式实时计算系统。Storm 为分布式实时计算提供了一组通用原语，其用法与 Hadoop 极其类似，也被称为实时计算版的 Hadoop。它也可被用于流处理中，实现实时处理消息并更新数据库。同时 Storm 可以采用任意编程语言编写。

4. Petuum 编程框架

Petuum 是一个专门针对机器学习的分布式平台，它致力于提供一个超大型机器学习的通用算法和系统接口。它的出现主要解决两类机器学习在规模上面临的问题：大数据（大量的数据样本）和大模型（大量的模型参数）。Petuum 能够在集群和云计算（如 Amazon EC2 和 Google GCE）上高效运行。

1.4.3 机器学习与数据分析平台

1. scikit-learn

scikit-learn 是开源的 Python 机器学习库，最早由数据科学家 David Cournapeau 于 2007 年发起，它基于 NumPy 和 Scipy，提供了大量用于数据挖掘和分析的工具。scikit-learn 可以实现数据预处理、分类、回归、降维、模型选择等常用的机器学习算法。

2. NetworkX

NetworkX 是 Python 的一个开源包，用于对复杂网络进行创建、操作和学习。利用 NetworkX 可以以标准化和非标准化的数据格式存储网络、生成多种随机网络和经典网络、分析网络结构、建立网络模型、设计新的网络算法、进行网络绘制等。

1.4.4 数据可视化工具

1. Matplotlib

Matplotlib 是 Python 常用的数据绘制包，绘图功能强大，可以轻易地画出各种统计图形，如散点图、条形图、饼状图等。Matplotlib 包含两个重要模块 Pylab 和 Pyplot。Pylab 已经几乎实现了 MATLAB 所支持的所有绘图功能，Pyplot 在 Pylab 的基础上支持数学运算套件 NumPy，使用者可以直接调用 NumPy 函数做计算后进行数据可视化。

2. Orange

Orange 的全部内容都是关于数据可视化，帮助发现隐藏的数据模式，提供数据分析过程背

后的直觉或支持数据科学家与领域专家之间的交流。可视化窗口小部件包括散点图、箱形图和直方图，以及特定于模型的可视化，如树状图、轮廓图和树可视化。许多其他可视化功能可用于附加组件，包括网络、词云、地理地图等的可视化。

1.5 本书的内容和组织

本书将主要从算法的角度介绍数据分析所使用的主要技术和原理，对于经典的数据分析算法，本书会配备大量的应用实例。本书主要内容为数据分析的基础概念和基本技术，对于有志于从事数据分析领域的读者，本书可以作为一个很好的起点。

我们将从 Python 编程语言（第 2 章）开始本书的技术讨论。该章将为读者介绍进行数据分析应用最广泛的编程语言 Python。通过此章的介绍，读者能够掌握 Python 语言的语法规则、编程规范、程序调试及异常处理方法，为进行数据分析奠定编程基础。

第 3 章将详细介绍数据分析的具体过程，教授读者如何开展一个完整的数据分析项目。如何进行数据准备、如何快速地理解数据并选择合适的模型算法，如何应用统计学的相关方法进行数据的描述性分析。

数据分析的主要任务，即回归、聚类、分类和降维，将依次在第 4 章、第 5 章、第 6 章和第 7 章进行介绍。各章的内容主要包括基本概念、经典算法的原理和实际步骤、算法的应用实例，最后探讨各个问题最新的前沿技术。

本书的第 8 章将介绍社交网络分析，包括对网络中的结点进行重要性排序、网络链路预测方法、网络结构的社团检测及各自的评价指标和实际应用。

第 9 章将详细介绍数据可视化技术，介绍如何基于 Matplotlib 可视化工具绘制各种数据图表，让读者能够更好地展示数据分析的结果。

思考与练习

1. 填空题

（1）IBM 提出大数据具有 5V 特点：_____、_____、_____、_____、_____。

（2）通过数据分析，可以发现个体对象的_____、类别分布，也可以_____个体对象未来的观测结果。

（3）对实际数据进行探索、提取知识的过程一般包括_____、数据分析、_____等步骤。

（4）由于数据分析任务中实际数据量的巨大差异，要求数据分析的模型算法能够处理大数据集，且时间复杂度不能太高，消耗的内存空间也有限，即必须是_____。

（5）_____是 Hadoop 的核心子项目，是一个可以运行在普通硬件设备上的分布式文件系统。

2. 简答题

（1）什么是数据分析？

（2）画出基于数据分析的知识提取过程。

(3)典型的数据领域包括哪些?

(4)常见的数据类型有哪些?

(5)对于降维任务,当前提出的方法有哪些?

第2章

Python 编程语言

(1)知道什么是 Python 编程语言,并了解 Python 编程的基本规范。
(2)了解 Python 编程环境的构建方法。
(3)学习 Python 典型的数据结构。
(4)了解函数、模块和包。
(5)理解 Python 的原理及其异常处理机制。

(1)Python 编程的语法。
(2)Python 语言的变量与数据结构。
(3)Python 语言中函数、模块和包的构建与使用方法。
(4)Python 编程的调试方法。

2.1 初识 Python

常见的计算机编程语言分为两类,即动态编程语言和静态编程语言。其中,动态编程语言是使用变量前不需要声明变量,在运行期间才去做数据类型检查的语言。编程时,不用给变量指定数据类型,该语言会在第一次给变量赋值时在内部将数据类型记录下来。例如,Python、JavaScript、PHP、Ruby 等语言是动态编程语言。而静态编程语言在使用变量前需要声明变量,数据类型检查发生在编译阶段,也就是说,在写程序时,要声明变量的数据类型。例如,Java、C++、C、Go 等语言就是静态编程语言。本书使用的 Python 语言是一门具有静态编程语言工程项目开发能力的动态编程语言。

Python 是一种跨平台的、开源的、免费的、解释型的高级编程语言,近几年发展迅猛,在编程语言排行榜名列第一。Python 的应用领域非常广泛,其中包括 Web 开发、大数据处理、云计算、机器

学习、自动化运维、网络爬虫、游戏开发等，如图 2-1 所示。Python 语言与当前其他编程语言相比有如下优点：学习简单、兼容性良好、面向对象、自动内存管理、第三方库丰富。

图 2-1　Python 编程语言的应用领域

2.1.1　Python 概述

Python 是由荷兰人 Guido van Rossum 于 1991 年发明的一种面向对象的解释型高级编程语言，其标志如图 2-2 所示。Python 语言的设计哲学为优雅、明确、简单，实际上，这门语言始终贯穿这一理念。Python 语言的目标是让编程人员的主要精力用于思考程序的逻辑，而不是程序本身的实现细节。Python 语言的提出受到了 C 语言、Shell 语言、ABC 语言等多种编程语言的影响，综合了多种编程语言的特点和优势。Python 语言的简单性、开放性、扩展性，使它受到广大程序员的欢迎，成为一个优秀并广泛使用的计算机编程语言。由于开源社区的贡献，Python 语言通过标准库及第三方包形成了功能强大的 Python 生态系统。

