第5章

程序设计入门

任务1 了解程序设计理念

主 编 | 傅连仲 等

目 录

Contents

了解程序设计理念

程序设计的理念是程序设计的基础，程序是解决某个问题所需的一系列指令序列集合，程序设计语言是人们与计算机进行沟通的工具。运用程序设计解决问题的过程和方法是程序设计理念中最重要的部分，它是一种逻辑思维理念，不仅体现在程序设计中，也可以迁移运用到其他问题的解决中。算法是求解问题的一系列计算步骤，这些计算步骤可能是顺序执行、选择执行或循环执行的，这也正是一个程序中常出现的三种基本结构。

任务情景

情景1：在开始之前，让我们先来玩一个游戏—盲人指路。请同学两两组队，其中一人需戴上眼罩扮演盲人，另外一人需用语言指挥同伴绕过障碍物到达终点。比比看哪支队伍最先到达终点。

情景2：数学课上，老师讲解概率的含义，为了让同学们更好地理解，老师拿出了一枚硬币，让同学们抛100次这枚硬币，并记录下每次硬币落地后的正反面，然后统计出100次中抛出正面的概率。小华心想：抛100次硬币太费时，听说计算机是个运算速度特别快的家伙，它可以帮忙“抛”硬币吗?

任务分析

情景1：在“盲人指路”的游戏中，负责指路的同学所发出的一系列指令就是一个“程序”，比如“前进”“左转”“右转”“停”等，这些指令所组成的序列最终让“盲人”同学顺利到达终点。通过“盲人指路”的游戏，模拟了一个最简单的“程序”。计算机程序设计就是让计算机按照一定步骤去解决某个问题或者完成某项任务。

情景2：计算机自诞生之日起就是以其超强的“计算”能力而著称。我国自主研制的神威·太湖之光超级计算机中安装了40960个“申威26010”众核处理器，每秒能运行九亿亿次。所以，即使让计算机“抛”1000次硬币，它也可以在不到1秒的时间内完成。不过，如何让计算机完成“抛”硬币的过程呢？我们需要先利用计算思维，将现实生活中的问题转化成计算机所能处理的形式，然后设计算法并编写程序来实现。

说一说

请结合生活经验，谈一谈对程序设计的理解。

5.1.1 了解程序设计基础知识

1．指令和程序

指令（Instruction）是给计算机下达的一个基本命令，它是一条语句或代码。例如“在输出窗口打印出hello world！”是一条指令；“计算20除以4的商”也是一条指令。

程序（Program）是为实现特定目标的一条或多条编程指令序列的集合。在“盲人指路”游戏中，指挥“盲人”从起点到终点所发出的一系列指令序列（例如：前进2步-左转-前进3步-右转-前进1步……）是一个程序；抛硬币并计算抛出正面的次数占比的过程也是一个程序。

事实上，生活中很多事情都有程序：一份菜谱里记录着这道菜的制作程序；一本活动策划书里记录着某个活动的流程；早上起床洗脸、刷牙、吃早餐的过程是一个程序；制作板凳时的打眼、组装、打磨也是一个程序……

当计算机运行一个程序时，程序中的指令就会被连续自动执行，就像我们获得一份如图5-2所示的菜谱之后，能自动按着菜谱中的操作步骤做出双面煎蛋一样，对于计算机来说，根据人设定好的程序自动完成一系列指令，叫作“自动化”。今天我们能很方便地使用一些计算机软件或App，是因为程序员编写了程序来告诉设备应该怎样做。

——双面煎蛋的做法——

1．开火热锅
2．锅热了改小火
3．放一点油，晃、擦…让油散开
4．撒盐
5．磕入鸡蛋，盖锅盖，煎 1 分半
6．翻面，盖锅盖，再煎 1 分半
7．出锅

2．程序设计

计算机是一个没有生命的机器，是一个不知道自己该做什么、但却十分愿意服从命令的机器。手机如果没有“程序”，就是一堆没有用的零件，我们无法用它通话、上网和玩游戏。程序设计（Program Design）就是将问题解决的方法步骤编写成计算机可执行的程序的过程。简单来说，就是告诉计算机要做什么，并且每一个行为的细节和顺序都要说清楚、可执行。这样，计算机就能够很快速地、正确地完成所有“指令”，最终解决问题或完成任务。

说一说

请说一说什么是程序员思维?

5.1.2 了解常见的程序设计语言

1．低级语言和高级语言

我们和计算机沟通的语言就是程序设计语言，程序设计语言包括低级语言和高级语言。最开始的程序设计语言只有两个符号，要么是1，要么是0，它们分别代表电路“开”和“关”，这是一种比较底层的语言，称为二进制语言，又称为机器语言。虽然它能够实现我们与计算机的沟通，但是面对一大串毫无可读性的01代码，人们显然非常希望能够找到一种更加简便的方法来告诉计算机要做什么。为了降低程序编写和维护的难度，人们又发明了汇编语言，利用特定的助记符来帮助程序员记忆机器指令。但是，利用汇编语言编写的程序通常不能是大规模的，它和机器语言一样，都是直接面向机器的，与人们使用的自然语言有很大区别，机器语言和汇编语言统称为低级语言。后来，随着计算机语言的发展，高级语言终于诞生了。

高级语言是以人们的日常语言为基础的一种编程语言，是能够直接表达运算操作和逻辑关系的语言，大大增强了程序代码的可读性和易维护性。例如，曾经我们想让计算机在输出窗口打印出“前进！”，写下的程序可能是无序的01代码；而如今我们想让计算机进行同样的操作，写下的程序就可以像表5-1右列这样，简洁且具有很强的可读性。

现在，人们已经发明了很多高级语言了，比如C、C++、Java、Python等，它们有着各自不同的语法和特点，而Python凭借着它明确、简单、可扩展性强等特点，逐渐成为世界上最受欢迎的程序设计语言之一。

2．常见的高级程序设计语言

C语言：C语言是一门通用计算机编程语言，功能丰富，使用灵活。同时，C语言还具有汇编语言的许多特点，比如能直接访问物理地址、进行位操作、直接对硬件进行操作等，因此，C语言也称为“中级语言”。C语言是编写应用软件、操作系统和编译程序的重要语言之一。

C++语言：C++语言是在C语言基础上开发的一门中级语言，既可以进行C语言的过程化程序设计，又可以进行面向对象的程序设计。C++的应用领域很广，是受广大程序员喜爱的编程语言之一。

Java语言：Java语言是一门面向对象的编程语言，不仅吸收了C++语言的各种优点，还删减了C++里难以理解的概念，功能强大，简单易用。Java可以编写桌面应用程序、Web应用程序、分布式系统和嵌入式系统应用程序等。

Python语言：Python语言是一种面向对象的解释型编程语言，语法简洁清晰，是完全面向对象的语言，函数、模块、数字、字符串都是对象。Python拥有强大的标准模块和第三方模块，能够快速开发出功能丰富的应用程序。此外，Python常被称为胶水语言，能够把用其他语言（如C和C++）制作的模块轻松联结在一起。常见的一种应用情形是，使用Python搭建程序框架，若对其中有特别要求的部分，可用更适合的语言改写，比如3D游戏中的图形渲染模块性能要求很高，就可以用C或C++重写，而后封装为Python可调用的扩展模块就可以了。

5.1.3 理解用程序设计解决问题的逻辑思维理念

利用程序设计解决问题的过程和我们人类解决问题的过程有很大的相似之处。比如，当我们解决问题时，首先会观察、分析问题，收集必要的信息，然后根据已有的知识、经验进行判断和推理，接着尝试按照一定的方法和步骤去解决问题。而要通过程序设计来解决问题，也需要经历类似的思维过程，我们将这种运用信息技术解决问题的思想方法称为计算思维。计算思维让我们能够：

① 运用所学知识和技能，通过界定问题、抽象特征、建立模型和组织数据等，将一个抽象的问题转化成计算机等信息技术可以处理的形式；

② 通过判断、分析和综合各种信息，运用信息技术工具和信息资源，设计算法形成解决问题的方案；

③ 总结信息技术应用的方法和技巧，并迁移到与之类似的相关问题的解决过程中，包括自己的职业岗位和生活情境。

计算思维不仅体现在程序设计中，在我们的学习、生活和工作中，计算思维也同样重要，它能帮助我们去发现问题、分析问题和解决问题，是一项重要的思维能力。

5.1.3 理解用程序设计解决问题的逻辑思维理念

利用程序设计解决问题的过程和我们人类解决问题的过程有很大的相似之处。比如，当我们解决问题时，首先会观察、分析问题，收集必要的信息，然后根据已有的知识、经验进行判断和推理，接着尝试按照一定的方法和步骤去解决问题。而要通过程序设计来解决问题，也需要经历类似的思维过程，我们将这种运用信息技术解决问题的思想方法称为计算思维。计算思维让我们能够：

① 运用所学知识和技能，通过界定问题、抽象特征、建立模型和组织数据等，将一个抽象的问题转化成计算机等信息技术可以处理的形式；

② 通过判断、分析和综合各种信息，运用信息技术工具和信息资源，设计算法形成解决问题的方案；

③ 总结信息技术应用的方法和技巧，并迁移到与之类似的相关问题的解决过程中，包括自己的职业岗位和生活情境。

计算思维不仅体现在程序设计中，在我们的学习、生活和工作中，计算思维也同样重要，它能帮助我们去发现问题、分析问题和解决问题，是一项重要的思维能力。

1．将抽象问题转化成计算机能处理的形式

将一个问题转化成计算机能处理的形式，首先需要抽象出问题中的关键对象和对象之间的关系，然后建立起合适的模型，并用计算机语言表达出来。简单来说，这是一个对问题进行重新表述的过程。问题的类型千千万万，其表述方式并不唯一，有的问题可以用数学模型来表述，有的问题可以用文字、表格或图形等形式表述。

每次抛硬币，落地后要么是正面，要么是反面，这便是硬币落地后的两种状态。在计算机中，我们可以用两个数字来表示两种不同的状态，这种方法也可称为“编码”。例如，我们用数字1表示抛出正面，用数字0表示抛出反面（当然，你也可以用其他数字或其他计算机能处理的形式分别表示正面和反面状态）。

抛硬币的结果具有随机性，每次可能出现正面，也可能出现反面，就好比“抽签”一样。在Python程序设计语言中，提供了一个用于“抽签”的工具箱—random随机数模块，其中提供了一些用于产生随机数的“工具”。我们将“工具箱”random模块导入程序，就可以使用其中的所有“工具”了。例如，randint(a,b)是random模块中的一个“工具”，用于从a～b中随机产生一个整数，因此，“抛硬币”的过程可以用下面这行语句进行表达：

操作过程Python语句抛硬币 result = random.randint(0,1)

注：result代表抛硬币的结果，result = random.randint(0,1)表示0～1中随机产生一个数，并把这个随机数赋给result。result的值要么是0，要么是1。

当result的值是0时，代表抛硬币的结果是反面；当result的值是1时，代表抛硬币的结果是正面。在Python中，可以用关系运算符“==”来表达两个对象之间的相等关系，因此，抛硬币的结果可以表述为Python逻辑表达式。

这样，我们就通过编码、程序语句和逻辑表达式，将“抛硬币”问题用一种计算机能够处理的形式进行了重新表述。接下来，需要着手设计具体的解决方案，即设计算法。

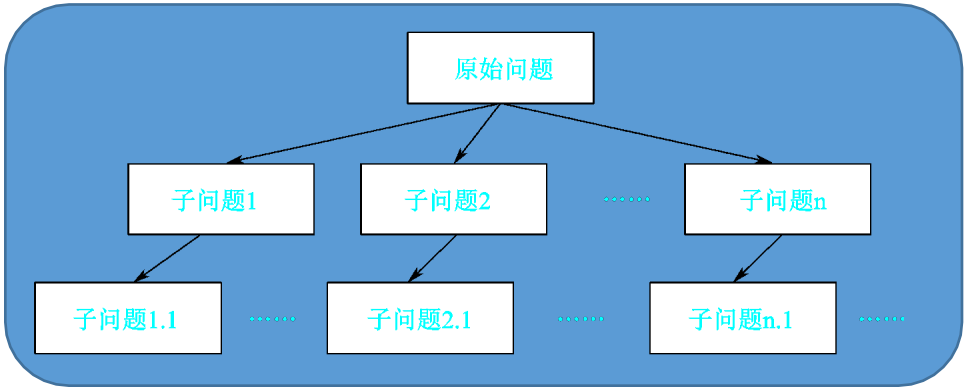
抛硬币的结果	Python 逻辑表达式
抛出正面	**result == 1**
抛出反面	**result == 0**

2．设计算法

在程序设计中，算法（Algorithm）就是程序执行的流程，是解决问题的步骤。

（1）问题分解。

对于较为复杂的问题，可以首先根据功能、流程或从其他角度将问题分解，并且分解出的子问题也可以根据需要进一步分解；之后，再对每个子问题设计详细的解决步骤，各个击破。

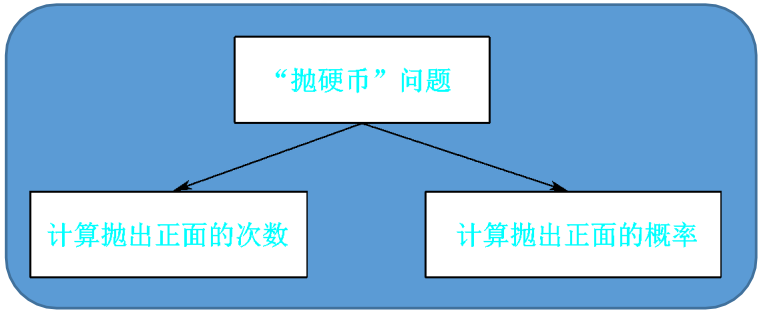


思维拓展：事实上，在很多时候，问题分解都能帮助我们更好地找到解决问题的办法。例如，作家写一本书之前，会先确定大纲，列出一级标题、二级标题……之后再对各个部分进行具体写作；做一项旅行攻略时，我们会将攻略分为景点、交通、住宿、用餐等几个模块，然后再针对各个模块进行详细规划。

抛硬币抛出正面的概率可以根据以下公式进行计算：

抛出正面的概率=抛出正面的次数÷实验总次数

已知：实验总次数为100次。于是，可根据计算公式将问题分解为两个子问题："计算抛出正面的次数"和"计算抛出正面的概率"，如图所示。当然，也可以从其他角度进行问题分解，或者也可以不分解。



（2）子问题1：计算抛出正面的次数。

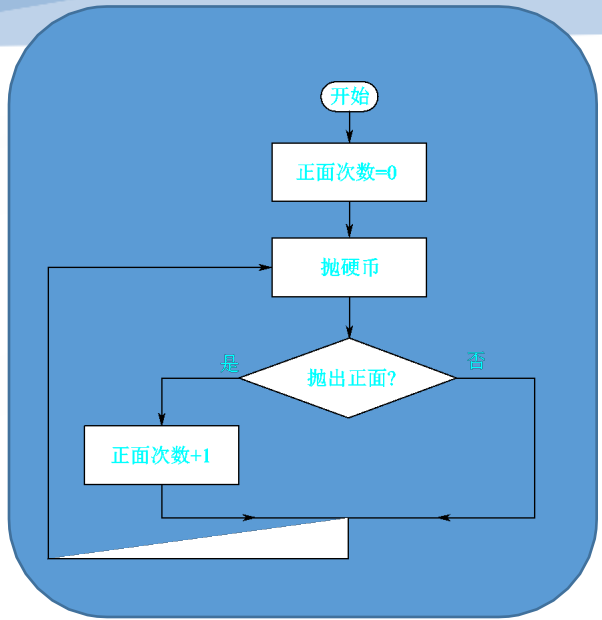
为计算抛出正面的次数，我们可以这样来设计算法，用自然语言描述为：

在开始“抛硬币”之前，将正面次数设为0。

执行“抛硬币”操作。

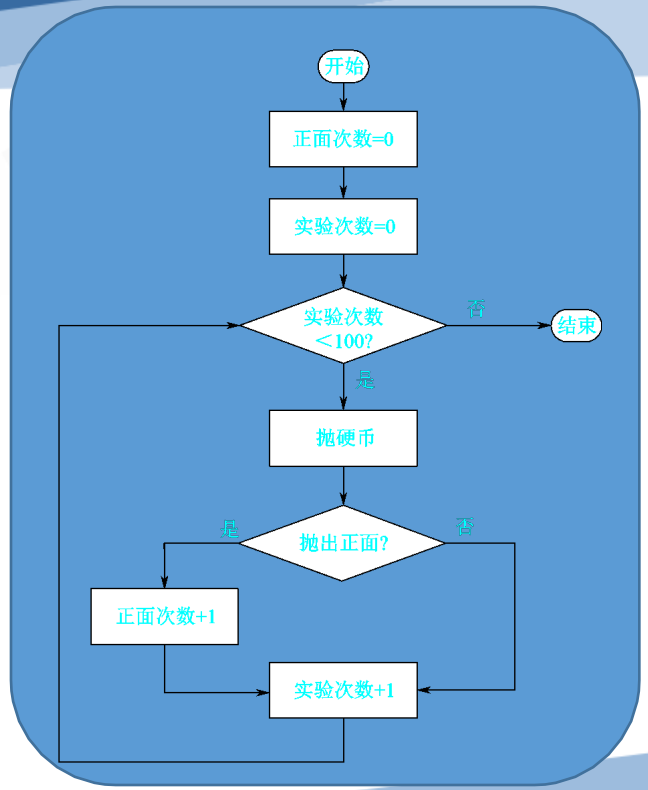
如果抛出正面，则正面次数+1，实验次数+1；如果抛出反面，则仅实验次数+1。

然后继续执行下一次“抛硬币”操作。



我们可以用流程图来描述这个算法，如图所示。其中，用圆角矩形表示“开始”，之后，程序将顺着箭头指引的方向进行；用菱形表示“判断”，在该判断条件下，有两条分支，一条是抛出正面之后要进行的操作（正面次数+1，然后进行下一次抛硬币），另一条是抛出反面之后要进行的操作（直接进行下一次抛硬币）。然而，仔细分析一下绘出的流程图，将发现这个流程并没有出口。

事实上，当抛完第100次硬币后，就可以不再继续抛硬币，进而结束流程。因此，需要在每次抛硬币之前，判断是否已经抛了100次，如果还没有抛够100次，就继续抛硬币；否则，就结束流程，进行下一个步骤。如果不判断实验次数是否达到100次，程序就会永不停止地“抛硬币”，陷入“死循环”。此外，每抛一次硬币，都应更新实验次数，以记录当前是第几次抛硬币。这样，我们可以在之前算法的基础上，增加对实验次数的判断，用自然语言描述新的算法：



在开始抛硬币之前，将正面次数设为0，将实验次数设为0。

判断实验次数是否<100，如果是，则执行“抛硬币”操作。如果抛出正面，则正面次数+1，实验次数+1；如果抛出反面，则仅实验次数+1。

继续判断实验次数是否<100，如果是，则执行“抛硬币”操作。如果抛出正面，则正面次数+1，实验次数+1；如果抛出反面，则仅实验次数+1。

直到某次判断发现实验次数≥100，结束“抛硬币”操作。

说一说

请根据上面的流程图，描述一下计算机抛100次硬币的具体过程，包括正面次数和实验次数的变化情况。

（3）子问题2：计算抛出正面的概率。

经过了100次抛硬币之后，我们可以得到抛出正面的次数，接下来，就可以根据概率计算公式计算抛出正面的概率了。

3．反思和迁移

下载并运行下面的示例程序tossCoin.py，体会一下用计算机解决“抛硬币”问题的过程，看看抛100次硬币得到正面的概率是多少？当抛硬币的次数更多时，抛出正面的概率接近于哪个数？

```
import random up_n = 0 #记录抛出正面的次数total_
代表实验总次数cnt = 0 #记录实验次数 while cnt<total_n:
= random.randint(0,1)     print(result)    if resul
up_n = up_n +1    cnt = cnt + 1 p = up_n/total_npri
面的频率为：'+str(p))
```

随机产生0或1。0代表抛出反面，1代表抛出正面。若抛出正面，正面次数+1。每次实验后，cnt自增1，用以记录实验次数，当实验次数达到总次数后，就不再抛硬币了，结束循环，计算抛出正面的概率。

说一说

用计算机解决问题有什么优势？生活中还有什么问题可以用类似的方法解决？

4．算法、程序流程图和程序基本结构

（1）算法。

算法（Algorithm）是求解问题的一系列计算步骤，我们计算抛100次硬币抛出正面的概率所采用的计算步骤就是解决这个问题的一个算法。解决不同的问题可能需要不同的算法，同一个问题也可能有不同的解决方案或算法。算法是软件的核心，无论是解决简单问题的程序，还是制造芯片的软件，都依靠算法。对于一些经典的问题，人们提出了很多解决办法，并总结成了经典的算法，如枚举算法、二分查找法、排序算法、递归算法、回溯算法等。

一个算法应该具有以下几个重要特征：

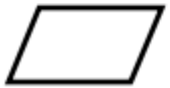
有穷性　确切性　输入项　输出项　可行性

程序是算法和数据结构的总和，其中，算法是程序的“灵魂”，数据结构是对数据的表达和处理。因此，算法独立于任何具体的程序设计语言之外，一个算法可以用多种程序设计语言来实现。我们可以用自然语言来描述一个算法，也可以用程序流程图来表示一个算法。

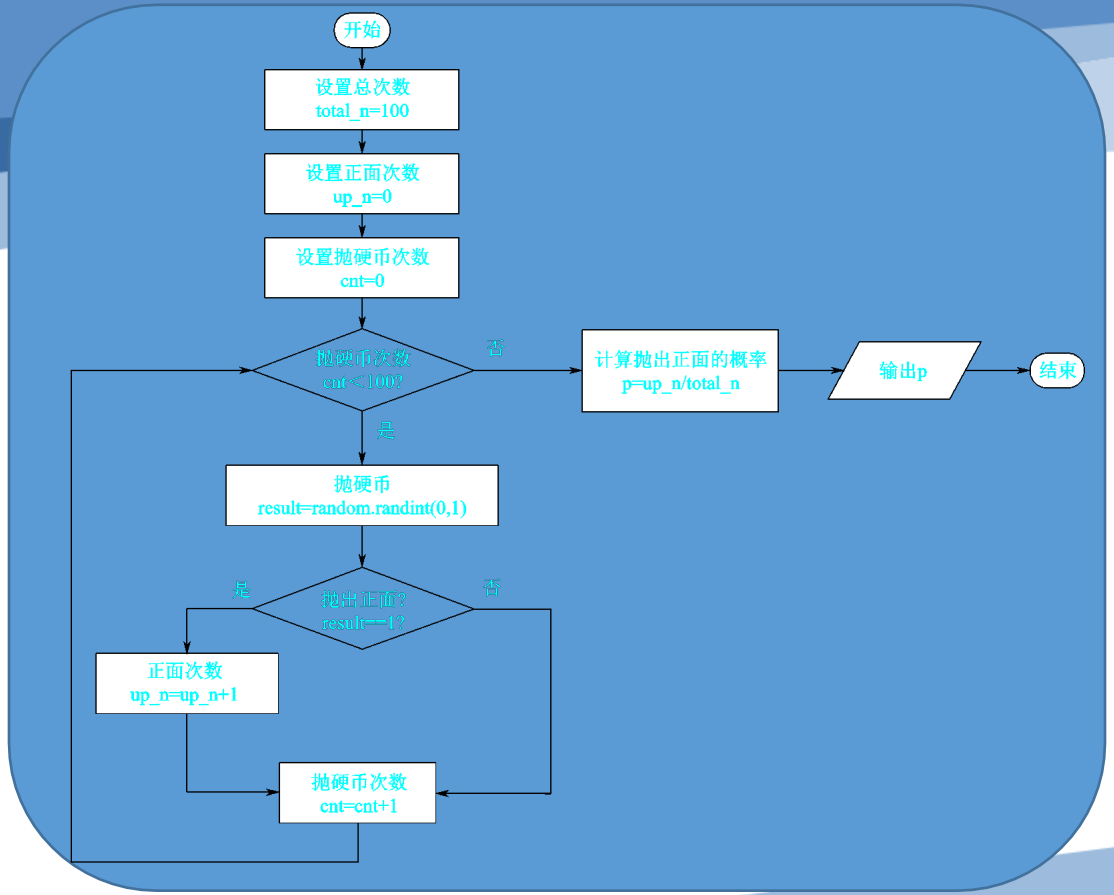
（2）程序流程图。

程序流程图是把计算机的主要运行步骤和顺序呈现出来的一种工具，是整个程序的一张蓝图，能够清晰直观地体现出程序的逻辑性和处理顺序。当然，这张蓝图并不唯一，对于同一个问题，按不同的算法就会画出不同的流程图。

为方便程序员对输入输出和数据处理过程进行分析，也便于程序员之间进行交流，程序流程图用统一规定的标准符号和图形来表示，通常包括处理框、判断框、输入输出框、起止框、连接点和流程线。

	**处理框：**具有处理功能，如数学计算、给变量赋值等		**流程线：**表示流程的路径和方向
	**判断框：**具有条件判断功能，有一个入口，二个出口		**连接点：**可将流程线连接起来
	**输入输出框：**获取用户输入的操作，或者计算机输出的操作		**起止框：**表示程序的开始或结束

“抛硬币”问题的流程图


开始
设置总次数
total_n=100
设置正面次数
up_n=0
设置抛硬币次数
cnt=0
抛硬币次数
cnt＜100?
否
计算抛出正面的概率
p=up_n/total_n
输出p
结束
是
抛硬币
result=random.randint(0,1)
是
抛出正面?
result==1?
否
正面次数
up_n=up_n+1
抛硬币次数
cnt=cnt+1

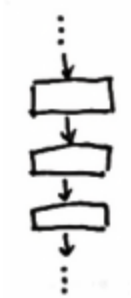
说一说

在上面的流程图中，每个步骤被执行的次数一样吗？有哪些步骤被多次执行了？

如果实验总次数变为1000次或更多，每个步骤被执行的次数会发生什么变化呢？

（3）程序基本结构

在程序设计中，不总是顺次执行每个步骤，有时需要在两个步骤中选择其中一个执行，有时需要连续多次执行某些步骤。程序中每个步骤的执行顺序构成了程序的结构，常见的程序结构包括顺序结构、选择结构和循环结构。



顺序结构

选择结构

循环结构

① 顺序结构。

从上往下，一步一步顺次往下执行。在选择结构和循环结构中也会有顺序结构。

② 选择结构。

在“抛硬币”程序中，判断是否抛了100次硬币、判断抛硬币结果是否是正面，并根据判断结果从接下来的步骤中选择其中一个执行，这便是选择结构。

③ 循环结构。

在“抛硬币”程序中，程序并不是从上到下地顺序执行，在100次抛硬币的过程中，每一次都需要先判断抛硬币的次数是否小于100，然后给result随机赋值为数字0或数字1，并判断正面次数是否加1等，这些步骤将被循环执行100次，这便是循环结构。

程序就像我们的人生一样，不会永远一帆风顺、顺序执行。有时我们会面临选择，有时可能会在一个地方原地打转，但也正是因为有了这些时刻，生命才更加精彩。在程序的世界里，也正因为有了选择和循环，程序设计也才更加有趣，更加便捷高效。

说一说

在“抛硬币”学习概率的活动中，每次抛硬币，硬币落地后有哪几种状态？在计算机中如何表示硬币的状态？如何体现“抛硬币”的随机性？

第5章

程序设计入门

任务2 设计简单程序

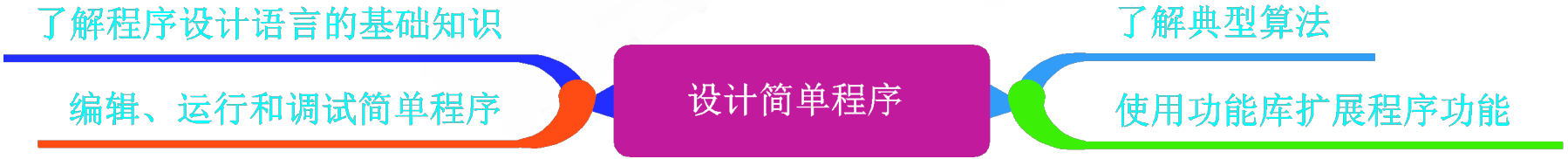
主 编 | 傅连仲 等

目 录

Contents

设计简单程序

设计程序是将解决问题的方案付诸实践的过程。而了解程序设计语言的基础知识是需要迈出的第一步，因为程序设计语言是我们与计算机进行沟通的重要工具。接下来，将学习如何编写、运行和调试简单程序，并了解典型算法和功能库的使用方法，编写程序来解决实际问题。在这个过程中，将不断体会运用程序设计解决问题的过程和方法，体会程序设计的理念。



任务情景

临近节日，小华发现很多商店都在打折，促销活动吸引了很多的顾客去购买商品。小华心想：虽然打折活动让每个商品的价格降低了，但是销售量也增多了，那么商家最后获得的利润是比平时更高了还是更低了呢？如果他将来也开一家店，到了打折季的时候，为了获得最高的利润，怎么决定打几折呢？怀着这些疑问，小华找到了堂兄。

任务分析

堂兄听了小华的疑问，说：“打折和给商品定价可是一门学问啊，根据对消费者消费心理等情况的了解，可以编写程序来计算打几折可以获得最高利润，还能够预测打折活动带来的具体利润呢。”

学习程序设计语言是与计算机进行沟通的基础，本节以Python语言作为编程工具，学习如何创建并运行程序，了解程序设计语言的基础知识，并设计程序来帮助小华解决打折问题。

5.2.1 了解程序设计语言的基础知识

1．Python开发环境IDLE

从Python的官网上下载并安装了Python之后，同时也就安装了IDLE（集成开发环境）—Python的官方标准开发环境。

IDLE集成了整个代码编辑时要用的东西，包括交互式Shell和编辑器。其中，交互式Shell相当于一个简化的编辑器，当只需要编写一些小的验证性代码，可以在Shell中编写代码并执行；但如果需要编写完整的Python程序，或者需要将代码保存并希望能够反复运行，就要使用编辑器了。

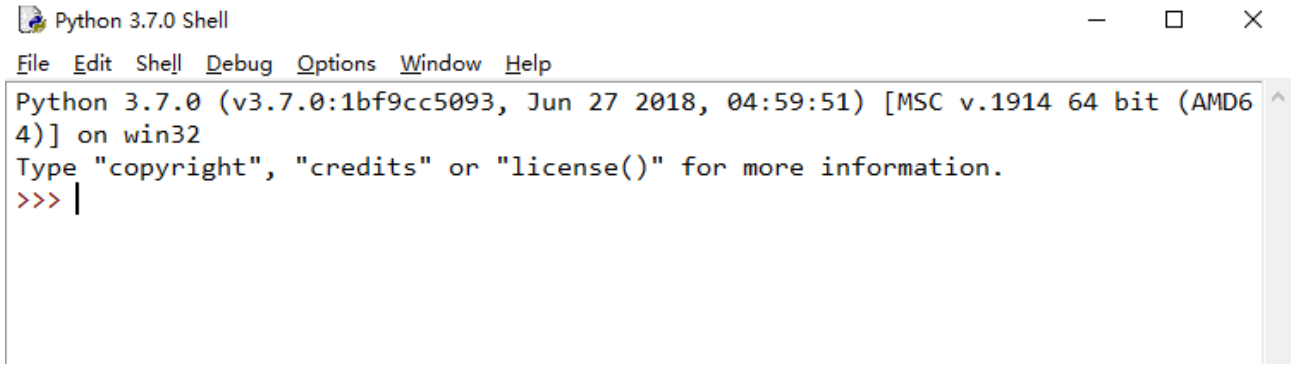
2．程序设计

计算机是一个没有生命的机器，是一个不知道自己该做什么、但却十分愿意服从命令的机器。手机如果没有“程序”，就是一堆没有用的零件，我们无法用它通话、上网和玩游戏。程序设计（Program Design）就是将问题解决的方法步骤编写成计算机可执行的程序的过程。简单来说，就是告诉计算机要做什么，并且每一个行为的细节和顺序都要说清楚、可执行。这样，计算机就能够很快速地、正确地完成所有“指令”，最终解决问题或完成任务。

（1）在Shell中输入并运行Python指令。

在Windows操作系统左下角的搜索框中，输入“IDLE”，找到IDLE应用程序，单击即可启动，出现Python的交互式Shell窗口，如图所示。

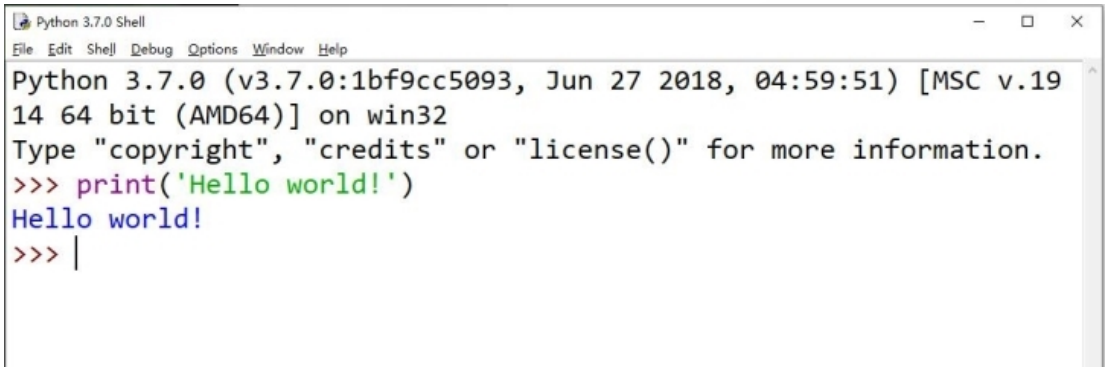
在“>>>”提示符后面，输入一条Python指令，回车，Python将执行该指令，并在下一行显示该指令执行的结果。指令执行完成后，将在下一行出现一个新的“>>>”提示符，等待下一条指令的输入。



例如，在>>>后输入一行Python语句：print('Hello world!')，并按下【Enter】键，Python将在下一行打印输出：“Hello world!”。

注意：代码中所有字符均为英文字符，包括引号和括号。并且，print()语句的前面没有空格，如果有多余的空格，Python执行指令时会报错，红色的SyntaxError是报告的错误类型。

Python语言以缩进控制语句的级别，就像编写文档时设置大纲级别为1级、2级、3级。在Python中，有相同缩进的一组连续语句属于同一逻辑层级的语句，在每行语句开头的空格或制表符就是缩进，通常用4个空格或1个制表符表示一个缩进。因此，在编写Python程序中，要严格控制每行语句开头的缩进。