图 2-2　Python 的标志

2.1.2　Python 环境搭建

为了在 Windows 上配置 Python 开发环境，首先须从 Python 的官方网站下载安装包，下载地址为 https://www.python.org/downloads/windows/，如图 2-3 所示。

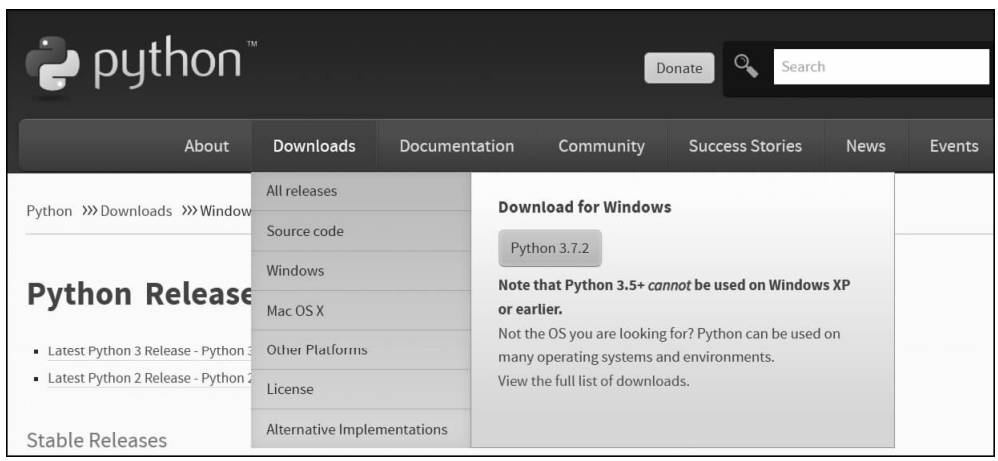

图 2-3　下载 Python 安装文件

运行下载的 Python 3.7.2.exe 文件，在选择安装组件时，选择所有的组件，特别要注意选中“Add Python 3.7 to PATH”复选框，如图 2-4 所示。然后连续单击 Next 按钮即可完成安装，如图 2-5 所示。