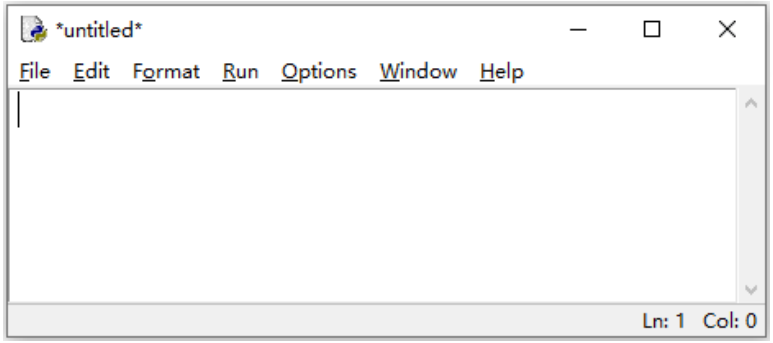

```
>>> print('Hello world!')

SyntaxError: unexpected indent
>>>
```

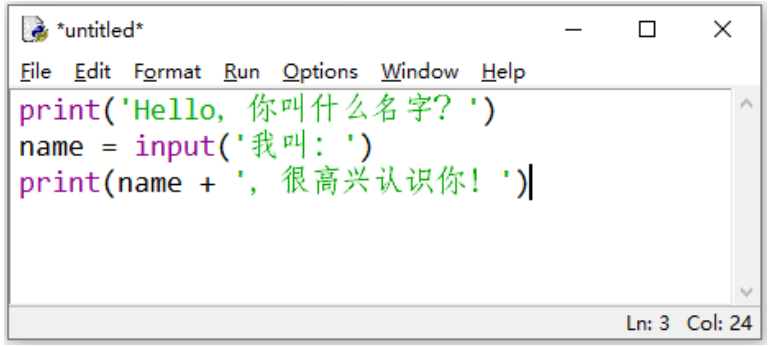
（2）在IDLE中创建并运行Python程序。

在IDLE的交互式Shell中，虽然能方便快速地执行Python指令，但每次只能输入一行代码、执行一条指令，不能连续执行多条指令。因此，我们需要一个新的方式来执行一连串的Python指令—程序。

① 第一步：创建程序。

启动IDLE，单击File→New File，弹出IDLE的文件编辑器窗口。接着，请在编辑器窗口中输入3行Python语句。

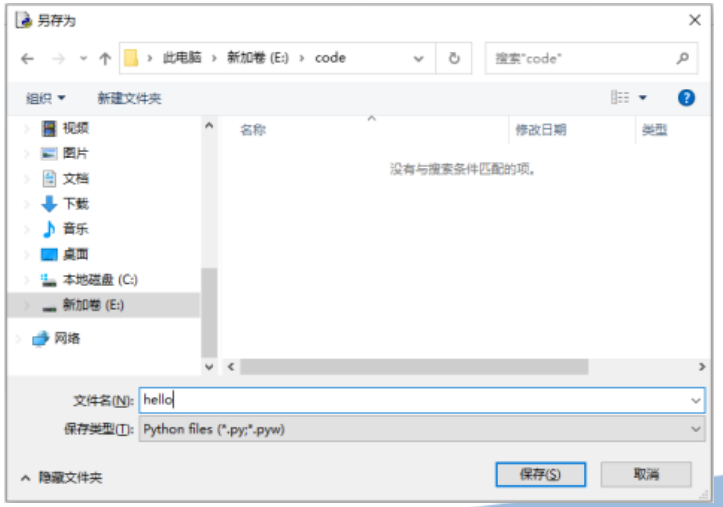




② 第二步：保存程序。

按【Ctrl+S】组合键或者单击File→Save as保存源代码文件，弹出另存为窗口，在文件名文本框中输入文件名，如“hello”，保存类型选择Python files，然后单击保存按钮。保存成功后，即可在保存的地方找到刚刚创建的Python程序文件hello.py。

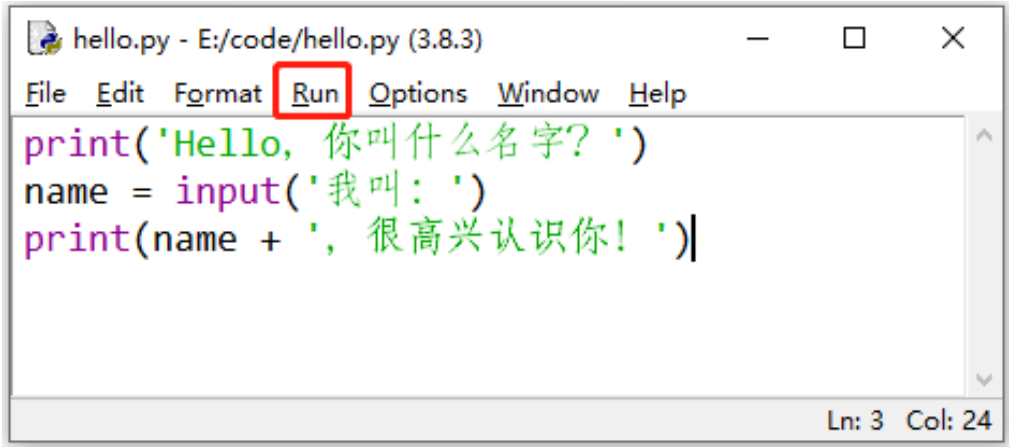
Python程序文件也叫Python可执行文件，它能够存储多条Python指令序列，是一个后缀名为.py的文件，运行它时，其中的指令可以被连续执行。



③ 第三步：运行程序。

单击Python编辑器菜单栏的Run→Run Module或者按【F5】快捷键即可运行程序，界面如图所示。

运行之后，将首先在IDLE的Shell面板输出一行文字“Hello，你叫什么名字？”，然后在第二行输出“我叫：”，同时光标闪烁，等待用户输入。此时，小华通过键盘输入了他的名字“小华”，并按下【Enter】键，之后，程序继续在第三行输出：“小华，很高兴认识你！”。



④ 第四步：再次打开保存的程序。

将一连串程序指令保存成一个Python程序文件，不仅可以让其中的指令连续执行，还可以将其保存，当下次需要其进行查看、编辑和修改时，再次打开保存的程序即可，打开程序的方式有以下两种：

方式一：从IDLE中打开保存的程序文件。单击File→Open，在所出现的窗口中选择文件，并且单击打开按钮。刚刚保存的hello.py程序将会在文件编辑窗口中打开。

方式二：直接打开程序文件。找到要弹出的Python程序文件并单击鼠标右键选择Edit with IDLE。

说一说

现在，请运行程序，输入你的名字，完成与计算机的第一次交流。

2．数据类型和表达式

在程序设计中，将现实生活中的问题转化成计算机能够处理的形式是利用计算机解决问题的关键步骤，而数据和表达式就是对问题进行重新表述的关键。

（1）数据类型。

数据是对信息的刻画，不同的数据类型可以表达不同的信息。在Python中，常见的数据类型见表。

数 据 类 型	描　　述
整数型	数学中的整数，包括正整数、负整数和0。如1，-21，0等
浮点型	数学中的小数，小数点是它的标志。如3.14，-2.0等
字符串型	用单引号、双引号或三引号表示。如'a'、"Hello"，'''你好！'''等
布尔型	只有两种值：True和False，分别表示逻辑的“真”和“假”

① 数据类型的转化。

不同的数据类型之间存在差异，不能进行相互运算。因此，必要的时候，需要对数据类型进行转化。在Python中，内置函数str()、int()、float()可以分别将数据转化成字符串型、整数型和浮点型，其作用及示例见表。

转化函数	作　用	示　例
str()	转化成字符串	str(123)　　转化结果：'123'
int()	将数字或长得像整数的字符串转化成整数	int(1.5)　　转化结果：1int('100')　　转化结果：100int('1.5')　　转化失败！！
float()	将数字或长得像浮点数的字符串转化成浮点数	float(2)　　转化结果：2.0float('2.5')　　转化结果：2.5float('hello')　　转化失败！！

例如，字符串和字符串之间可以通过“+”连接运算符，将两个字符串连接成一个字符串：

```
>>> 'Hello ' + '2021'
'Hello 2021'
```

数字和数字之间也可以通过加法运算符“+”计算两个数字的和：

```
>>> 10 + 2021
2031
```

但字符串和数字之间却不可以进行“+”加法运算，在下面的反例中，可以从TypeError看出错误的原因：

```
>>> 'Hello ' + 2021Traceback (most recent call last):  File "<pyshell#5>", line 1, in <module> 'Hello ' + 2021TypeError: can only concatenate str (not "int") to str
```

```
>>> '10' + 2021Traceback (most recent call last): File "<pyshell#7>", line 1, in <module>    '10' + 2021TypeError: can only concatenate str (not "int") to str
```

这时，我们可以根据需要对数据类型进行转化：

```
>>> 'Hello ' + str(2021)'Hello 2021'
```

```
>>> int('10') + 20212031
```

② 组合数据类型。

除了以上几种常见的数据类型，还有一种组合数据类型，可以将一组数据统一存储和管理，以便于程序对一组数据进行批量处理。例如，遍历一组水果的英文单词、找到运动会报名了跑步和跳远的同学、登记一个学生的各项信息……Python中，字符串实际上也是一种组合数据类型，每个字符都是它的元素。除此之外，常见的组合数据类型见表。

组合数据类型	描　述
列表	用方括号[]创建，其中的元素有序存放，索引号从0开始，且可修改、增添、删除元素。如[1,1,3]，['apple','banana']等
元组	用小括号()创建，其中的元素有序存放，索引号从0开始，不可修改、增添、删除元素。如(1,1,3)，('apple','banana')等
集合	用花括号{}创建，其中的元素无序存放，无索引号，但不能重复。如{1,2,3}，{'小华','大华','天歌'}，{'小华','清波'}
字典	用花括号{}创建，其中每个元素都是一个键值对（key=>value），具有映射关系，无序存放，可通过键获取对应的值。如{'name':'小华', 'age':15}

例如，将一组水果单词存放在fruit列表中，表示用户喜欢的水果。开始用户将第一喜欢的水果变成了桃子（peach），接着用户又新增了一个喜欢的水果草莓（strawberry），最后用户又不喜欢梨（pear）了。用程序表示这个过程：

```
>>> fruit = ['apple', 'banana', 'pear', 'grape']

>>> fruit[0] = 'peach'
>>> fruit.append('strawberry')
>>> fruit.remove('pear')

>>> fruit
['peach', 'banana', 'grape', 'strawberry']
```

修改第一个元素为'peach'

在末尾添加一个元素'strawberry'

移除值为'pear'的元素

（2）表达式。

计算机不仅能进行数学运算，还能进行逻辑运算，对应的Python中表达式也有算术表达式和逻辑表达式。

① 算术表达式。

在Python中，“算术表达式”就是数学中的“算式”，1+2、2-5都是算术表达式。它由算术运算符和操作数组成，例如，1+2中，1和2都是操作数，“+”是算术运算符。

✓ 算术表达式的值。

算术表达式的值就是算式的计算结果，比如：算术表达式1+2的值是3。另外，单个数字也可以看作一个特殊的算术表达式，其值就是这个数本身。在Python的交互式Shell中输入一个算术表达式，按【Enter】键后即可得到该算术表达式的值。例如，计算表达式1+2的值：

```
>>> 1+2
3
```

除了“+”，Python中还有其他一些常用的算术运算符。

算术运算符	算术表达式示例	描　述	值
+	1+2	1加2	3
-	1-2	1减2	-1
*	1*2	1乘以2	2
/	10/4	10除以4	2.5
//	10//4	10整除4（10除以4，取商的整数部分）	2
%	10%4	10除以4，取余数	2

② 逻辑表达式。

算术表达式用以表达数字之间的计算，逻辑表达式则通常用以表达对象之间的关系，例如大小关系、包含关系等。

✓ 逻辑表达式的值。

逻辑表达式的值只有两种，分别是True和False，表示这个逻辑是否成立，即表示这件事情的真和假。通常用逻辑表达式进行逻辑判断，若逻辑成立，则逻辑值为True；否则，逻辑值为False。在Python的交互式Shell中输入一个逻辑表达式，按【Enter】键后可得到该逻辑表达式的值。例如，1>2是一个“假”命题，因此逻辑表达式1>2的值为False：

```
>>> 1>2False
```

✓ Python关系运算符。

除了表示“大于”的关系运算符“>”，Python中还有其他一些常用的关系运算符，见表。

关系运算符	逻辑表达式示例	描　述	值
>	1>2	1大于2	False
<	1<2	1小于2	True
==	1==2	1等于2	False
>=	1>=2	1大于等于2	False
<=	1<=2	1小于等于2	True
!=	1!=2	1不等于2	True
in	'e' in 'hello'	字符串'e'包含在字符串'hello'中	True

✓ Python逻辑运算符。

对于一些更复杂的逻辑，例如“并且”“或”“非”这样的复合逻辑，可以用逻辑运算符来表达。在Python中，逻辑运算符为and（并且）、or（或）、not（非）。

在复合逻辑中，表达式的值是True或False，不仅取决于表达式中各个条件的真假，还取决于连接这些条件的逻辑运算符是什么，具体分析见表。

	A和B均为True	A为True，B为False	A为False，B为True	A和B均为False
A and B	True	False	False	False
A or B	True	True	True	False
not A	False	False	True	True

当条件A和条件B同时满足，表达式“A and B”的值为True；当条件A和条件B中有一个条件不满足，表达式“A and B”的值为False。例如，10<20<30是一个复合逻辑：10<20且20<30，由于10<20和20<30都成立，因此10<20<30的逻辑值为True：

```
>>> 10<20 and 20<30
True
```

3．变量和赋值

在Python程序中，为了让计算机“记住”某个信息，可以通过创建“变量”，将信息保存在计算机里一个负责“记忆”的地方—内存。

（1）变量：对象的名字。

变量指向内存中某个数据对象存储的位置，我们可以把变量看作对象的名字或代号。例如，在hello.py程序中，将用户输入的名字信息存储在了变量name中，变量name就代表着用户输入的名字。

```
>name = input('我叫: ')
```

变量命名规则一般有以下4条：

① 变量名必须以字母或下画线“_”开头；

② 变量名中其他字符必须是字母、数字或者下画线“_”，这意味着一个名字中不能用空格（例如：my name就不是一个合法的变量名，因为中间有空格）；

③ 变量名区分大小写，abc和ABC是两个不同的变量；

④ 变量不能与Python的关键字、内置函数名、内置模块名等重名，如不可给变量取名为int、print等。

通常，一个有意义的名字会比一个没有意义的名字更受青睐。当变量名字的含义较复杂时，常采用驼峰命名法和下画线命名法进行命名。

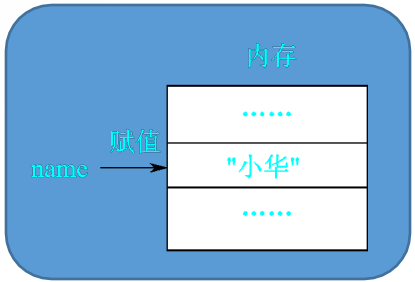
例如，变量名userName、printNum采用的是驼峰命名法，变量名user_name、print_num采用的是下画线命名法。

（2）赋值：创建变量的过程。

“赋值”是创建变量的过程，当需要程序记住某个信息（或对象）的时候，就在内存中开辟一块地方，把这个对象放在里面，同时，给这个对象取一个名字，并让这个名字指向对象所在的位置。在Python中，一个变量只有被赋予了一个具体的值才会被创建，即只有变量名和内存中的某个对象建立起联系后才算赋值成功。

在Python中，赋值操作是通过赋值操作符“=”来完成的。这里的“=”不是数学上的“等于”，而是一个赋值符，其作用是把右边的内容赋值给左边的变量，使对象和变量名建立起对应联系。

例如，name = "小华"，将字符串"小华"赋给了变量name，name将指向内存中字符串"小华 "存储的位置，字符串"小华"就是变量name的值，如图所示。当然，如果用户输入了其他内容，name将被赋为其他值，并指向对应对象所在的存储位置。



（3）变量类型。

在Python中，变量可以被赋予为不同类型的值，而被赋予了不同类型值的变量，是不同类型的变量。例如，在下面的程序中，变量name的值是字符串' 小华'，因此变量name是字符串变量；变量age的值是整数15，因此变量age是整数变量。

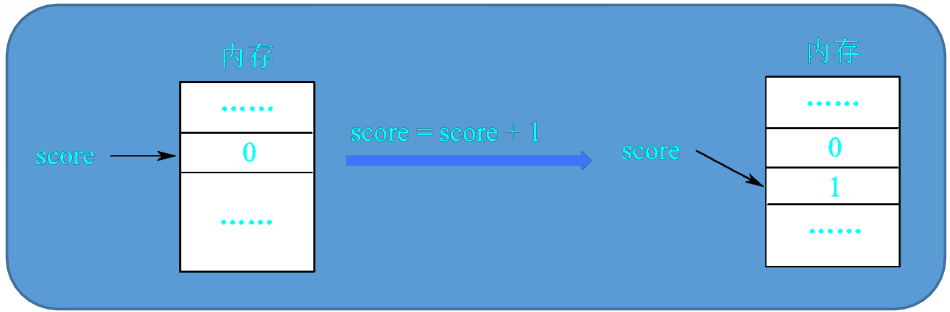
```
name = '小华'        age = 15
```

（4）访问变量的值。

赋值操作完成之后，变量就代表内存中存储的该数据对象，因此，可以通过变量名来访问具体的值。例如，在hello.py的第3行代码中，通过变量name输出用户的名字。

在程序中，变量的值可以改变。当一个变量被赋予了一个新值，它就会指向新值所在的位置。例如，创建变量score来记录玩家得分，初始化score为0，每次游戏获胜时，score被重新赋值为score+1，程序将先计算出右边score+1的和为1，然后将score变量指向内存中存储数字1的地方，而不再指向内存中存储数字0的地方，如图所示。

```
print('Hello，你叫什么名字？')name = input
('我叫：')print(name + '，很高兴认识你!
')
```



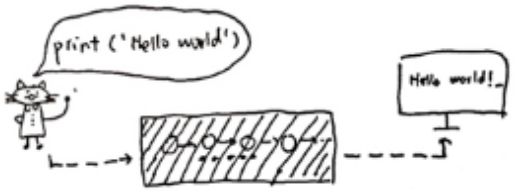
4．函数和模块

在Python中，函数和模块都可以看作Python的“工具”，它们让程序设计变得更加简单和方便。

（1）函数。

① 函数的概念。

在hello.py程序中，print()和input()都是Python的函数，分别用于打印输出和键盘输入。函数是将一系列复杂的操作或一系列连续的指令打包，“封装”成一条指令，这样，在程序的其他地方，就可以根据需要随时调用这条函数指令。就像我们去餐厅吃饭，顾客只需要“点菜”，厨师就会做出美味佳肴，而顾客自己不需要亲自去执行做菜的每个步骤。



要在屏幕上打印出一行文本，计算机其实需要进行很多复杂的操作，但是由于这是一个很常用的功能，因此Python的设计者便将所有用于打印输出的底层指令“封装”起来，并将其命名为print，这样，我们只需调用一条指令—print()，就能自动调用函数中被封装的底层代码了，如图所示为调用函数示意图。

② 函数参数。

为了让函数的使用更具有灵活性，我们可以向函数传递参数。函数的参数就像做菜时加的调料，加入不同的调料，就会烹饪出不一样的味道。在数学课上，f(x)=x+1也是一个函数，x的值不同，得到的f(x)就不同。

比如，print()函数是一个带参数的函数，其功能是打印输出一行文本并自动换行，括号里的参数可以为空，也可以是一个文本或一个数字，表示要打印输出的内容。

③ 函数的返回值。

✓ 没有返回值的函数

没有返回值的函数只需完成一个或多个操作，完成操作后不必向调用它的地方返回任何数据。例如，print()函数是一个没有返回值的函数，它只需要完成“输出”操作即可。

✓ 有返回值的函数

有返回值的函数执行完成一系列操作后，会将函数的执行结果返回到调用它的地方。例如，input()函数是一个有返回值的函数，它在获取到用户输入的数据之后，会将输入内容以字符串形式返回到调用它的地方。因此，有返回值的函数也可以被看作是一个数据对象，可以在程序中进行赋值和运算操作。在hello.py程序中，第二行代码就是将用户输入的“名字”赋给name变量。

```
name = input("我叫: ")
```

④ 自定义函数。

print()、int()等是Python的设计者已经定义好的函数，称为“内置函数”。在Python中，也可以根据需要自己定义函数，并在程序中调用它们，自己定义的函数是“自定义函数”。

- ✓ def是函数定义的关键字，取自英文单词definition（定义）；
- ✓ 函数定义时需指明函数的名字，好比为一道菜取一个名字；
- ✓ 括号中可以设置函数的参数，多个参数用逗号“，”隔开；
- ✓ 冒号“:”表示即将开始定义函数内部的语句；
- ✓ 封装在函数里的代码有相同的缩进，表示它们属于这个函数，是同一个语句块。语句块是具有相同缩进的一组连续语句。在Python中，用“：”标志一个语句块的开始，在其他程序设计语言中，使用特殊单词（如Begin和End）或字符（如“{”和“}”）标志一个语句块的开始和结束。

```
def 函数名(参数1,参数2……):
    语句1
    语句2
    ……
```

函数调用

和调用Python的内置函数一样，自定义函数也通过函数名和括号来调用，函数名需与函数定义时的函数名保持一致。如果定义了参数，可以在调用时传入参数。

```
函数名(参数1,参数2, ……)
```

例如，自定义一个“做牛肉面”的函数makeNoodles()，并设置辣度参数spicy，然后调用该函数，传入辣度选项为“特辣”，输出做一碗“特辣”牛肉面的步骤。

运行输出：

noodles.py

```
01.def makeNoodles(spicy):
02.    print('开始制作牛肉面')
03.    print('烧水')
04.    print('煮面')
05.    print('配味：' + spicy)
06.    print('加汤')
07.    print('加牛肉')
08.    print('牛肉面做好了！')
09.makeNoodles ("特辣")
```

函数定义：教“厨师”做牛肉面的方法步骤。

运行输出：开始制作牛肉面
烧水煮面配味：特辣加汤加牛肉
牛肉面做好了！

上面的程序中，如果只定义makeNoodles()函数而不调用它，即没有第9行代码，程序还会执行函数里的内容，将做牛肉面的步骤打印出来吗？为什么？

“函数定义”只是教会了厨师做面的方法，“函数调用”才是下达“做面”指令，因此，若程序定义了函数而没有调用函数，函数里的代码也不会被执行。另外，由于程序是从上到下执行的，当它看到函数定义时，才会将其记录下来，完成函数定义的“注册”。因此，Python中的函数定义必须放在函数调用之前，就像让厨师做面之前，得先教会厨师怎么做面。

⑤ 函数的妙用

✓ 避免代码冗余

函数是对代码的一种封装，我们只需要在函数中写一次代码，就可以对这些代码进行重复利用，非常方便。例如，在noodles.py中，将做面的步骤封装到makeNoodles()函数中，之后每次需要“做面”时，只需调用函数，就可以执行所有“做面”的步骤。

✓ 方便程序修改

当代码需要修改时，只需在函数中修改一次，而不用在每一个需要做这件事的地方修改代码。例如，当“做面”的步骤发生更改，或者增加了面的一些规格选项，如酸度、是否加香菜等，只需要在函数定义中修改一次即可。如果不使用函数，则要对每一次“做面”的程序进行修改。

实现模块化编程

函数将多行代码封装成一行语句，通过函数名能很容易地知道程序在做什么，增加程序的可读性。在更复杂的程序中，可将任务分解为几个子任务，若我们将每个子任务的实现代码封装成函数，将每个子任务看作一个“模块”，则可实现模块化编程。当程序有问题时，我们只需关注是哪个模块出现了问题，再进行深入排查，而不必在整个程序中进行“大海捞针”般的工作。

在设计较复杂的程序时，一般采用自上而下的方法，将问题划分为几个部分，各个部分再进行细化，直到分解为较好解决的问题为止。模块化编程，简单地说就是程序的编写不是一开始就逐条录入计算机语句和指令，而是首先用主程序、子程序（函数定义中的代码是子程序，函数调用的代码是主程序）等框架把程序的主要结构和流程描述出来，并定义好各个框架之间的输入输出关系。

就像“搭积木”一样，一块积木就是负责一个功能的小程序块，模块化编程就是将这些积木有组织地搭建起来。模块化的目的是降低程序的复杂度，使程序设计、调试和维护等操作简单化。

事实上，我们也常常用“模块化”的思想解决生活中或学习中的问题。比如我们会将自己的生活安排分为学习模块、锻炼模块、休息模块等，在不同模块中再进行不同的安排。模块化，能提高我们的学习和工作的效率。

（2）模块。

① 模块是什么?

在本章最开始的“抛硬币”问题中，random随机数模块就是Python中的一个模块。模块的本质是一个Python可执行文件，里面有许多已经定义好的功能相似的函数、变量、类及一些可执行代码。模块就好比一个工具箱，里面装着各种各样的工具，可供我们使用。例如，random模块这个“工具箱”里有很多与随机数相关的“工具”，randint(a,b)函数就是其中一个“工具”，用于获取a～b之间的一个随机整数。

random模块是Python自带的一个模块，称为Python的内置模块。除此之外，我们也可以自己设计模块，称为自定义模块；由其他程序员设计好的模块，则称为第三方模块，或第三方功能库。

② 模块的分类

✓ 内置模块

Python内部早已设定好模块，可直接导入使用。如random模块（随机数“工具箱”）、time模块（时间“工具箱”）、math模块（数学“工具箱”）、tkinter模块（图形界面开发“工具箱”）等。

✓ 自定义模块

由编程者自己设计的.py文件作为模块。

✓ 第三方模块

其他程序员设计的并开放给大家使用的模块，需先下载至本地或通过网络连接该模块，再导入使用。如matplotlib绘图模块、pandas数据分析模块等。

③ 模块的使用

模块是一个“工具箱”，因此，要使用模块中的“工具”，必须先把“工具箱”买回来。在Python程序中，使用模块之前应先将模块导入到程序中。

第一步：导入模块。

将模块导入到程序中，其目的是将模块中的程序代码导入到自己的程序中。在Python中，通过关键字import导入整个模块：

```
import random
```

第二步：使用模块。

模块导入成功之后，就像是把“工具箱”买回家了，里面的“工具”自然都可以使用。将模块导入程序后，模块中的代码都可以看成是程序的代码，模块中定义好的变量、函数等代码都可以调用。

模块中函数或变量的调用方式：“模块.成员”。其中，“.”是Python中的成员访问运算符，通过“模块.成员”，程序才能知道这些函数和变量来自哪个“工具箱”，这样不仅便于模块管理，也解决了模块中的函数或变量与程序中已有的函数或变量可能重名的问题。

例如，导入random模块并调用random模块中的randint(a,b)函数。random(1,2)表示要么产生一个1，要么产生一个2：

```
>>> import random
>>> random.randint(1,2)
1
>>> random.randint(1,2)
2
```

5．判断和循环

程序主要由三种结构组成，分别是顺序结构、选择结构和循环结构。在程序设计语言中，条件控制语句和循环语句让程序能够实现更加复杂的逻辑，完成更加复杂的任务。

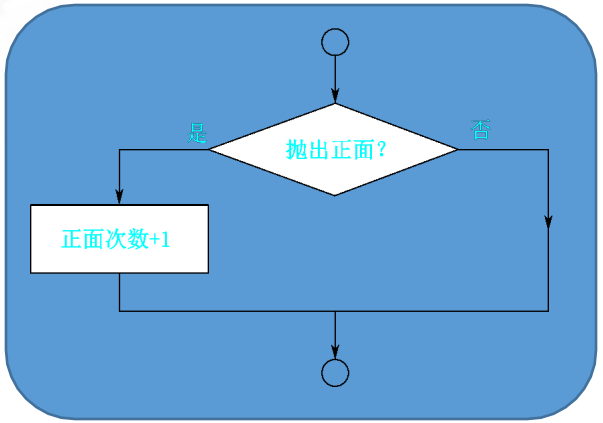
（1）条件控制语句。

条件判断在我们的生活中无处不在，在“抛硬币”问题中：

如果抛出正面，

则正面次数+1。

用程序流程图来表示这个逻辑，如图所示。



在Python中，有一个特殊语句—条件控制语句，用于进行条件判断。条件语句就像一个岔路口，程序会首先在岔路口进行条件判断，然后根据条件判断的结果进入相应的道路，控制程序执行相应的语句。

在Python中，有一个特殊语句—条件控制语句，用于进行条件判断。条件语句就像一个岔路口，程序会首先在岔路口进行条件判断，然后根据条件判断的结果进入相应的道路，控制程序执行相应的语句。

① 条件控制语句的构成。

条件控制语句由关键字、判断条件，以及子句构成。if、elif和else都是Python中的关键字，所以当输入if、elif和else时，可以看到它们的颜色为橙色（不同的编辑器颜色可能不同）。在if语句、elif语句和else语句后有一个冒号“:”标志接下来缩进的语句为其子句。

```
if 条件1:
    <语句块1>

elif 条件2:
    <语句块2>

……

else:
    <语句块n>
```

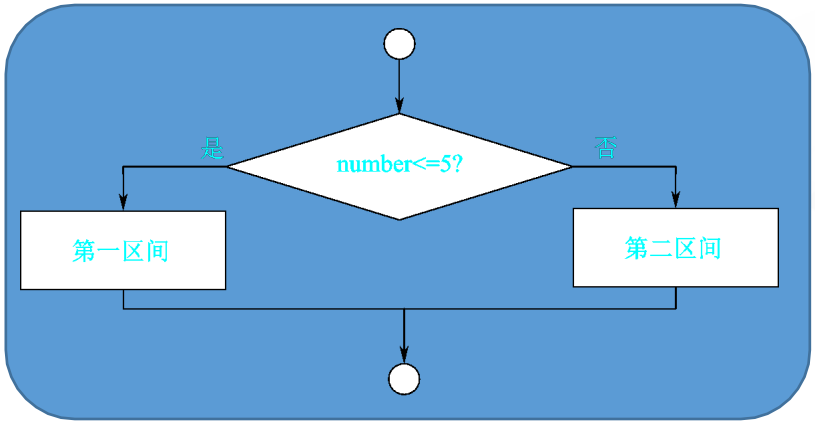
若条件 1 成立(条件 1 的逻辑值为 True):执行语句块 1 中的语句。

若条件 1 不成立，继续判断条件 2 是否成立，如果条件 2 成立(条件 2 的逻辑值为 True)：执行语句块 2 中的语句。

若前面的所有条件都不成立：执行语句块 n 中的语句。

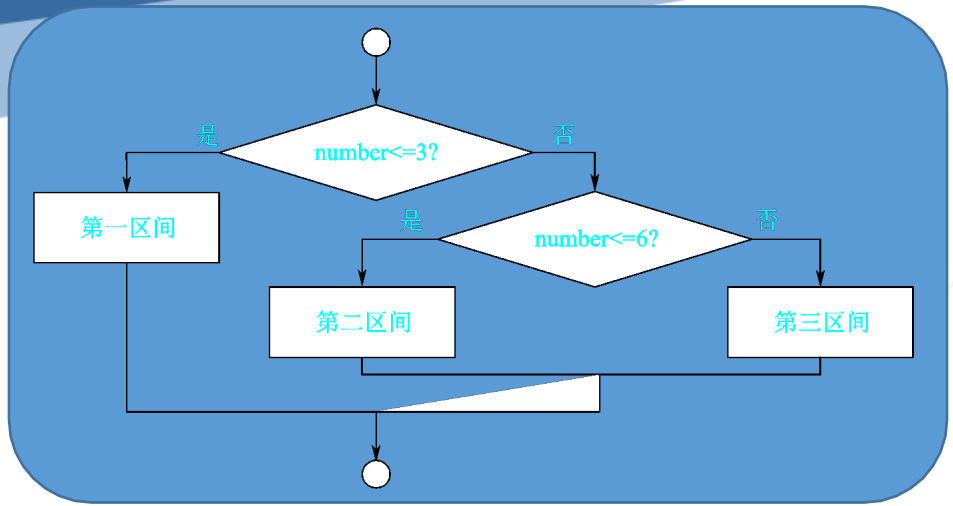
条件控制语句的运行让程序不再只有一条道路可以走，它能给程序提供多个选择，并根据不同的条件让程序走向不同的道路，得出不同的结果。在程序运行时，总是先判断if语句的条件，若if语句的条件不成立，再顺次判断后面的elif条件是否成立，若所有条件都不成立，才会执行else语句中的内容。

例1：将1～10的整数分为1～5、6～10这2个区间，获取1～10的随机数，并判断随机数所属区间。流程图如图所示。




```
checkNumber1.py
01.import random
02.number = random.randint(1,10)
03.if number<=5:
04.    print("第一区间")
05.else:
06.    print("第二区间")
```

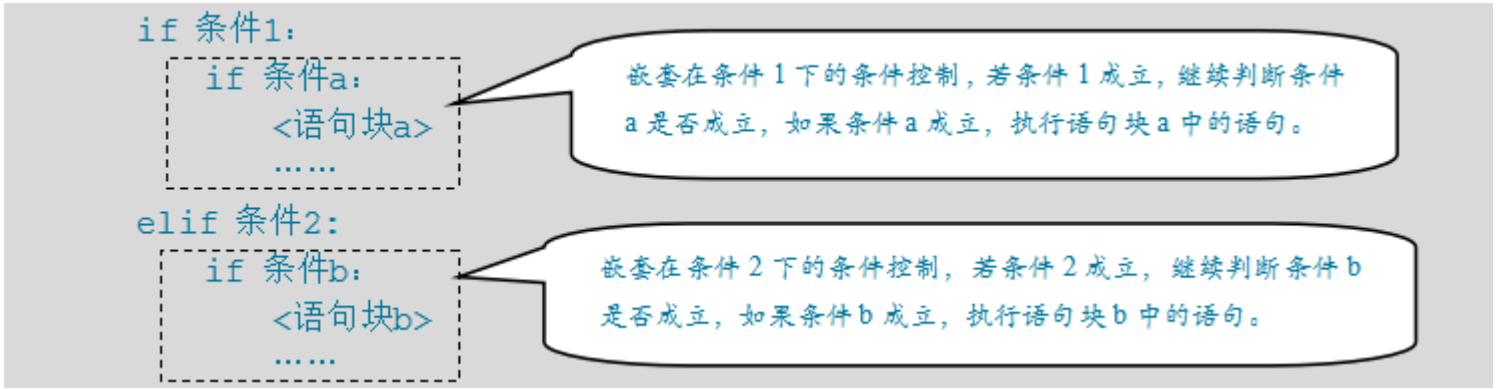
例2：将1～10的整数分为1～3、4～6、7～10这3个区间，获取1～10的随机数，并判断随机数所属区间。流程图如图所示。