图 2-4　Python 安装过程

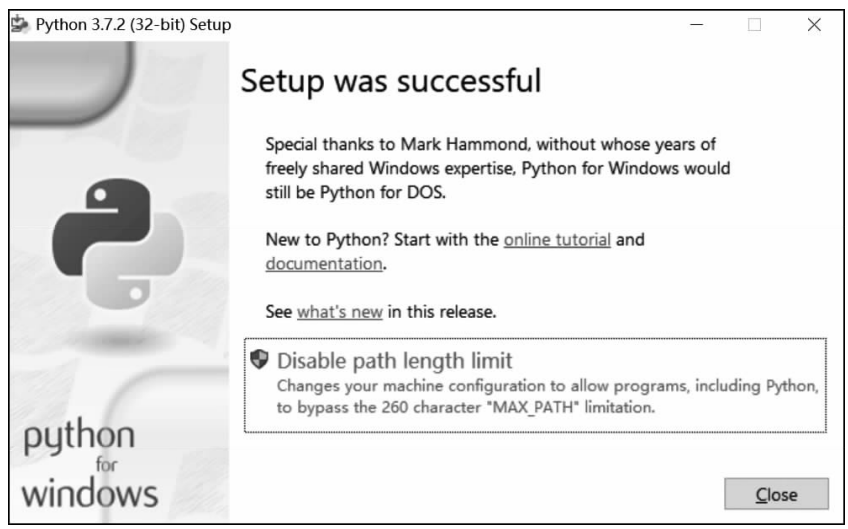

图 2-5　Python 安装结果

选择将 Python 安装到 C:\Python37 目录下，安装完成后，打开命令提示符窗口，输入 Python 并回车后，会出现两种情况。

情况一：看到图 2-6 所示的界面，说明 Python 安装成功，提示符“>>>”说明已经在 Python 交互式环境中。输入 exit()并回车，退出 Python 交互式环境。

```
C:\Python37>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 2-6　Python 命令行交互环境

情况二：得到一个错误提示："Python 不是内部或外部命令，也不是可运行的程序或批处理文件。"这说明 Windows 没有找到对应的可执行程序，需要将 Python 添加到 Path 环境变量中。

接下来进行 Python 环境变量修改。右击桌面上的计算机图标，在弹出的快捷菜单中选择"属性"命令，打开"系统"窗口，在左侧窗格中单击"高级系统设置"链接，弹出"系统属性"对话框，切换到"高级"选项卡，单击"环境变量"按钮，如图 2-7 所示。

图 2-7 "系统属性"对话框

随即弹出"环境变量"对话框，在"系统变量"列表框中添加 Python 的安装路径，如图 2-8 和图 2-9 所示。

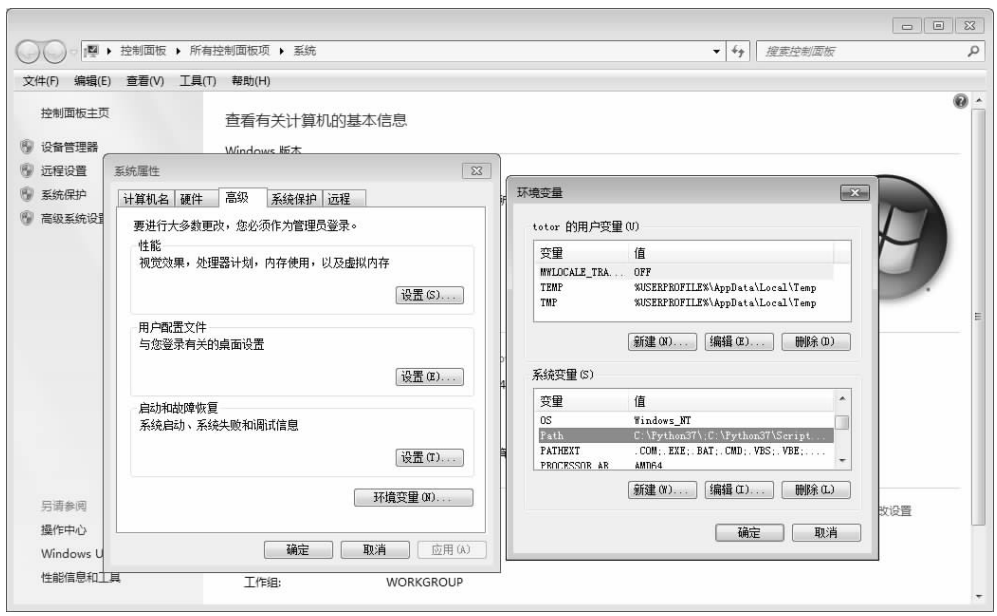

图 2-8 "环境变量"对话框

图 2-9 “编辑系统变量”对话框

在安装 Python 编译器之后，为了方便 Python 程序开发，可以选择安装编辑器。常见的编辑器有 Sublime Text、Vim、PyCharm、文本编辑器等，我们选择 PyCharm 作为 Python 编辑器，官网地址为 http://www.jetbrains.com/pycharm/。下载完成后，双击安装程序，弹出图 2-10 所示的界面。

图 2-10 PyCharm 安装界面

单击 Next 按钮，弹出 Choose Install Location 界面，设置程序的安装路径，如图 2-11 所示。

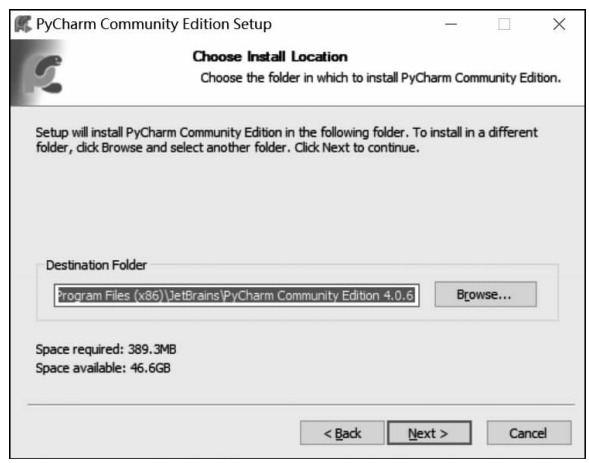

图 2-11 PyCharm 安装路径选择

单击 Next 按钮，开始安装 PyCharm，如图 2-12 所示。

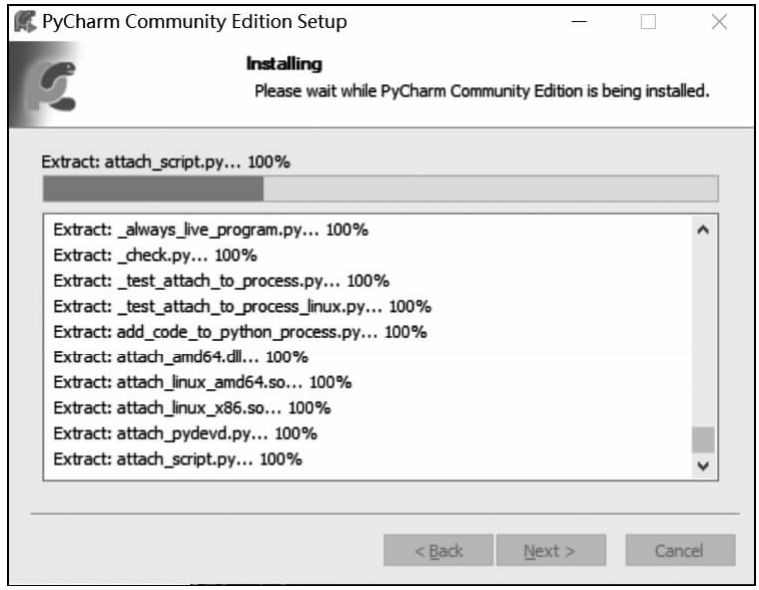

图 2-12　PyCharm 安装过程

完成 PyCharm 安装的结果如图 2-13 所示。

图 2-13　PyCharm 安装结果

打开 PyCharm 编辑器，在 PyCharm 中编写 Python 代码。首先，新建一个 Python 工程，执行 File→New Project 命令，确定工程的存储位置和对应的编译器，如图 2-14 和图 2-15 所示。

在工程上面右击，然后在弹出的快捷菜单中执行 New→Python File 命令，如图 2-16 所示，创建一个新的 Python 文件。

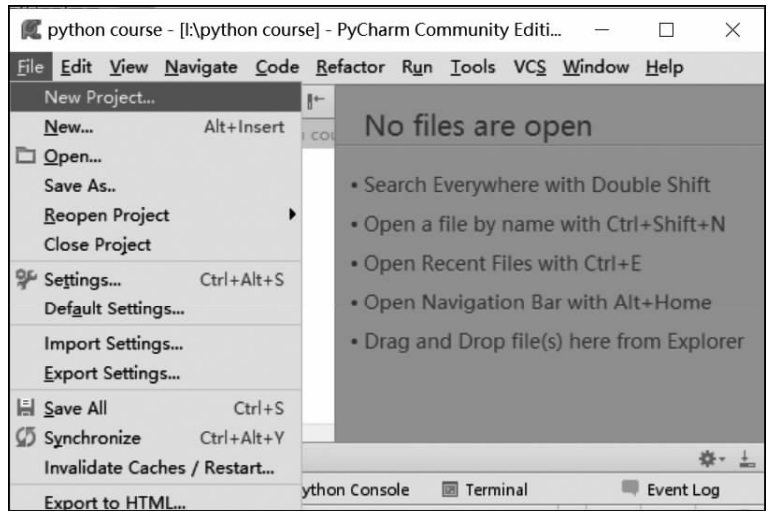

图 2-14　在 PyCharm 中新建 Python 工程

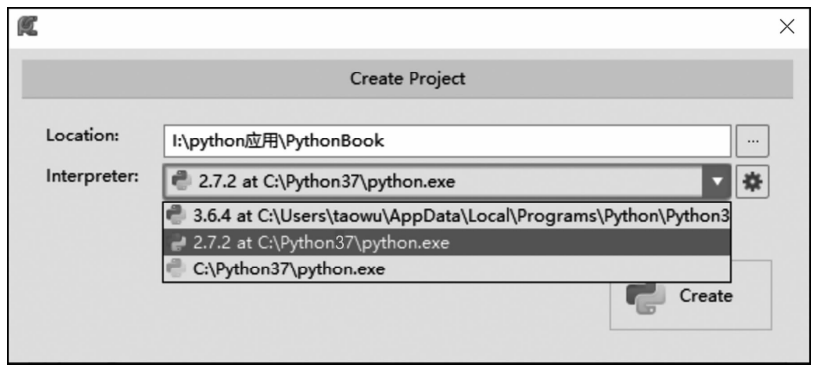

图 2-15　设置工程名称与解释器

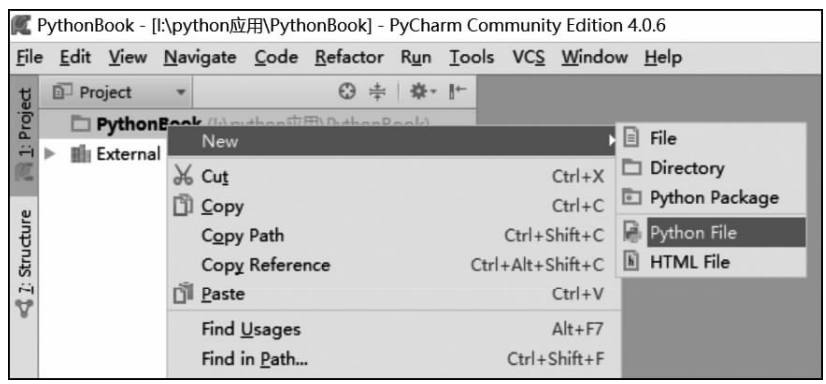

图 2-16　在工程中新建 Python 文件

然后，在新建的 Python 文件中进行编码，完成需要的相关功能。对于完成编码的 Python 文件，在文件名上右击并选择 Run 命令即可执行文件，如图 2-17 所示。在代码执行完毕之后，程序的执行结果或者可能出现的相关错误信息会出现在 PyCharm 界面下方的结果区域。在本实例中，按照程序要求输入用户名称，然后程序执行完毕输出最终结果，如图 2-18 和图 2-19 所示。

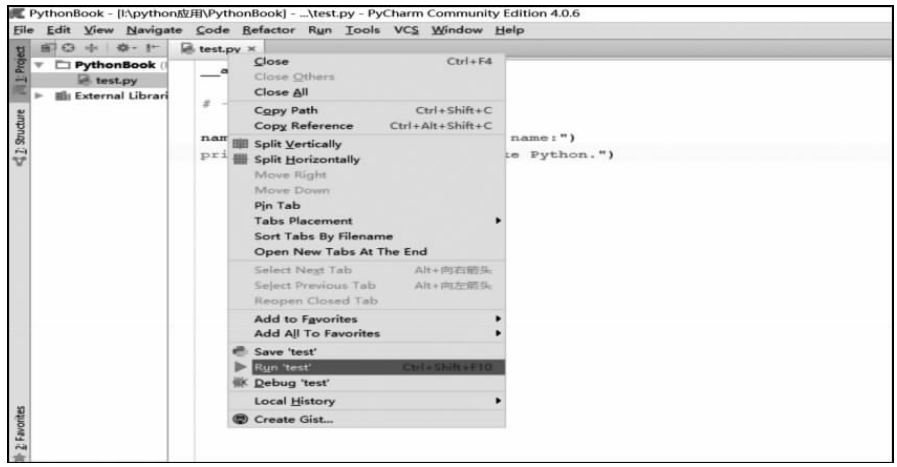

图 2-17　Python 代码的编写与运行

图 2-18　按程序要求进行用户输入

图 2-19　代码运行结果

2.1.3 Python 语法特点

Python 语法特点主要包括注释规则、代码缩进及编码规范。

1. Python 注释

Python 注释是指在 Python 程序代码中添加的标注性文字，类似于产品的使用说明。当程序被处理时，这些注释会被自动忽略不会被当作代码处理。通过添加程序注释，可以帮助我们理清代码逻辑。在与别人合作进行程序开发时，添加注释可以减少沟通成本。在 Python 模块开发时添加注释可以减少他人的使用成本。另外，在编码过程中，可以临时注释一段代码，方便调试。具体地，Python 程序的注释主要可分为以下三类。

(1)单行注释，即＃，如以下代码所示：