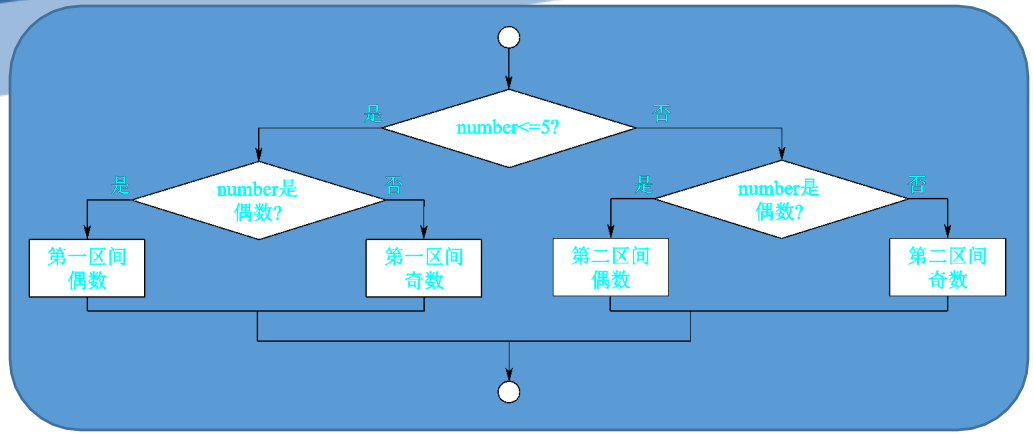

```
checkNumber2.py
01.import random
02.number = random.randint(1,10)
03.if number<=3:
04.    print("第一区间")
05.elif number<=6:
06.    print("第二区间")
07.else:
08.    print("第三区间")
```

②条件控制语句的嵌套。

像树枝从主干分了岔之后，在枝干上也可以继续分出更小的枝干一样，在复杂的条件判断中，有时需要在某个判断条件下“嵌套”新的条件判断。所谓“嵌套”，就像俄罗斯套娃一样，把一个条件控制语句嵌套在另一个条件语句块里，嵌套在里面的条件控制语句看作一个整体。在Python中，根据代码的缩进量来控制语句的嵌套层级。



例3：将1～10的整数分为1～5、6～10两个区间，获取1～10的随机数，并判断随机数所属区间及其奇偶性。流程图如图所示。




```
checkNumber3.py
01.import random
02.number = random.randint(1,10)
03.if number<=5:
04.    if number%2 == 0:
05.        print("第一区间的偶数")
06.    else:
07.        print("第一区间的奇数")
08.else:
09.    if number%2 == 0:
10.        print("第二区间的偶数")
11.    else:
12.        print("第二区间的奇数")
```

内层判断
内层判断
外层判断

（2）循环语句。

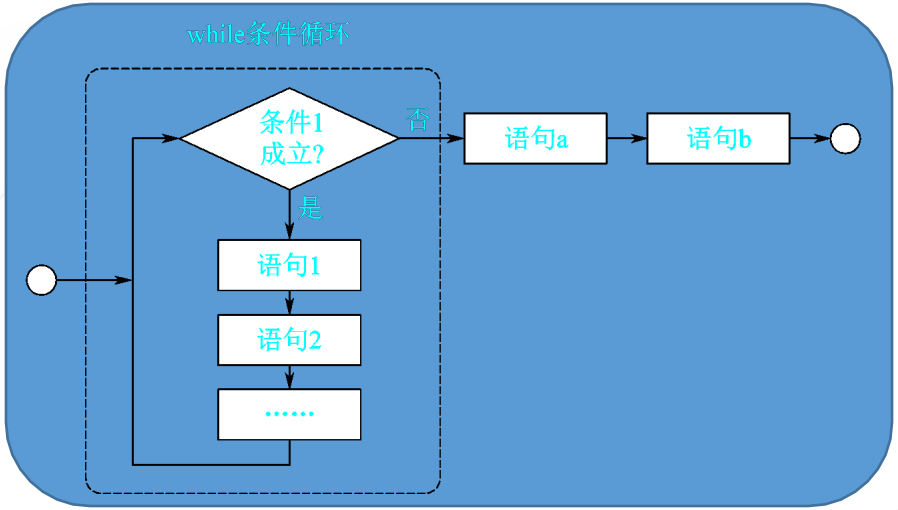
① while条件循环语句。

条件循环语句由while关键字、判断条件及循环体组成，其流程图如图所示。在while语句后面，以一个冒号“:”标志其子句的开始。

```
while 条件1:
        语句1
        语句2
        ……
语句a
语句b
……
```

循环体

循环条件：条件满足方可执行继续循环，条件不满足时结束循环



while是“当……时”的意思。当程序运行到while语句时，首先判断条件1是否成立，若条件1成立，则执行一次循环体（语句1→语句2→……）。循环体执行一次之后，再次判断条件1是否成立，若条件1依然成立，则再执行一次循环体，如此循环往复。直到条件1不被满足时，程序不再执行循环体，结束循环，继续执行后面的语句（语句a→语句b→……）。

在“抛硬币”问题中，抛100次硬币就是一个循环任务，循环条件为“实验次数是否达到100次？”，在开始循环之前，将实验次数cnt变量初始化为0，进入循环后，每次抛硬币cnt都自增1，因此，循环次数为100次。

tossCoin.py

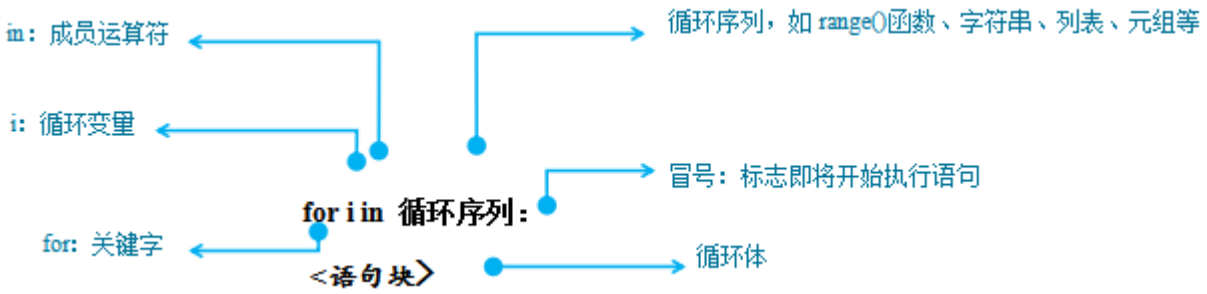
```
01.import random

02.up_n = 0 #记录扔出正面的次数
03.total_n = 100 #代表实验总次数
04.cnt = 0 #记录实验次数

05.while cnt<100:
06.    result = random.randint(0,1)
07.    print(result)
08.    if result==1:
09.        up_n = up_n +1
10.    cnt = cnt + 1

11.p = up_n/total_n
12.print('扔出正面的频率为: '+str(p))
```

② for计数循环语句。

for循环和while循环一样，都能控制一个程序段重复连续执行多次。但for循环的循环条件比较特殊，它根据循环执行的次数来判断是否继续循环，因此for循环也称为计数循环。

for循环由关键字for、循环变量、循环序列和循环体组成，其结构如图所示。

✓ 循环序列：for循环中的“储物柜”中有序存放着一组数据。每次循环，循环序列中的数据将被依次取出，因此，循环执行的次数就是循环序列中元素的个数。序列型数据都可以作为循环序列，如range()函数、字符串、列表、元组等，其中，range()函数是Python的内置函数，它将产生一个数字序列。



✓ 循环变量：for循环中的“计数员”，本质是一个变量。每次循环时，都从循环序列中取出下一个元素，并将该元素赋值给循环变量i，当序列中的元素取完时，返回False，循环结束。

✓ 循环体：for循环中的“执行者”，每次循环中被执行的语句（块）。

在“抛硬币”问题中，由于“抛硬币”的过程要执行100次，即以循环次数来作为循环的判断条件，因此，也可以用for计数循环来实现：

tossCoin.py

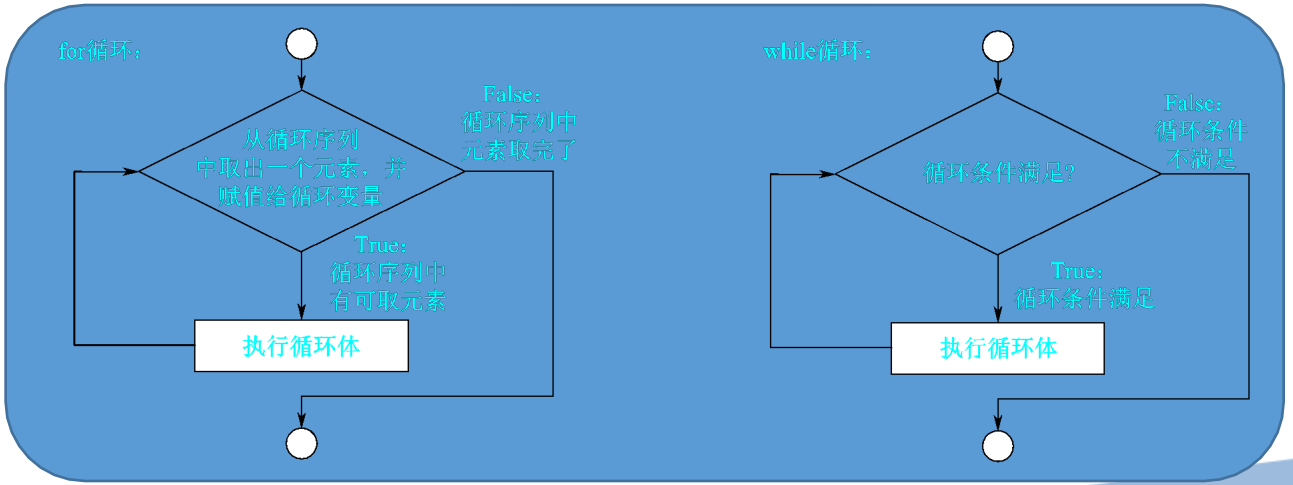
```
01.import random

02.up_n = 0 #记录扔出正面的次数
03.total_n = 100 #代表实验总次数

04.for i in range(100):
05.    result = random.randint(0,1)
06.    print(result)
07.    if result==1:
08.        up_n = up_n +1

09.p = up_n/total_n
10.print('扔出正面的频率为：'+str(p))
```

③ while条件循环与for计数循环。

for循环和while循环都属于循环语句，它们有何共同之处？有何不同之处？其适用场景分别是怎样的？

while条件循环和for计数循环都属于循环结构，都能控制程序循环执行一段代码，且对程序的控制都依赖于条件判断，若条件满足则执行循环体，否则循环结束。

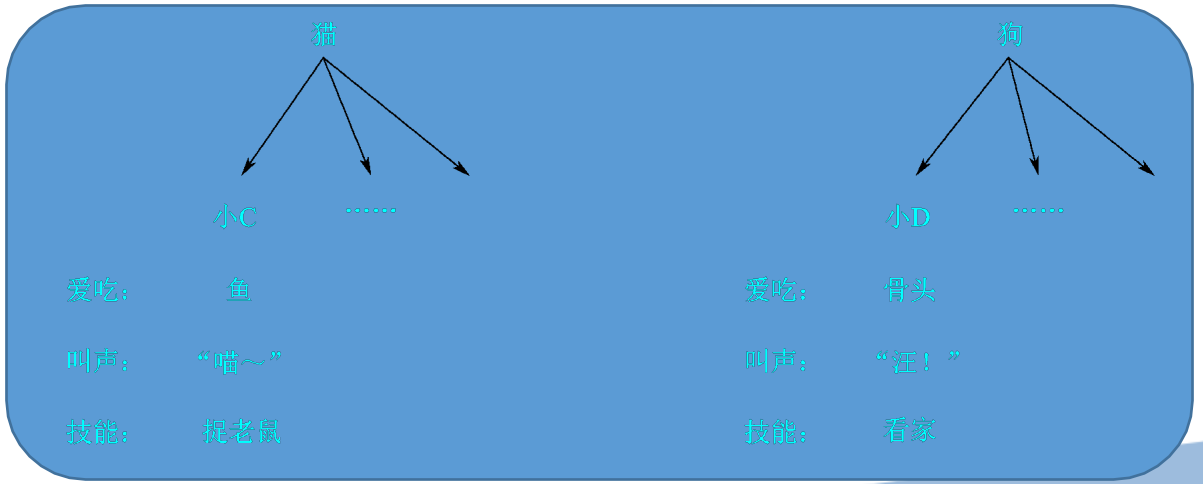
不同的是，for循环需要对序列进行遍历，且其条件判断比较特殊，为“循环序列中的元素是否被取完？”；而while循环可以对任何条件进行判断，应用更为广泛。两种循环语句的对比如图所示。



6．面向对象

（1）类和对象。

类和对象的概念来源于我们的生活。例如，小C是一只猫，它喜欢吃鱼，叫声是“喵~”，会捉老鼠；小D是一只狗，它喜欢啃骨头，叫声是“汪！”，会看家。而世界上除了小C和小D，还有其他的猫和狗，并且所有的猫都有相同的属性（如喜欢吃鱼、叫声是“喵~”）和技能（如会捉老鼠），所有的狗也都有相同的属性（如喜欢啃骨头，叫声是“汪！”）和技能（如会看家）。其示意图如图所示。



可以说，“猫”是一个“类”，小C是这个类中的一个“对象”或“实例”；“狗”也是一个“类”，小D是这个类中的一个“对象”或“实例”。因此，类（class）是具有相同属性和相同操作的一些对象的“模板”，对象（object）则是根据类这个模板创造出来的一个具体的“实例”。

Python中每个数据都可看作一个对象，而每个数据的类就是它所属的数据类型，不同类型的数据属于不同的类，具有不同的特征。比如，123是一个整数，它是整数这个类的一个对象，可以进行数学运算；'123'是一个字符串，它是字符串这个类的一个对象，可以进行文本操作。

（2）面向对象和面向过程。

“面向对象”和“面向过程”本质都是解决问题的一种思想方法。相比之下，面向过程关注解决问题的一系列步骤；面向对象则是一种以事物为中心的思想方式，事物就是对象，具有一定的属性和方法。但在解决实际问题时，面向对象的方法也含有面向过程的思想，可以说面向过程是一种基础的方法，它考虑的是实际的实现。

举个生活中常见的例子：玩五子棋游戏。面向过程的设计思路是先分析问题的每个步骤：开始五子棋→黑子先走→黑子落子→判断输赢→白子落子→判断输赢→黑子落子→判断输赢……最终结束。将每个步骤用不同的方法来实现，这就是面向过程。

如果使用面向对象的方法来解决问题，整个五子棋便可以分为：黑方、白方、棋盘、裁判这四个对象。其中，黑方和白方负责接收两名选手的输入，并告知棋盘，棋盘负责显示当前棋局中棋子的布局情况，同时，裁判来对棋局的胜负进行判定。

说一说

请结合“双十一”网络购物满减活动，谈谈程序设计思想的应用。

5.2.2 编辑、运行和调试简单程序

掌握了程序设计的基础知识后，小华找到堂兄，想要学习怎样设计程序来解决他还没想明白的打折问题。堂兄告诉小华，和之前的“抛硬币”问题一样，要用程序设计来解决打折问题，首先得对这个问题进行分析，然后设计相应的算法，并根据所设计的算法来编写程序。运行程序之后，还可以对问题进行进一步的反思和迁移，对程序进行优化和改进，并将解决问题的办法迁移到其他问题的解决中。这一节里，我们就来编写程序解决打折问题，体会通过程序设计解决问题的过程和方法。

1．问题描述

假设，某商店出售的某件商品，成本为4.5元，售价为15元，在第一周的营业中，销售量为100个。在接下来的一周中，将进行打折活动。经过市场调查，发现商品每降价1元，顾客就会增加20%的购买意愿，这意味着商品每降价1元，销售量就可能增加20%。

若最低折扣为5折，在接下来一周的打折活动中，为让商店获得最高利润，应该打几折?