```
1.  ＃输出人员名称
2.  print("Nan Shan")
```

(2)多行注释，通常用 '''、"""表示，将多行注释文字放在这两个标识之间，或者在每行需要注释的内容之前加＃，具体用法如以下代码所示：

```
1.  '''注释信息'''
2.  """注释信息"""
3.  ＃在行首添加
4.
5.  '''
6.  输出人员名称
7.  此处姓名是 Nan Shan
8.  '''
9.  print("Nan Shan")
```

(3)特殊注释，包括解释器说明和中文支持。在通过“./”执行 Python 程序时，在程序中应指明解释器的路径，如以下代码所示：

```
1.  ＃! /usr/bin/python
2.  print"你好，世界"
```

Python 程序中如果包含中文且未指定编码，在执行过程会出现语法报错“SyntaxError: Non-ASCII character”。实际上，Python 中默认的编码格式是 ASCII 格式，在未修改编码格式时无法正确打印汉字，所以在读取中文时会报错。解决方法为只要在文件开头加入＃-＊-coding:utf-8-＊-或者＃coding＝utf-8 就可以了。

```
1.  ＃! /usr/bin/python
2.  ＃-＊-coding:utf-8-＊-
3.  print"你好，世界"
```

2. 代码缩进

在 Python 编程语言中，代码缩进是指在每一行代码左端空出一定长度的空白，从而可以更

加清晰地从外观上看出程序的逻辑结构。具体地，在 Python 里不能用括号来表示语句块；行首的空白（空格或制表符）用来决定逻辑行的缩进层次，从而用来决定语句的分组；同一层次的语句必须有相同的缩进，每一组这样的语句称为一个块。例如：

```
__author__='taowu'
#-*-coding:utf-8-*-
#Filename:test.py
#function:input checking

num=float(input("输入一个数字:"))
if num>0:
    print("正数")
elif num==0:
    print("零")
 else:
    print("负数")
```

3. 编码规范

除了注释和代码缩进之外，Python 程序还应该遵守基本的编码规范：

（1）每个 import 语句只导入一个模块，尽量避免一次导入多个模块；

（2）不要在行尾添加分号“;”，也不要用分号将两条命令放在同一行；

（3）运算符两侧、函数参数之间、逗号“,”两侧建议使用空格进行分隔；

（4）建议每行不超过 80 个字符；

（5）使用必要的空行可以增加代码的可读性；

（6）适当使用异常处理结构提高程序容错性。

2.1.4 Python 的基本数据类型

Python 的基本数据类型包括数值类型、字符串和布尔值，不同数据类型之间可以基于特定的规则进行转换。

1. 数值类型

（1）整型（int）：正整数或负整数，不带小数点。

（2）浮点型（float）：由整数部分与小数部分组成。

（3）复数（complex）：由实数部分和虚数部分构成，可以用 a+bj 或者 complex(a,b)表示。

数值类型支持加、减、乘、除等相关操作，同时 Python 整数运算结果仍然是整数，浮点数运算结果仍然是浮点数。Python 整数和浮点数混合运算的结果就变成浮点数。整数除法，即使除不尽，结果仍然是整数，余数直接被舍掉。相关运算实例代码如下：

```
#! /usr/bin/python
#-*-coding:utf-8-*-

```

```
4.  1+2+3                                  #===>6
5.  4*5-6                                  #===>14
6.  7.5/8+2.1                              #===>3.0375
7.
8.  #Python 整数运算结果仍然是整数,浮点数运算结果仍然是浮点数
9.  1+2                                    #===>整数 3
10.  1.0+2.0                               #===>浮点数 3.0
11.
12.  #整数和浮点数混合运算的结果变成浮点数
13.  1+2.0                                 #===>浮点数 3.0
14.
15.  #整数除法,即使除不尽,结果仍然是整数,余数直接被舍掉
16.  11/4                                  #===>2
17.  11%4                                  #===>3
```

2. 字符串

字符串是以 ' ' 或" "括起来的任意文本,如 'abc',"xyz"等。注意,' ' 或" "本身只是一种表示方式,不是字符串的一部分。字符串举例说明如下:

```
1.  s='???'
2.  print(s)                               #===>???
3.  s="I'm OK"
4.  print(s)                               #===>I'm OK
```

3. 布尔值

布尔值和布尔代数的表示完全一致,一个布尔值只有 True、False 两种值。注意,Python 把 0、空字符串 '' 和 None 看成 False,其他数值和非空字符串都看成 True。布尔值举例说明如下:

```
1.  Flag1=True
2.  Flag2=False
3.  print(Flag1 and Flag2)                 #===>False
```

4. 数据类型转换

(1)int(x):将 x 转换为一个整数。

(2)float(x):将 x 转换为一个浮点数。

(3)str(x):将 x 转换为一个字符串类型。

(4)complex(x):将 x 转换为一个复数,实部为 x,虚部为 0。

(5)complex(x, y):将 x 和 y 转换为一个复数,实部为 x,虚部为 y。

(6)hex(x):将整数转换为十六进制字符串。

(7)oct(x):将整数转换为八进制字符串。

5. 运算符

Python 的基本运算符包括算术运算符、赋值运算符、比较运算符、逻辑运算符和位运算符。假设变量 a 的值是 10，变量 b 的值是 21，则应用算术运算符的示例如表 2-1 所示。

表 2-1 算术运算符

运 算 符	符号作用	示 例
+	加法运算，将运算符两边的操作数相加	a+b=31
−	减法运算，将运算符左边的操作数减去运算符右边的操作数	a−b=−11
*	乘法运算，将运算符两边的操作数相乘	a * b=210
/	除法运算，用左操作数除以右操作数	b/a=2.1
%	模运算，用左操作数除以右操作数并取余	b%a=1
**	对运算符进行指数计算	a ** b，表示 a 的 b 次幂

假设变量 a 的值为 10，变量 b 的值为 20，则应用赋值运算符的示例如表 2-2 所示。

表 2-2 赋值运算符

运 算 符	符号作用	示 例
=	将右操作数的值赋给左操作数	c=a，表示将 a 的值赋给 c
+=	将右操作数相加到左操作数并将结果分配给左操作数	c+=a，表示 c=c+a
−=	将左操作数减去右操作数并将结果分配给左操作数	c−=a，表示 c=c−a
*=	将左操作数与右操作数相乘，并将结果分配给左操作数	c *=a，表示 c=c * a
/=	将左操作数除以右操作数并将结果分配给左操作数	c/=a，表示 c=c/a
%=	将左操作数除以右操作数的模数并将结果分配给左操作数	c%=a，表示 c=c%a
**=	执行指数计算并将结果分配给左操作数	c * *=a，表示 c=c * *a

假设变量 a 的值为 10，变量 b 的值为 20，则应用比较运算符的示例如表 2-3 所示。

表 2-3 比较运算符

运 算 符	符号作用	示 例
==	若两个操作数的值相等，则条件为真	a==b 的结果为 False
!=	若两个操作数的值不相等，则条件为真	a!=b 的结果为 True
>	若左操作数大于右操作数的值，则条件为真	a>b 的结果为 False
<	若左操作数小于右操作数的值，则条件为真	a<b 的结果为 True
>=	若左操作数大于等于右操作数的值，则条件为真	a>=b 的结果为 False
<=	若左操作数小于等于右操作数的值，则条件为真	a<=b 的结果为 True

假设变量 a 的值为 True，变量 b 的值为 False，则应用逻辑运算符的示例如表 2-4 所示。

表 2-4　逻辑运算符

运 算 符	符号作用	示　　例
and	若两个操作数都为真，则条件成立	a and b 的结果为 False
or	若两个操作数中任何一个非零，则条件为真	a or b 的结果为 True
not	用于反转操作数的逻辑状态	not(a and b)的结果为 True

假设变量 a＝60，b＝13(a＝00111100，b＝00001101)，则应用位运算符的示例如表 2-5 所示。

表 2-5　位运算符

运 算 符	符号作用	示　　例
&	若它存在于两个操作数中，则操作符复制位到结果	a&b 的结果为 00001100
|	若它存在于任一操作数，则复制位	a|b 的结果为 00111101
^	二进制异或，若它是一个操作数集合，但不同时是两个操作数则将复制位	a^b 的结果为 00110001
～	二进制补码，它是一元的，具有翻转的效果	～a＝－61(有符号的二进制数)，表示为 11000011 的补码形式
＜＜	二进制左移，左操作数按右操作数指定的位数左移	a＜＜2 的结果为 11110000(240)
＞＞	二进制右移，左操作数按右操作数指定的位数右移	a＞＞2 的结果为 00001111(15)

2.2　字符串与流程控制

2.2.1　字符串

字符串中常用的操作有拼接字符串、计算字符串长度、截取字符串、分割字符串和合并字符串。

1. 拼接字符串

在 Python 中，使用“＋”运算符可完成多个字符串的拼接。例如：

```
1.  '''拼接字符串'''
2.  str1='我今天一共走了'
3.  num=19888
4.  str2='步'
5.  print(str1+str(num)+str2)
```

输出“我今天一共走了 19888 步”。字符串不允许与其他类型的数据拼接，否则会报错。

2. 计算字符串长度

在 Python 中，使用 len()函数可计算字符串的长度。例如：

```
"""字符串长度"""
str='我今天一共走了 19888 步'
length=len(str)
print(length)
```

输出“13”。默认情况下，不区分英文、数字和汉字，所有字符都按一个字符计算。但是，对于不同的编码(UTF-8，GBK)，一个汉字会占不同的字节。

3. 截取字符串

可以使用切片方法实现字符串的截取。语法格式如下：

```
string[start:end:step]
```

例如：

```
str='我爱学 Python!'
print(str)                    #我爱学 Python!
print(str[1])                 #爱
print(str[:5])                #我爱学 Py
print(str[5:])                #thon!
```

4. 分割字符串

使用 split()函数可实现字符串的分割。语法格式如下：

```
str.split(sep,maxsplit)
```

例如：

```
str='我爱学>>>>https://www.Python.org/'
print(str.split())       #['我','爱','学','>>>>','https://www.Python.org/']
print(str.split('>>>>')) #['我爱学','https://www.Python.org/']
print(str.split('.'))    #['我爱学>>>>https://www','Python','org/']
```

5. 合并字符串

使用 joint()函数可实现字符串的合并。语法格式如下：

```
strnew=string.join(iterable)
```

例如：

```
goods=['番茄','土豆','南瓜']
str_goods='|'.join(goods)     #用|进行连接
print(str_goods)              #番茄|土豆|南瓜
```

2.2.2 流程控制

在计算机编程语言中，程序的流程控制大致包括三种逻辑结构，即顺序结构、选择结构和循环结构，如图 2-20 所示。具体地，顺序结构是指顺序执行程序中包含的所有语句；选择结构根据程序中的条件表达式进行判断，根据不同的结果执行不同的语句块；循环结构是指在满足条件表达式要求时多次循环执行某段程序语句，然后在不满足条件表达式要求时退出循环。由于顺序结构不需要任何流程控制语句，本节主要讲解 Python 语言中与选择结构和循环结构相关的逻辑控制语句。

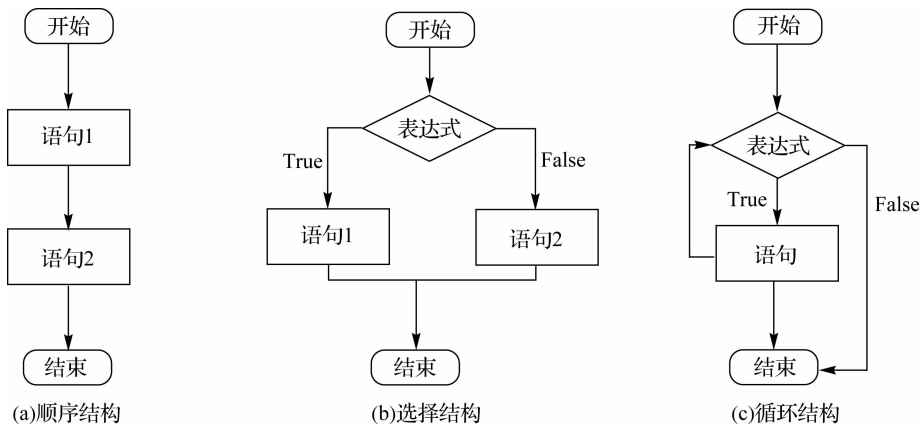

图 2-20 三种基本的逻辑结构

1. 选择语句

(1)if…else…语句。一个 if 语句后可跟一个可选的 else 语句，如果布尔表达式为 True，则执行 if 块内的代码。如果布尔表达式为 False，则执行 else 块内的代码。例如：

```
1.  name='Nan Shan'
2.  if name=='Python':                  #判断变量是否为 'Python'
3.      print('I like Python')          #并输出欢迎信息
4.  else:
5.      print(name)                     #条件不成立时输出变量名称
```

(2)if…elif…else 语句。一个 if 语句后可跟可选的 elif…else 语句，这可用于测试多种条件。if 语句之后可跟一个或多个 elif…else 语句，如果某一条件表达式为真，则执行相应的语句块。只有在所有的条件表达式都为假的情况下，才会执行 else 语句。例如：

```
1.  num=5
2.  if num==3:                          #判断 num 的值
3.  print('administrator')
4.  elif num==2:
5.      print('user')
6.  elif num==1:
```

```
    print('guest')
else:
    print('attacker')              # 条件均不成立时输出
```

2. 循环语句

(1)while 循环语句。通过一个条件来判断是否要继续执行循环体,若条件为真则执行,否则退出。例如:

```
flag=True
num=1
while flag:                        # 条件表达式 1
    print('Hello')
    num+=1
    if num>4:                      # 条件表达式 2
        flag=False
```

(2)for 循环语句。for 循环允许编写一个执行指定次数的循环控制结构,在达到指定次数之前,不断执行程序块,并且在每次执行之前进行条件检查。如果达到了指定的循环次数,则退出。例如:

```
for letter in 'Python':            # 第一个实例
    print('当前字母:',letter)

fruits=['banana','apple','mango']
for fruit in fruits:               # 第二个实例
    print('当前字母:',fruit)
print("Good bye!")
```

3. 跳转语句

对于循环结构,除了基于 while 循环语句和 for 循环语句构建循环逻辑之外,在满足特定条件时,需要直接退出循环或者跳过循环结构中某一轮的执行。因此,需要跳转语句进行 Python 循环结构的逻辑控制。

(1)break 语句。其用于终止当前循环(打破最小封闭 for 或 while 循环)。例如:

```
for letter in 'Python':            # 第一个实例
    if letter=='h':
        break
    print('当前字母:',letter)
```

输出结果：

```
当前字母：P
当前字母：y
当前字母：t
```

(2)continue 语句。其用于提前终止本次循环而进入下一次循环中。例如：

```
1.  for letter in 'Python':
2.      if letter==='h':
3.          continue
4.      print('当前字母:',letter)
```

输出结果：

```
当前字母：P
当前字母：y
当前字母：t
当前字母：o
当前字母：n
```

(3)pass 语句。其不做任何事情，仅起到占位作用，方便以后需要时进行补充。例如：

```
1.  for letter in 'Python':
2.      if letter==='h':
3.          pass
4.      print('这是 pass 块')
5.      print('当前字母:',letter)
6.  print("Good bye!")
```

2.3 列表、元组与字典

序列是 Python 中最基本的数据结构。对序列中的每个元素，都有一个数字与之相对应。通过这个数字，可以访问序列中的数据元素，此数字可以被称为元素的位置或者索引。Python 有 6 个序列的内置类型，最常见的是列表、元组和字典。

2.3.1 列表

列表是最常用的 Python 数据类型，其数据项不需要具有相同的类型。创建一个列表，只要把由逗号分隔的不同数据项用方括号括起来即可。如下所示：

```
1.  list1=[1,2,3,4,5];
2.  list2=["a","b","c","d"];
```

1. 遍历列表

对列表中的每个元素进行查询和处理。例如：

```
1.  list=['sichuan','chongqing','guangdong','hunan']
2.  for item in list:
3.      print(item)
```

输出结果：

```
sichuan
chongqing
guangdong
hunan
```

2. 添加、修改和删除列表

可以通过 append()函数在列表末尾添加元素。对于列表中的已有元素，可以通过索引直接进行修改和删除。例如：

```
1.  list=['sichuan','chongqing','guangdong','hunan']
2.  list.append('jiangsu')
3.  print(list)
```

输出：

```
['sichuan', 'chongqing', 'guangdong', 'hunan', 'jiangsu']
```

```
1.  list=['sichuan','chongqing','guangdong','hunan']
2.  list[0]=''
3.  print(list)
```

输出：

```
['', 'chongqing', 'guangdong', 'hunan']
```

```
1.  list=['sichuan','chongqing','guangdong','hunan']
2.  del list[3]
3.  print(list)
```

输出：

```
['sichuan', 'chongqing', 'guangdong']
```

3. 统计和计算

通过 count()获取列表中指定元素的出现次数，代码如下：

```
1.  list=['sichuan','chongqing','sichuan','hunan']
2.  list.count('sichuan')
```

通过 index()获取指定元素首次出现的下标，代码如下：

```
1.  list=['sichuan','chongqing','sichuan','hunan']
2.  list.index('sichuan')
```

通过 sum()获取数值列表的元素和，代码如下：

```
1.  list=[10,10,10,10]
2.  print(sum(list))
```

4. 排序

使用列表对象的 sort()方法改变原始列表元素的排列顺序，其中可以通过 reverse 参数设置排序规则，reverse 等于 True，则降序排列，否则升序排列。例如：

```
1.  char=['cat','Tom','Angela','pet']
2.  char.sort()
3.  print(char)
```

输出：