2．问题分析

商品每降价1元，销售量就增加20%，根据这一市场调查结果，可以根据以下公式的计算步骤，计算出每种打折情况下活动期间的总利润：

折后价=原价×折扣×0.1

活动期间销售量=上周销售量×[1+20%×（原价-折后价）]

活动期间总利润=（折后价–成本）×下周销售量

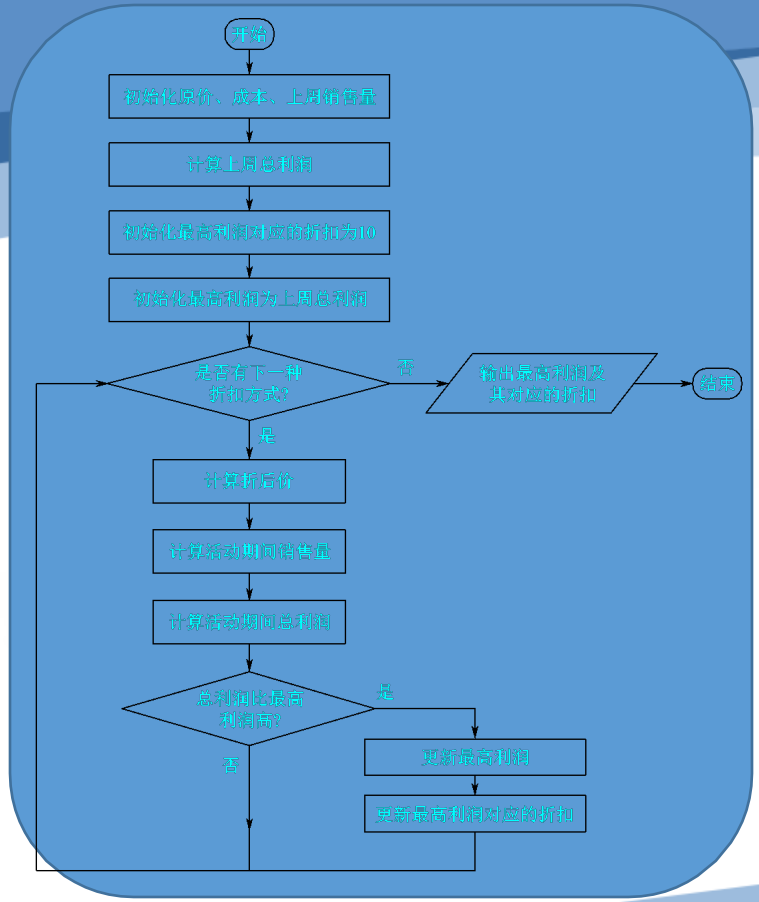
为各个数据创建变量见表所示。

数　据	变　量
原价	price
成本	cost
上周销售量	sellNum
折扣	discount
折后价	discPrice
活动期间销售量	discSellNum
活动期间总利润	discTotalProfit

知道了活动期间每种打折情况下总利润的计算方法，接下来，只需要计算出所有打折情况下，活动期间的总利润，并找到其中总利润最高的打折方法。

3．算法设计

在拍卖活动中，工作人员会先给出一个底价，然后让所有参与者在此基础上报价，每当出现比当前最高价更高的价格时就更新当前最高价……直到没有人再报出更高的价格，就以当前最高价成交。寻找最佳打折方法的过程也一样，首先创建两个变量分别存储最高利润和最高利润对应的折扣，将最高利润对应的折扣初始化为10，表示不打折，最高利润初始化为不打折时的一周利润。然后从5折开始递增折扣，依次计算每种打折方式下的总利润。如果某种折扣方式下，总利润比最高利润高，则更新最高利润和对应的折扣……直到计算并比较完所有的折扣方式下的总利润，也就找到了所有方式下的最高利润及其对应的折扣方式。将这一算法用流程图表示，如图所示。


开始
初始化原价、成本、上周销售量
计算上周总利润
初始化最高利润对应的折扣为10
初始化最高利润为上周总利润
是否有下一种折扣方式?
否
输出最高利润及其对应的折扣
结束
是
计算折后价
计算活动期间销售量
计算活动期间总利润
总利润比最高利润高?
是
否
更新最高利润
更新最高利润对应的折扣

可以看出，计算每种打折情况下的总利润的过程是一个循环结构，并且在每一次循环中，还嵌套了一个选择结构，用于判断是否需要更新最高利润及其对应的折扣。

接下来，就可以根据所设计的算法，编写程序来解决打折问题了，示例程序如下。

```
discount.py01.price = 1502.cost = 4.503.sellNum = 100
04.totalProfit = (price - cost) * sellNum 05.discOfBigProfit
= 1006.bigProfit = totalProfit 07.for discount in
range(5,10):08.    newPrice = price * discount * 0.109.
newSellNum = sellNum * (1 + 0.2 * (price - newPrice))10.
newTotalProfit = (newPrice - cost) * newSellNum11.    if
newTotalProfit > bigProfit:12.        bigProfit =
newTotalProfit13.        discOfBigProfit = discount 14.print
('打'+ str(discOfBigProfit) + '折，预计利润最高：' +
str(bigProfit) + '元。')
```

注：第7行，range(a,b)函数是Python的内置函数，将返回一个[a,b)左闭右开的整数序列，如range(5,10)将返回数字序列5,6,7,8,9。在for循环的遍历中，循环变量discount将依次被赋值为5,6,7,8,9。这样，就能通过for循环遍历每种折扣，并计算和比较该折扣方式下的总利润了。

反思与迁移

在打折问题中，商品只有1项，但如果商店有多个商品，每种商品的成本和原价都各不相同，如何计算不同打折情况下，所有商品的总利润呢？假设：商店有4种商品，每种商品的成本、原价，以及上周销售量见表5-11，请设计算法编写程序计算出商店上周营业的总利润。

商　品	成本（元）	原价（元）	上周销售量（个）
商品1	4.5	15	100
商品2	8	20	120
商品3	8.5	20	150
商品4	9	22	120

最简单的办法，我们可以依次计算出4种商品的利润，然后将4种商品的利润相加。如下所示：

```
calProfit.py01.price1 = 1502.price2 = 2003.price3 =
2004.price4 = 22 05.cost1 = 4.506.cost2 = 807.cost3 =
8.508.cost4 = 9 09.sellNum1 = 10010.sellNum2 = 12011.sellNum3
= 15012.sellNum4 = 120 13.totalProfit1 = (price1 - cost1) *
sellNum114.totalProfit2 = (price2 - cost2) *
sellNum215.totalProfit3 = (price3 - cost3) *
sellNum316.totalProfit4 = (price4 - cost4) *
sellNum417.totalProfit = totalProfit1 + totalProfit2 +
totalProfit3 + totalProfit418.print('上一周总利润为：' +
str(totalProfit))
```

可以看出，这种方法虽然比较直观，但是当商品数量更多时，代码会很长，不利于进行程序的编写和维护。因此，不妨将每种商品的各项信息都存储在列表里，使信息的表达和处理都更加方便。示例程序如下：

注：第5行中，range(4)是range(0,4)的简写，将返回一个长度为4的整数序列0,1,2,3。第5～7行，将循环4次，每次循环中，用oneProfit变量存储单个商品的利润，并将单个商品的利润叠加到总利润中（见第7行）。这样，当循环结束时，totalProfit的值就是所有商品的总利润。

```
calProfit.py01.prices = [15, 20, 20, 22]02.costs = [4.5, 8,
8.5, 9]03.sellNums = [100, 120, 150, 120] 04.totalProfit =
005.for i in range(4):06.    oneProfit = (prices[i] -
costs[i]) * sellNums[i]07.    totalProfit = totalProfit +
oneProfit 08.print('上一周总利润为：' + str(totalProfit))
```

说一说

在拍卖活动中，拍卖流程是如何的呢？类比该问题，请描述一个寻找最佳打折方法的解决办法。

5.2.3 了解典型算法

1．枚举算法

在解决打折问题的时候，我们实际上对每种打折方式进行了遍历，即计算了每种打折方式下的总利润，并判断该利润是否是最高利润。像这样将所有可能的情况遍历的方法，就是枚举，枚举算法是解决问题的一种典型算法，举例如下：

【例1】要将100元兑换成10元或者5元的币值，共有哪些兑换方式？

【例2】一个两位数密码，每一位数可以取0～9的任意数字，如何找到真正的密码？

【例3】田忌赛马问题中，让三个不同等级的马以什么顺序出场有可能获胜？

【例4】到达目的地有多条路径，哪条路径用时最短？

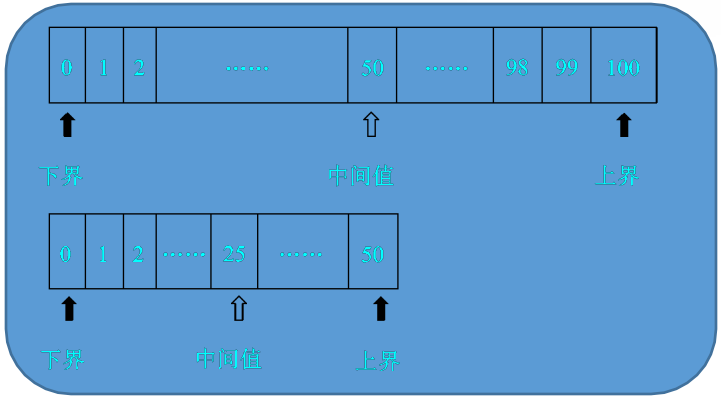
……

适用情况：研究对象可数且有范围，最优解可从该范围中逐一列举检验得到。

枚举算法的基本结构：确定范围—列举—检验。

2．二分查找法

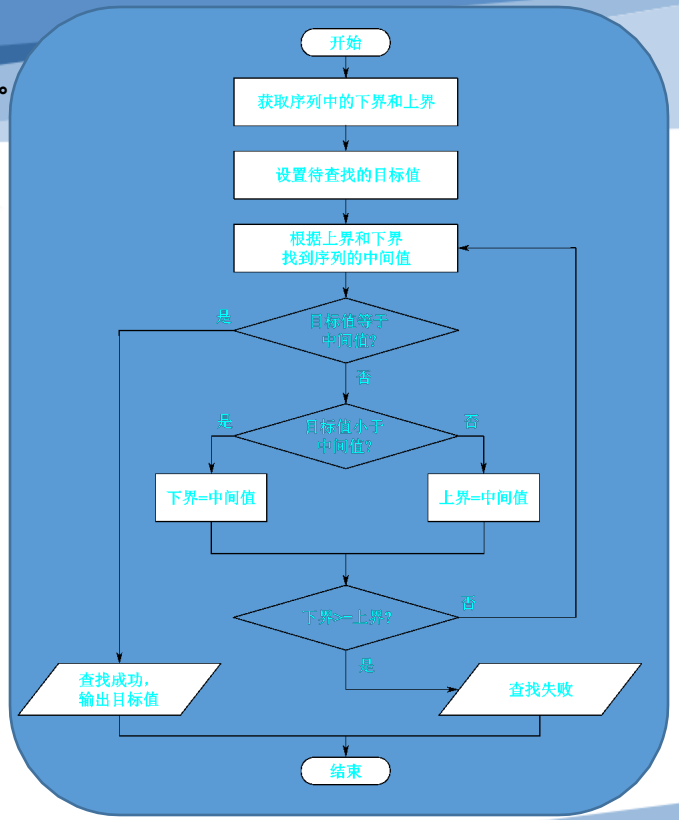
二分查找法也称折半查找，通常用于从一组有序数据中查找目标数据，利用二分查找法可以提高查找的效率。二分查找法的基本思路和玩猜数游戏的思路类似：假设要猜测的数字是0～100的一个数，最好的办法就是从0～100的中间数50开始猜，若目标数字小于50，就猜0～50中间的数字25；若目标数字大于50，就猜50～100中间的数字75；若目标数字刚好是50，结束猜数。重复此过程，不断缩小目标数字的范围，直到猜出正确的数字。猜数游戏示意图所示。



任务情景

临近节日，小华发现很多商店都在打折，促销活动吸引了很多的顾客去购买商品。小华心想：虽然打折活动让每个商品的价格降低了，但是销售量也增多了，那么商家最后获得的利润是比平时更高了还是更低了呢？如果他将来也开一家店，到了打折季的时候，为了获得最高的利润，怎么决定打几折呢？怀着这些疑问，小华找到了堂兄。

二分查找法的基本思路与此类似，用程序流程图表示该算法，如图所示。



说一说

逐个列举判断，看似简单，但有时也确不失为一种好办法。在生活中，我们也常常运用枚举的思想解决问题，比如轮胎漏气了，得一点一点挨着去找漏气的位置；为了挑出一碗红豆中的绿豆，也得一个一个去挑……而当我们把这种方法运用到程序编写中时，计算机凭借自己的高超的计算速度和准确度，可以帮我们解决很多问题。请想一想枚举算法还可以解决学习和生活中的什么问题呢？枚举算法的使用情况和基本结构是怎样的呢？

5.2.4 使用功能库扩展程序功能

1．拓展任务描述

请设计程序，让用户输入商店一周中每一天的销售量，并绘制柱状图分析商店一周内的销售情况。

2．问题分析和算法设计

通过input()函数可以获取用户的输入，对于一周七天的输入，可以通过for循环实现七天销售量的连续输入。

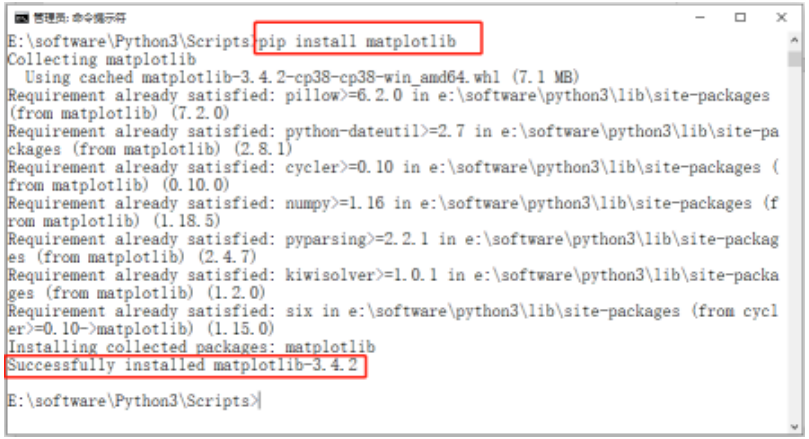
而要绘制一周销售量的柱状图，可以通过Python的第三方功能库matplotlib（2D绘图功能库，非常适合绘图）来实现，周一到周日作为柱状图的横轴数据，每天的销售量作为柱状图的纵轴数据。

3．matplotlib第三方功能库的安装和使用

（1）安装第三方功能库。

第三方功能库和内置模块不同，需要通过pip命令联网下载安装。搜索“cmd”，打开“命令提示符”窗口，如图所示。

接下来，将路径定位到Python文件夹下的Scripts文件夹下，例如，若Python安装在E盘下的“software”文件夹下，则首先在cmd命令窗口中将路径定位到“E:\software\Python3\ Scripts”。接着，输入pip安装命令“pip install 库名”，按回车键后即可开始进行第三方功能库的下载安装，直到提示“Successfully installed matplotlib-3.4.2”，表示安装成功。如图所示。



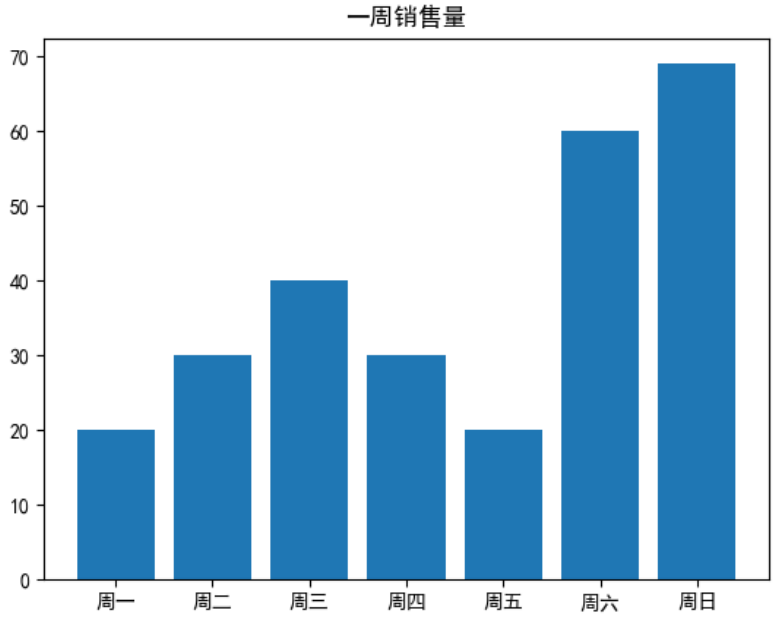
（2）使用matplotlib第三方功能库绘制柱状图。

Python第三方功能库的使用和内置模块一样，需要先将其导入程序中。通常，习惯在导入matplotlib功能库时为其取别名为mpl。另外，matplotlib库中包含多个子库用于不同的图形绘制，其中的pyplot子库是用于绘制柱状图的功能库，通常习惯为pyplot取别名为plt。

在绘制柱状图时，只需调用pyplot功能库中的bar()函数，并传入两个列表参数，分别作为柱状图的横轴和纵轴数据。例如，周一到周日的销售量分别为20、30、40、30、20、60、69，下面设计程序绘制这周的销售量柱状图。绘制出的柱状图如图所示.。

weekSell.py

```
01.import matplotlib as mpl
02.import matplotlib.pyplot as plt

03.mpl.rcParams['font.sans-serif']=['SimHei'] #用于正常显示中文标签

04.daySellNums = [20, 30, 40, 30, 20, 60, 69]
05.dayNames = ["周一", "周二", "周三", "周四", "周五", "周六", "周日"]

06.#绘制柱状图
07.fig=plt.figure()  #创建一个画布
08.plt.title("一周销售量")  #设置柱状图的标题
09.plt.bar(dayNames, daySellNums)
10.plt.show() #让图形显示出来
```



若需要由用户来输入每天的销售量，则初始化daySellNums列表为一个空列表，然后在for循环中获取用户输入的销售量，并将每天的销售量添加到daySellNums列表中，最后再利用matplotlib功能库绘制柱状图即可。

```
      weekSell.py 01.import matplotlib as mpl02.import
matplotlib.pyplot as plt 03.mpl.rcParams['font.sans-
serif']=['SimHei'] #用来正常显示中文标签 04.daySellNums =
[]05.dayNames = ["周一", "周二", "周三", "周四", "周五", "周六",
"周日"] 06.for day in dayNames:07.    daySellNum = int(input("
请输入" + day + "销售量："))08.
daySellNums.append(daySellNum)09.#绘制柱状图10.fig=plt.figure()11.plt.title
("一周销售量")12.plt.bar(dayNames, daySellNums)13.plt.show()
```