```
['Angela', 'Tom', 'cat', 'pet']
```

```
1.  num=[43,23,111,71]
2.  num.sort(reverse=True)
3.  print(num)
```

输出：

```
[111, 71, 43, 23]
```

2.3.2 元组

元组由一系列按特定顺序排列的元素组成，其中的元素不可修改。与列表使用方括号[]不同的是，元组使用圆括号“()”。元组可以使用下标索引来访问元组中的值，实例如下：

```
1.  tup1=('physics','chemistry',1997,2000)
2.  tup2=(1,2,3,4,5,6,7)
3.  print("tup1[0]:",tup1[0])
4.  print("tup2[1:5]:",tup2[1:5])
```

输出：

```
tup1[0]: physics
tup2[1:5]: (2, 3, 4, 5)
```

元组具有以下优点：

(1)速度比列表更快。如果定义了一系列常量值，而所需做的仅是对它进行遍历，那么一般使用元组而不用列表。

(2)对不需要改变的数据进行“写保护”将使代码更加安全。

(3)可用作字典键，而列表不能当作字典键使用。

2.3.3 字典

字典是一种可变容器模型，可存储任意类型对象，其中的每个键值对(key=>value)用冒号(:)分割，每个对之间用逗号(,)分割，整个字典包括在花括号({})中，语法格式为

```
d={key1:value1,key2:value2}
```

例如：

```
1.  dict={'A1':'11','B1':'22','C1':'33'}
2.  dict1={'abc':456}
3.  dict2={'abc':123,98.6:37}
```

在定义了字典之后，需要判断字典是否存在某个特定的 key，这可以通过 has_key()函数实现。如下所示：

```
1.  dict={'A1':'11','B1':'22','C1':'33'}
2.  print(dict.keys())
3.  if dict.has_key('B1'):
4.      print('B1 exists')
```

输出：

```
['A1', 'C1', 'B1']
B1 exists
```

为字典添加新内容的方法是增加新的键值对，修改已有键值对，如下所示：

```
1.  dict={'Name':'Zara','Age':7,'Class':'First'}
2.  dict['Age']=18                          #修改字典内容
3.  dict['School']="CQUPT"                  #为字典添加新内容
4.  print("dict['Age']:",dict['Age'])
5.  print("dict['School']:",dict['School'])
```

输出：

```
("dict['Age']:", 18)
("dict['School']:", 'CQUPT')
```

对于字典中的元素，可以通过不同方法进行删除，包括删除字典中的某一元素、删除字典中

的全部元素及完全删除字典。具体如下：

```
1.  dict={'Name':'Zara','Age':7,'Class':'First'}
2.  print(dict)
3.  del dict['Name']                    #删除键是 'Name' 的条目
4.  print(dict)
5.  dict.clear()                        #清空字典所有条目
6.  print(dict)
7.  del dict                            #删除字典
```

输出：

```
{'Age': 7, 'Name': 'Zara', 'Class': 'First'}
{'Age': 7, 'Class': 'First'}
{}
```

2.4 函数、模块与包

2.4.1 函数

函数是组织好的、可重复使用的、用来实现单一或相关功能的代码段。函数能提高应用的模块性和代码的重复利用率。Python 提供了许多内建函数，如 print()，同时用户也可以自己创建函数，即自定义函数。

自定义函数中的函数代码块以 def 关键词开头，后接函数名称和圆括号()。圆括号之间可以用于定义参数，函数的第一行语句可以存放函数说明。函数内容以冒号起始，以 return 语句结束。

定义一个简单的 Python 函数，它将一个字符串作为传入参数，再打印到标准显示设备上。

```
1.  def printme(str):
2.      #打印传入的字符串到标准显示设备上
3.      print(str)
4.      return
5.  printme("调用自定义函数")
```

定义一个函数只需给出函数名称，指定函数里包含的参数及代码块结构。函数的基本结构完成以后，便可以通过另一个函数调用执行。

1. 参数传递

对于 Python 语言，自定义函数的参数对应的对象可以是不可变对象或者可变对象。在 Python 中，整型、字符串、元组等是不可更改的对象，而列表、字典等则是可变对象。对于不可变参数传递，函数 fun(a)传递的只是对象 a 的值，没有影响对象 a 本身。在 fun(a)内部修改 a 的

值，只是修改另一个复制的对象，不影响 a 本身的值。相对地，传递可变对象作为参数是引用传递，函数 fun(a)会将对象 a 真正地传过去，在函数 fun(a)内部修改 a 的值，函数外部 a 的取值也会发生变化。例如：

```
#Python 传递不可变对象
#! /usr/bin/Python
#-*-coding:UTF-8-*-

def ChangeInt(a):
    a=10

b=2
ChangeInt(b)
print(b)                          #结果是 2
```

```
#Python 传递可变对象
#! /usr/bin/Python
#-*-coding:UTF-8-*-

def changeme(mylist):
    mylist.append([1,2,3,4])      #"修改传入的列表"
    print("函数内取值:",mylist)
    return

mylist=[10,20,30]                 #调用 changeme 函数
changeme(mylist)
print("函数外取值:",mylist)
```

输出：

```
函数内取值:[10, 20, 30, [1, 2, 3, 4]]
函数外取值:[10, 20, 30, [1, 2, 3, 4]]
```

2. 返回值

在 Python 中，可以在函数体内使用 return 语句为函数指定返回值，该返回值可以是任意类型，可以返回参数值或返回 None。只要 return 语句被执行，就会直接结束函数的执行。例如：

```
#Python 函数返回值
def sum(arg1,arg2):
    #返回 2 个参数的和
    total=arg1+arg2
```

```
    print("函数内:",total)
    return total

#调用 sum 函数
total=sum(10,20)
print("函数外:",total)
```

输出：

```
函数内：30
函数外：30
```

3. 变量作用域

一个程序的所有变量并不是在哪个位置都可以被访问的。访问权限决定于这个变量被赋值的位置。定义在函数内部的变量拥有一个局部作用域，即为局部变量。定义在函数外的拥有全局作用域，即为全局变量。局部变量只能在其被定义的函数内部访问，而全局变量可以在整个程序范围内访问。调用函数时，所有在函数内声明的变量名称都将被加入作用域中。

```
                                    #Python 函数变量作用域
total=0                             #全局变量
def sum(arg1,arg2):
    total=arg1+arg2                 #局部变量
    print("局部作用域:",total)
    return total

sum(10,20)                          #调用 sum 函数
print("全局作用域:",total)
```

输出：

```
局部作用域：30
全局作用域：0
```

2.4.2 模块

模块是能够独立完成某一方面功能的相关代码的集合。使用模块可以规范代码，让代码更易于阅读，也方便其他程序员使用已经编写好的代码，提高开发效率。为了创建模块，需要将相关代码编写在一个单独的文件中，并将此文件命名为“模块名+.py”的形式。例如：