Python的一大特点是具有丰富的功能库，从上面的例子中可以感受到，利用功能库可以实现很多复杂的功能。所以，当我们需要实现一些复杂功能时，可以先了解是否已经有相关的功能库能够实现该功能，这样将大大提高编程效率。而学会检索和学习功能库的使用方法，是利用功能库解决问题的关键。

说一说

请说出利用Python功能库实现复杂功能的案例，如网络爬虫。

第5章

程序设计入门

任务2 设计简单程序

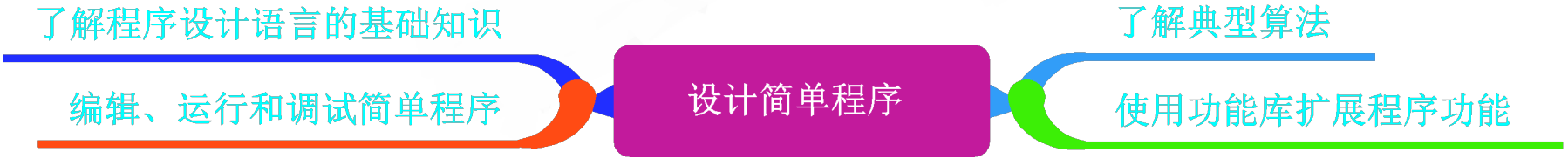
主 编 | 傅连仲 等

目 录

Contents

设计简单程序

设计程序是将解决问题的方案付诸实践的过程。而了解程序设计语言的基础知识是需要迈出的第一步，因为程序设计语言是我们与计算机进行沟通的重要工具。接下来，将学习如何编写、运行和调试简单程序，并了解典型算法和功能库的使用方法，编写程序来解决实际问题。在这个过程中，将不断体会运用程序设计解决问题的过程和方法，体会程序设计的理念。



任务情景

临近节日，小华发现很多商店都在打折，促销活动吸引了很多的顾客去购买商品。小华心想：虽然打折活动让每个商品的价格降低了，但是销售量也增多了，那么商家最后获得的利润是比平时更高了还是更低了呢？如果他将来也开一家店，到了打折季的时候，为了获得最高的利润，怎么决定打几折呢？怀着这些疑问，小华找到了堂兄。

任务分析

堂兄听了小华的疑问，说：“打折和给商品定价可是一门学问啊，根据对消费者消费心理等情况的了解，可以编写程序来计算打几折可以获得最高利润，还能够预测打折活动带来的具体利润呢。”

学习程序设计语言是与计算机进行沟通的基础，本节以Python语言作为编程工具，学习如何创建并运行程序，了解程序设计语言的基础知识，并设计程序来帮助小华解决打折问题。

5.2.1 了解程序设计语言的基础知识

1．Python开发环境IDLE

从Python的官网上下载并安装了Python之后，同时也就安装了IDLE（集成开发环境）—Python的官方标准开发环境。

IDLE集成了整个代码编辑时要用的东西，包括交互式Shell和编辑器。其中，交互式Shell相当于一个简化的编辑器，当只需要编写一些小的验证性代码，可以在Shell中编写代码并执行；但如果需要编写完整的Python程序，或者需要将代码保存并希望能够反复运行，就要使用编辑器了。

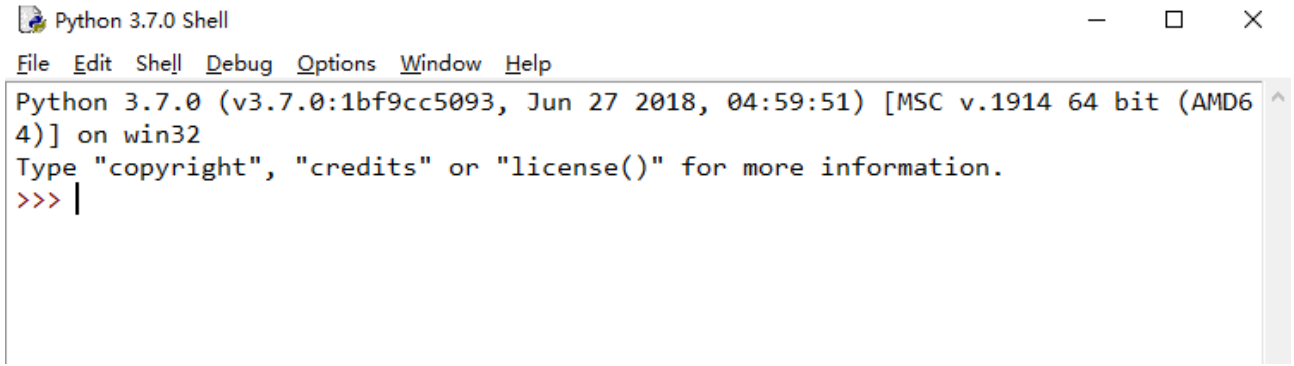
2．程序设计

计算机是一个没有生命的机器，是一个不知道自己该做什么、但却十分愿意服从命令的机器。手机如果没有“程序”，就是一堆没有用的零件，我们无法用它通话、上网和玩游戏。程序设计（Program Design）就是将问题解决的方法步骤编写成计算机可执行的程序的过程。简单来说，就是告诉计算机要做什么，并且每一个行为的细节和顺序都要说清楚、可执行。这样，计算机就能够很快速地、正确地完成所有“指令”，最终解决问题或完成任务。

（1）在Shell中输入并运行Python指令。

在Windows操作系统左下角的搜索框中，输入“IDLE”，找到IDLE应用程序，单击即可启动，出现Python的交互式Shell窗口，如图所示。

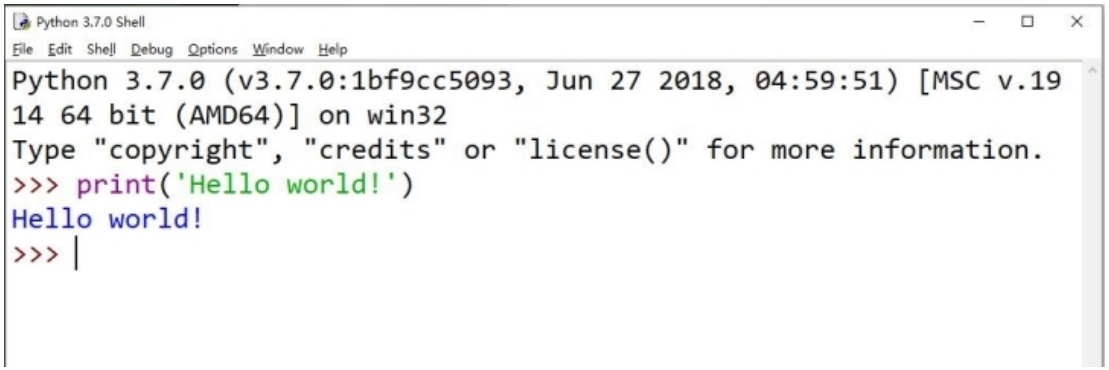
在“>>>”提示符后面，输入一条Python指令，回车，Python将执行该指令，并在下一行显示该指令执行的结果。指令执行完成后，将在下一行出现一个新的“>>>”提示符，等待下一条指令的输入。



例如，在>>>后输入一行Python语句：print('Hello world!')，并按下【Enter】键，Python将在下一行打印输出：“Hello world!”。

注意：代码中所有字符均为英文字符，包括引号和括号。并且，print()语句的前面没有空格，如果有多余的空格，Python执行指令时会报错，红色的SyntaxError是报告的错误类型。

Python语言以缩进控制语句的级别，就像编写文档时设置大纲级别为1级、2级、3级。在Python中，有相同缩进的一组连续语句属于同一逻辑层级的语句，在每行语句开头的空格或制表符就是缩进，通常用4个空格或1个制表符表示一个缩进。因此，在编写Python程序中，要严格控制每行语句开头的缩进。



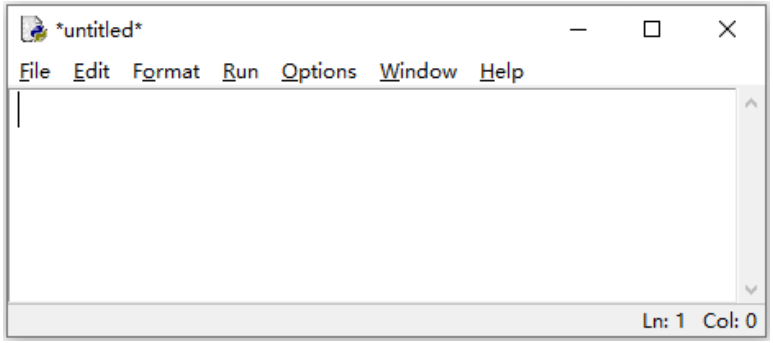

```
>>> print('Hello world!')

SyntaxError: unexpected indent
>>>
```

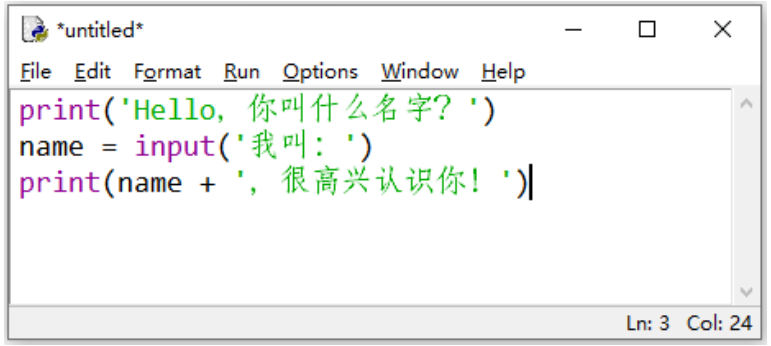
（2）在IDLE中创建并运行Python程序。

在IDLE的交互式Shell中，虽然能方便快速地执行Python指令，但每次只能输入一行代码、执行一条指令，不能连续执行多条指令。因此，我们需要一个新的方式来执行一连串的Python指令—程序。

① 第一步：创建程序。

启动IDLE，单击File→New File，弹出IDLE的文件编辑器窗口。接着，请在编辑器窗口中输入3行Python语句。

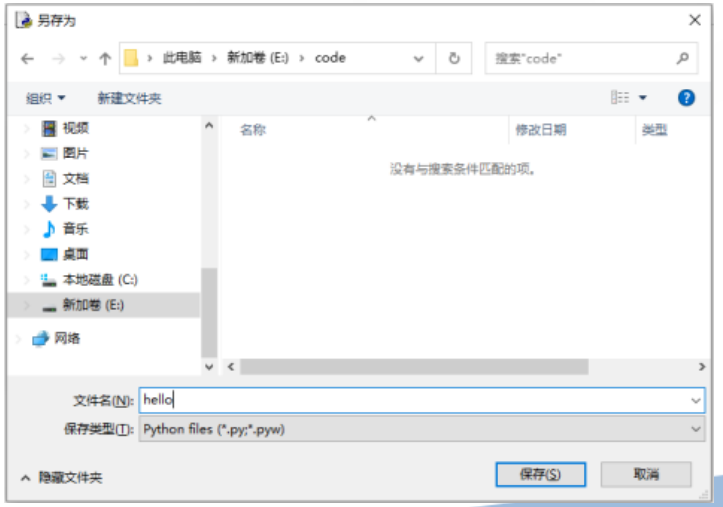




② 第二步：保存程序。

按【Ctrl+S】组合键或者单击File→Save as保存源代码文件，弹出另存为窗口，在文件名文本框中输入文件名，如“hello”，保存类型选择Python files，然后单击保存按钮。保存成功后，即可在保存的地方找到刚刚创建的Python程序文件hello.py。

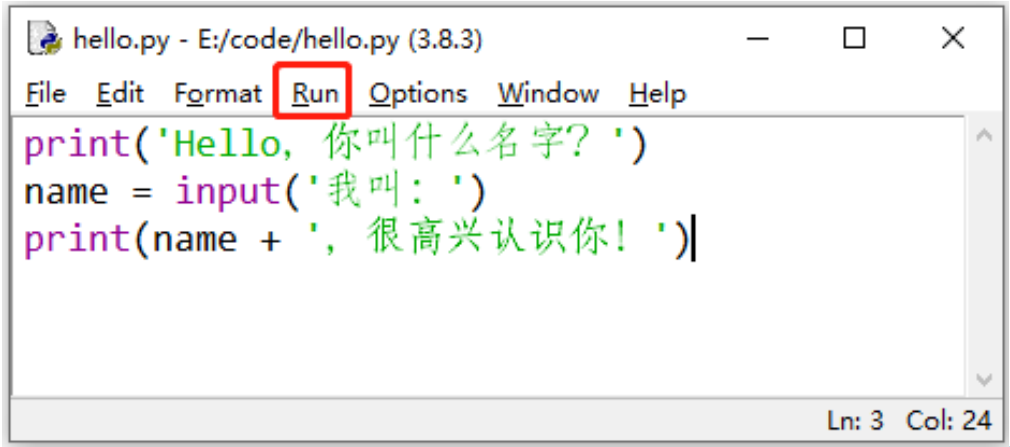
Python程序文件也叫Python可执行文件，它能够存储多条Python指令序列，是一个后缀名为.py的文件，运行它时，其中的指令可以被连续执行。



③ 第三步：运行程序。

单击Python编辑器菜单栏的Run→Run Module或者按【F5】快捷键即可运行程序，界面如图所示。

运行之后，将首先在IDLE的Shell面板输出一行文字“Hello，你叫什么名字？”，然后在第二行输出“我叫：”，同时光标闪烁，等待用户输入。此时，小华通过键盘输入了他的名字“小华”，并按下【Enter】键，之后，程序继续在第三行输出：“小华，很高兴认识你！”。



④ 第四步：再次打开保存的程序。

将一连串程序指令保存成一个Python程序文件，不仅可以让其中的指令连续执行，还可以将其保存，当下次需要其进行查看、编辑和修改时，再次打开保存的程序即可，打开程序的方式有以下两种：

方式一：从IDLE中打开保存的程序文件。单击File→Open，在所出现的窗口中选择文件，并且单击打开按钮。刚刚保存的hello.py程序将会在文件编辑窗口中打开。

方式二：直接打开程序文件。找到要弹出的Python程序文件并单击鼠标右键选择Edit with IDLE。

说一说

现在，请运行程序，输入你的名字，完成与计算机的第一次交流。

2．数据类型和表达式

在程序设计中，将现实生活中的问题转化成计算机能够处理的形式是利用计算机解决问题的关键步骤，而数据和表达式就是对问题进行重新表述的关键。

（1）数据类型。

数据是对信息的刻画，不同的数据类型可以表达不同的信息。在Python中，常见的数据类型见表。

数 据 类 型	描　　述
整数型	数学中的整数，包括正整数、负整数和0。如1，-21，0等
浮点型	数学中的小数，小数点是它的标志。如3.14，-2.0等
字符串型	用单引号、双引号或三引号表示。如'a'、"Hello"，'''你好！'''等
布尔型	只有两种值：True和False，分别表示逻辑的“真”和“假”

① 数据类型的转化。

不同的数据类型之间存在差异，不能进行相互运算。因此，必要的时候，需要对数据类型进行转化。在Python中，内置函数str()、int()、float()可以分别将数据转化成字符串型、整数型和浮点型，其作用及示例见表。

转化函数	作　用	示　例
str()	转化成字符串	str(123)　　转化结果：'123'
int()	将数字或长得像整数的字符串转化成整数	int(1.5)　　转化结果：1int('100')　　转化结果：100int('1.5')　　转化失败！！
float()	将数字或长得像浮点数的字符串转化成浮点数	float(2)　　转化结果：2.0float('2.5')　　转化结果：2.5float('hello')　　转化失败！！

例如，字符串和字符串之间可以通过“+”连接运算符，将两个字符串连接成一个字符串：

```
>>> 'Hello ' + '2021'
'Hello 2021'
```

数字和数字之间也可以通过加法运算符“+”计算两个数字的和：

```
>>> 10 + 2021
2031
```

但字符串和数字之间却不可以进行“+”加法运算，在下面的反例中，可以从TypeError看出错误的原因：

```
>>> 'Hello ' + 2021Traceback (most recent call last):  File "<pyshell#5>", line 1, in <module> 'Hello ' + 2021TypeError: can only concatenate str (not "int") to str
```

```
>>> '10' + 2021Traceback (most recent call last): File "<pyshell#7>", line 1, in <module>    '10' + 2021TypeError: can only concatenate str (not "int") to str
```

这时，我们可以根据需要对数据类型进行转化：

```
>>> 'Hello ' + str(2021)'Hello 2021'
```

```
>>> int('10') + 20212031
```

② 组合数据类型。

除了以上几种常见的数据类型，还有一种组合数据类型，可以将一组数据统一存储和管理，以便于程序对一组数据进行批量处理。例如，遍历一组水果的英文单词、找到运动会报名了跑步和跳远的同学、登记一个学生的各项信息……Python中，字符串实际上也是一种组合数据类型，每个字符都是它的元素。除此之外，常见的组合数据类型见表。

组合数据类型	描　述
列表	用方括号[]创建，其中的元素有序存放，索引号从0开始，且可修改、增添、删除元素。如[1,1,3]，['apple','banana']等
元组	用小括号()创建，其中的元素有序存放，索引号从0开始，不可修改、增添、删除元素。如(1,1,3)，('apple','banana')等
集合	用花括号{}创建，其中的元素无序存放，无索引号，但不能重复。如{1,2,3}，{'小华','大华','天歌'}，{'小华','清波'}
字典	用花括号{}创建，其中每个元素都是一个键值对（key=>value），具有映射关系，无序存放，可通过键获取对应的值。如{'name':'小华', 'age':15}