```
#模块 support.py
def print_func(par):
    print("Hello:",par)
    return
```

将以上代码保存为名为“support. py”的文件，即创建了名为 support 的模块。为了使用已经存在的模块，需要先使用 import 语句进行模块加载。当解释器遇到 import 语句时，如果模块在当前的搜索路径中，就会被导入。为了从模块中导入一个指定部分到当前命名空间中，可以使用 from…import 语句。

```
1.  #导入模块
2.  import support
3.
4.  #现在可以调用模块中包含的函数了
5.  support.print_func("Zara")
```

在导入模块的过程中，Python 解析器对模块位置的搜索顺序依次是当前目录、PYTHONPATH 下的每个目录及默认路径。模块搜索路径存储在 system 模块的 sys. path 变量中。PYTHONPATH 变量由装在一个列表里的许多目录组成。在 Windows 系统中，典型的 PYTHONPATH 为：set PYTHONPATH = C: \ Python20 \ lib。在 UNIX 系统中，典型的 PYTHONPATH 为：set PYTHONPATH = /usr/local/lib/Python。

除了用户创建的自定义模块，Python 自带了许多实用的模块，称为标准模块。标准模块可以直接通过 import 语句导入使用。常用的标准模块有：random(随机)、math(数学)、sys(系统环境)、time(时间)、os(操作系统)、calendar(日期)、json(序列化)和 tkinter(GUI 编程)。另外，对于第三方模块，需要先下载安装，然后可以像标准模块一样使用。下载、安装第三方模块可以使用 pip 命令实现。例如，安装用于科学计算的 numpy、用于机器学习的 scikit-learn 等第三方模块，可以执行以下命令：

```
1.  pip install numpy
2.  pip install scikit-learn
```

2.4.3 包

包是由一个分层次的目录结构，将一组功能相近的模块组织在一个目录下。其作用是规范代码的使用，并避免模块名引起的冲突。创建包的步骤如下：

(1)创建一个文件夹，如 settings。

(2)在文件夹下创建一个名为“_init_. py”的空文件。

(3)在包中创建所需的模块，如 size. py 文件。

使用包的三种方式如下：

```
1.  import settings.size
2.  from settings import size
3.  from settings.size import width,height
```

2.5 Python 异常处理

在程序运行过程中，经常会遇到各种各样的错误，这些错误统称为异常。这些异常通常是由代码的编写错误导致的，这类错误产生的是 SyntaxError：invalid syntax（无效的语法）错误，会导致程序无法正确运行。异常实际上是一个表示错误的 Python 对象，当 Python 脚本发生异常时，我们需要捕获并处理它，否则程序会终止执行。Python 语言中常见的异常如表 2-6 所示。

表 2-6 Python 语言中常见的异常

常见异常	释　义
AttributeError	对象没有这个属性
IOError	输入/输出操作失败
ImportError	导入模块/对象失败
SyntaxError	Python 语法错误
TypeError	对类型无效的操作
ValueError	传入无效的参数

为了捕获 Python 程序中的异常，需要使用 try/except 语句。为了避免在运行过程中程序由于发生异常而崩溃，可以将程序写在 try 语句块内。这样，对于 try 语句块中发生的异常，except 语句会进行捕获并处理。try/except 语句的语法格式如下：

```
try:
    <语句>                    #主要代码
except <名字>:
    <语句>                    #如果在 try 部分引发了“名字”异常
else:
    <语句>                    #如果没有异常发生
```

异常捕获语句 try/except 的工作原理如下：

(1)当开始一个 try 语句后，Python 就在当前程序的上下文中做标记，这样当异常出现时就可以回到这里。

(2)try 子句先执行，接下来会发生什么依赖于执行时是否出现异常。如果发生异常，Python 就跳回到 try 并执行第一个匹配该异常的 except 子句，异常处理完毕后控制流就通过整个 try 语句；如果发生异常却没有匹配的 except 子句，异常将被递交到上层的 try，或者到程序的最上层。

异常捕获语句 try/except 的使用方法如下：

```
#! /usr/bin/Python
#-*-coding:UTF-8-*-
try:
```

```
    fh=open("testfile","w")
    fh.write("异常测试!")
except IOError:
    print "Error:没有找到文件或读取文件失败"
else:
    print "内容写入文件成功"
    fh.close()
```

除了以上用法,try/except 语句还可以用来同时捕获多个异常,语法格式如下:

```
try:
    正常的操作
    ......
except(Exception1[, Exception2[,…ExceptionN]]):
    发生以上多个异常中的一个,执行这块代码
    ......
else:
    如果没有异常,执行这块代码
```

为了实现无论是否发生异常都将执行某一部分代码,Python 提供了 try-finally 语句。其语法格式如下:

```
try:
    <语句>
finally:
    <语句>                    #退出 try 时总会执行
raise
```

try-finally 异常捕获语句实例如下:

```
#! /usr/bin/Python
#-*-coding:UTF-8-*-
try:
    fh=open("testfile","w")
    try:
        fh.write("异常测试!")
    finally:
        print"关闭文件"
        fh.close()
```

```
except IOError:
    print"Error:没有找到文件或读取文件失败"
```

如果在某个函数或者方法中可能产生异常，但不想在当前函数或方法中处理这个异常，则可以使用 raise 语句在函数或方法中抛出异常，等待后续处理。抛出异常的 raise 语句的语法格式如下：

```
raise ExceptionName(reason)
```

其中，ExceptionName 是异常的类型，其一般是标准异常中的任意一种。为了能够捕获异常，except 语句必须用相同的异常来进行捕获。reason 是可选参数，用来对抛出的异常进行描述。raise 异常抛出语句的实例如下：

```
#! /usr/bin/Python
#-*-coding:UTF-8-*-

def mye(level):
    if level<1:
        raise ValueError("Invalid level!")
#触发异常后，后面的代码就不会再执行
try:
    mye(0)                          #触发异常
except ValueError as err:
    print("Error1:",err)
else:
    print("Error2")
```

执行以上程序，输出：

```
"Error1: Invalid level!"
```

思考与练习

1. 简答题

(1)Python 语言和其他高级语言 C、C++等有什么区别？

(2)简述 Python 语言的流程控制方法。

(3)试比较列表、元组及字典之间的区别。

(4)常见的异常有哪些类型？为什么需要异常处理机制？

2. 实践题

(1)使用 print 函数在命令行窗口中输出“我爱学 Python!”。

(2)在 IDEL 中创建一个名为 friend. py 的文件,然后在文件中定义一个字符串,内容为“@比尔盖茨@沃伦巴菲特@阿里巴巴董事长马云”,通过字符串分割获取各个名称,并进行输出。

(3)在 IDLE 中创建一个名为 students. py 的文件,在文件中利用字典存储同学的姓名、年龄、性别等信息,并实现根据用户输入的姓名进行信息检索并输出。

(4)创建圆形模块,对应的文件名为 circular. py,在文件中定义两个函数:一个用于计算圆形的周长,另一个用于计算圆形的面积。