例如，将一组水果单词存放在fruit列表中，表示用户喜欢的水果。开始用户将第一喜欢的水果变成了桃子（peach），接着用户又新增了一个喜欢的水果草莓（strawberry），最后用户又不喜欢梨（pear）了。用程序表示这个过程：

```
>>> fruit = ['apple', 'banana', 'pear', 'grape']

>>> fruit[0] = 'peach'
>>> fruit.append('strawberry')
>>> fruit.remove('pear')

>>> fruit
['peach', 'banana', 'grape', 'strawberry']
```

修改第一个元素为'peach'

在末尾添加一个元素'strawberry'

移除值为'pear'的元素

（2）表达式。

计算机不仅能进行数学运算，还能进行逻辑运算，对应的Python中表达式也有算术表达式和逻辑表达式。

① 算术表达式。

在Python中，“算术表达式”就是数学中的“算式”，1+2、2-5都是算术表达式。它由算术运算符和操作数组成，例如，1+2中，1和2都是操作数，“+”是算术运算符。

✓ 算术表达式的值。

算术表达式的值就是算式的计算结果，比如：算术表达式1+2的值是3。另外，单个数字也可以看作一个特殊的算术表达式，其值就是这个数本身。在Python的交互式Shell中输入一个算术表达式，按【Enter】键后即可得到该算术表达式的值。例如，计算表达式1+2的值：

```
>>> 1+2
3
```

除了“+”，Python中还有其他一些常用的算术运算符。

算术运算符	算术表达式示例	描　述	值
+	1+2	1加2	3
-	1-2	1减2	-1
*	1*2	1乘以2	2
/	10/4	10除以4	2.5
//	10//4	10整除4（10除以4，取商的整数部分）	2
%	10%4	10除以4，取余数	2

② 逻辑表达式。

算术表达式用以表达数字之间的计算，逻辑表达式则通常用以表达对象之间的关系，例如大小关系、包含关系等。

✓ 逻辑表达式的值。

逻辑表达式的值只有两种，分别是True和False，表示这个逻辑是否成立，即表示这件事情的真和假。通常用逻辑表达式进行逻辑判断，若逻辑成立，则逻辑值为True；否则，逻辑值为False。在Python的交互式Shell中输入一个逻辑表达式，按【Enter】键后可得到该逻辑表达式的值。例如，1>2是一个“假”命题，因此逻辑表达式1>2的值为False：

```
>>> 1>2False
```

✓ Python关系运算符。

除了表示“大于”的关系运算符“>”，Python中还有其他一些常用的关系运算符，见表。

关系运算符	逻辑表达式示例	描　述	值
>	1>2	1大于2	False
<	1<2	1小于2	True
==	1==2	1等于2	False
>=	1>=2	1大于等于2	False
<=	1<=2	1小于等于2	True
!=	1!=2	1不等于2	True
in	'e' in 'hello'	字符串'e'包含在字符串'hello'中	True

✓ Python逻辑运算符。

对于一些更复杂的逻辑，例如“并且”“或”“非”这样的复合逻辑，可以用逻辑运算符来表达。在Python中，逻辑运算符为and（并且）、or（或）、not（非）。

在复合逻辑中，表达式的值是True或False，不仅取决于表达式中各个条件的真假，还取决于连接这些条件的逻辑运算符是什么，具体分析见表。

	A和B均为True	A为True，B为False	A为False，B为True	A和B均为False
A and B	True	False	False	False
A or B	True	True	True	False
not A	False	False	True	True

当条件A和条件B同时满足，表达式“A and B”的值为True；当条件A和条件B中有一个条件不满足，表达式“A and B”的值为False。例如，10<20<30是一个复合逻辑：10<20且20<30，由于10<20和20<30都成立，因此10<20<30的逻辑值为True：

```
>>> 10<20 and 20<30
True
```

3．变量和赋值

在Python程序中，为了让计算机“记住”某个信息，可以通过创建“变量”，将信息保存在计算机里一个负责“记忆”的地方—内存。

（1）变量：对象的名字。

变量指向内存中某个数据对象存储的位置，我们可以把变量看作对象的名字或代号。例如，在hello.py程序中，将用户输入的名字信息存储在了变量name中，变量name就代表着用户输入的名字。

```
>name = input('我叫: ')
```

变量命名规则一般有以下4条：

① 变量名必须以字母或下画线“_”开头；

② 变量名中其他字符必须是字母、数字或者下画线“_”，这意味着一个名字中不能用空格（例如：my name就不是一个合法的变量名，因为中间有空格）；

③ 变量名区分大小写，abc和ABC是两个不同的变量；

④ 变量不能与Python的关键字、内置函数名、内置模块名等重名，如不可给变量取名为int、print等。

通常，一个有意义的名字会比一个没有意义的名字更受青睐。当变量名字的含义较复杂时，常采用驼峰命名法和下画线命名法进行命名。

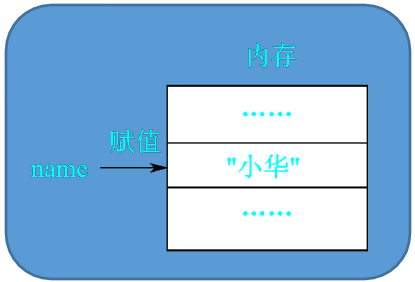
例如，变量名userName、printNum采用的是驼峰命名法，变量名user_name、print_num采用的是下画线命名法。

（2）赋值：创建变量的过程。

“赋值”是创建变量的过程，当需要程序记住某个信息（或对象）的时候，就在内存中开辟一块地方，把这个对象放在里面，同时，给这个对象取一个名字，并让这个名字指向对象所在的位置。在Python中，一个变量只有被赋予了一个具体的值才会被创建，即只有变量名和内存中的某个对象建立起联系后才算赋值成功。

在Python中，赋值操作是通过赋值操作符“=”来完成的。这里的“=”不是数学上的“等于”，而是一个赋值符，其作用是把右边的内容赋值给左边的变量，使对象和变量名建立起对应联系。

例如，name = "小华"，将字符串"小华"赋给了变量name，name将指向内存中字符串"小华 "存储的位置，字符串"小华"就是变量name的值，如图所示。当然，如果用户输入了其他内容，name将被赋为其他值，并指向对应对象所在的存储位置。



（3）变量类型。

在Python中，变量可以被赋予为不同类型的值，而被赋予了不同类型值的变量，是不同类型的变量。例如，在下面的程序中，变量name的值是字符串' 小华'，因此变量name是字符串变量；变量age的值是整数15，因此变量age是整数变量。

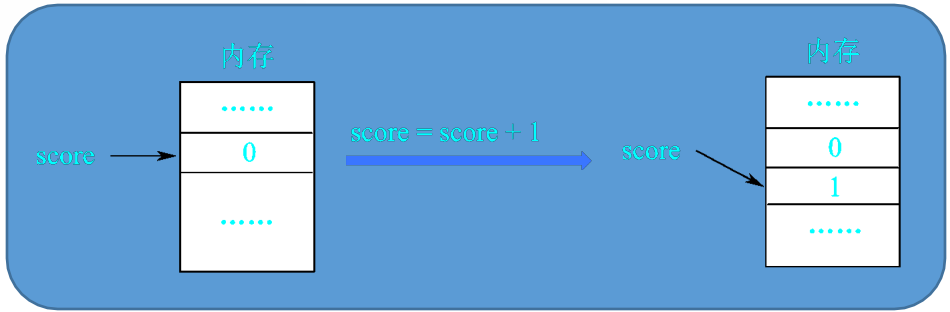
```
name = '小华'        age = 15
```

（4）访问变量的值。

赋值操作完成之后，变量就代表内存中存储的该数据对象，因此，可以通过变量名来访问具体的值。例如，在hello.py的第3行代码中，通过变量name输出用户的名字。

在程序中，变量的值可以改变。当一个变量被赋予了一个新值，它就会指向新值所在的位置。例如，创建变量score来记录玩家得分，初始化score为0，每次游戏获胜时，score被重新赋值为score+1，程序将先计算出右边score+1的和为1，然后将score变量指向内存中存储数字1的地方，而不再指向内存中存储数字0的地方，如图所示。

```
print('Hello，你叫什么名字？')name = input
('我叫：')print(name + '，很高兴认识你!
')
```



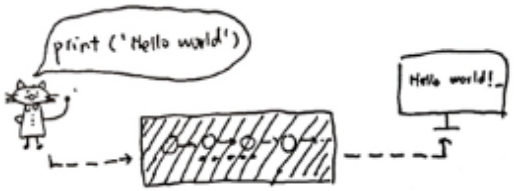
4．函数和模块

在Python中，函数和模块都可以看作Python的“工具”，它们让程序设计变得更加简单和方便。

（1）函数。

① 函数的概念。

在hello.py程序中，print()和input()都是Python的函数，分别用于打印输出和键盘输入。函数是将一系列复杂的操作或一系列连续的指令打包，“封装”成一条指令，这样，在程序的其他地方，就可以根据需要随时调用这条函数指令。就像我们去餐厅吃饭，顾客只需要“点菜”，厨师就会做出美味佳肴，而顾客自己不需要亲自去执行做菜的每个步骤。



要在屏幕上打印出一行文本，计算机其实需要进行很多复杂的操作，但是由于这是一个很常用的功能，因此Python的设计者便将所有用于打印输出的底层指令“封装”起来，并将其命名为print，这样，我们只需调用一条指令—print()，就能自动调用函数中被封装的底层代码了，如图所示为调用函数示意图。

② 函数参数。

为了让函数的使用更具有灵活性，我们可以向函数传递参数。函数的参数就像做菜时加的调料，加入不同的调料，就会烹饪出不一样的味道。在数学课上，f(x)=x+1也是一个函数，x的值不同，得到的f(x)就不同。

比如，print()函数是一个带参数的函数，其功能是打印输出一行文本并自动换行，括号里的参数可以为空，也可以是一个文本或一个数字，表示要打印输出的内容。

③ 函数的返回值。

✓ 没有返回值的函数

没有返回值的函数只需完成一个或多个操作，完成操作后不必向调用它的地方返回任何数据。例如，print()函数是一个没有返回值的函数，它只需要完成“输出”操作即可。

✓ 有返回值的函数

有返回值的函数执行完成一系列操作后，会将函数的执行结果返回到调用它的地方。例如，input()函数是一个有返回值的函数，它在获取到用户输入的数据之后，会将输入内容以字符串形式返回到调用它的地方。因此，有返回值的函数也可以被看作是一个数据对象，可以在程序中进行赋值和运算操作。在hello.py程序中，第二行代码就是将用户输入的“名字”赋给name变量。

```
name = input("我叫: ")
```

④ 自定义函数。

print()、int()等是Python的设计者已经定义好的函数，称为“内置函数”。在Python中，也可以根据需要自己定义函数，并在程序中调用它们，自己定义的函数是“自定义函数”。

- ✓ def是函数定义的关键字，取自英文单词definition（定义）；
- ✓ 函数定义时需指明函数的名字，好比为一道菜取一个名字；
- ✓ 括号中可以设置函数的参数，多个参数用逗号“，”隔开；
- ✓ 冒号“:”表示即将开始定义函数内部的语句；
- ✓ 封装在函数里的代码有相同的缩进，表示它们属于这个函数，是同一个语句块。语句块是具有相同缩进的一组连续语句。在Python中，用“：”标志一个语句块的开始，在其他程序设计语言中，使用特殊单词（如Begin和End）或字符（如“{”和“}”）标志一个语句块的开始和结束。

```
def 函数名(参数1,参数2……):
    语句1
    语句2
    ……
```

函数调用

和调用Python的内置函数一样，自定义函数也通过函数名和括号来调用，函数名需与函数定义时的函数名保持一致。如果定义了参数，可以在调用时传入参数。

```
函数名(参数1,参数2, ……)
```

例如，自定义一个“做牛肉面”的函数makeNoodles()，并设置辣度参数spicy，然后调用该函数，传入辣度选项为“特辣”，输出做一碗“特辣”牛肉面的步骤。

运行输出：

noodles.py

```
01.def makeNoodles(spicy):
02.    print('开始制作牛肉面')
03.    print('烧水')
04.    print('煮面')
05.    print('配味：' + spicy)
06.    print('加汤')
07.    print('加牛肉')
08.    print('牛肉面做好了！')
09.makeNoodles ("特辣")
```

函数定义：教“厨师”做牛肉面的方法步骤。

运行输出：开始制作牛肉面
烧水煮面配味：特辣加汤加牛肉
牛肉面做好了！

上面的程序中，如果只定义makeNoodles()函数而不调用它，即没有第9行代码，程序还会执行函数里的内容，将做牛肉面的步骤打印出来吗？为什么？

“函数定义”只是教会了厨师做面的方法，“函数调用”才是下达“做面”指令，因此，若程序定义了函数而没有调用函数，函数里的代码也不会被执行。另外，由于程序是从上到下执行的，当它看到函数定义时，才会将其记录下来，完成函数定义的“注册”。因此，Python中的函数定义必须放在函数调用之前，就像让厨师做面之前，得先教会厨师怎么做面。

⑤ 函数的妙用

✓ 避免代码冗余

函数是对代码的一种封装，我们只需要在函数中写一次代码，就可以对这些代码进行重复利用，非常方便。例如，在noodles.py中，将做面的步骤封装到makeNoodles()函数中，之后每次需要“做面”时，只需调用函数，就可以执行所有“做面”的步骤。

✓ 方便程序修改

当代码需要修改时，只需在函数中修改一次，而不用在每一个需要做这件事的地方修改代码。例如，当“做面”的步骤发生更改，或者增加了面的一些规格选项，如酸度、是否加香菜等，只需要在函数定义中修改一次即可。如果不使用函数，则要对每一次“做面”的程序进行修改。

实现模块化编程

函数将多行代码封装成一行语句，通过函数名能很容易地知道程序在做什么，增加程序的可读性。在更复杂的程序中，可将任务分解为几个子任务，若我们将每个子任务的实现代码封装成函数，将每个子任务看作一个“模块”，则可实现模块化编程。当程序有问题时，我们只需关注是哪个模块出现了问题，再进行深入排查，而不必在整个程序中进行“大海捞针”般的工作。

在设计较复杂的程序时，一般采用自上而下的方法，将问题划分为几个部分，各个部分再进行细化，直到分解为较好解决的问题为止。模块化编程，简单地说就是程序的编写不是一开始就逐条录入计算机语句和指令，而是首先用主程序、子程序（函数定义中的代码是子程序，函数调用的代码是主程序）等框架把程序的主要结构和流程描述出来，并定义好各个框架之间的输入输出关系。

就像“搭积木”一样，一块积木就是负责一个功能的小程序块，模块化编程就是将这些积木有组织地搭建起来。模块化的目的是降低程序的复杂度，使程序设计、调试和维护等操作简单化。

事实上，我们也常常用“模块化”的思想解决生活中或学习中的问题。比如我们会将自己的生活安排分为学习模块、锻炼模块、休息模块等，在不同模块中再进行不同的安排。模块化，能提高我们的学习和工作的效率。

（2）模块。

① 模块是什么?

在本章最开始的“抛硬币”问题中，random随机数模块就是Python中的一个模块。模块的本质是一个Python可执行文件，里面有许多已经定义好的功能相似的函数、变量、类及一些可执行代码。模块就好比一个工具箱，里面装着各种各样的工具，可供我们使用。例如，random模块这个“工具箱”里有很多与随机数相关的“工具”，randint(a,b)函数就是其中一个“工具”，用于获取a～b之间的一个随机整数。

random模块是Python自带的一个模块，称为Python的内置模块。除此之外，我们也可以自己设计模块，称为自定义模块；由其他程序员设计好的模块，则称为第三方模块，或第三方功能库。

② 模块的分类

✓ 内置模块

Python内部早已设定好模块，可直接导入使用。如random模块（随机数“工具箱”）、time模块（时间“工具箱”）、math模块（数学“工具箱”）、tkinter模块（图形界面开发“工具箱”）等。

✓ 自定义模块

由编程者自己设计的.py文件作为模块。

✓ 第三方模块

其他程序员设计的并开放给大家使用的模块，需先下载至本地或通过网络连接该模块，再导入使用。如matplotlib绘图模块、pandas数据分析模块等。

③ 模块的使用

模块是一个“工具箱”，因此，要使用模块中的“工具”，必须先把“工具箱”买回来。在Python程序中，使用模块之前应先将模块导入到程序中。

第一步：导入模块。

将模块导入到程序中，其目的是将模块中的程序代码导入到自己的程序中。在Python中，通过关键字import导入整个模块：

```
import random
```

第二步：使用模块。

模块导入成功之后，就像是把“工具箱”买回家了，里面的“工具”自然都可以使用。将模块导入程序后，模块中的代码都可以看成是程序的代码，模块中定义好的变量、函数等代码都可以调用。

模块中函数或变量的调用方式：“模块.成员”。其中，“.”是Python中的成员访问运算符，通过“模块.成员”，程序才能知道这些函数和变量来自哪个“工具箱”，这样不仅便于模块管理，也解决了模块中的函数或变量与程序中已有的函数或变量可能重名的问题。

例如，导入random模块并调用random模块中的randint(a,b)函数。random(1,2)表示要么产生一个1，要么产生一个2：

```
>>> import random
>>> random.randint(1,2)
1
>>> random.randint(1,2)
2
```

5．判断和循环

程序主要由三种结构组成，分别是顺序结构、选择结构和循环结构。在程序设计语言中，条件控制语句和循环语句让程序能够实现更加复杂的逻辑，完成更加复杂的任务。

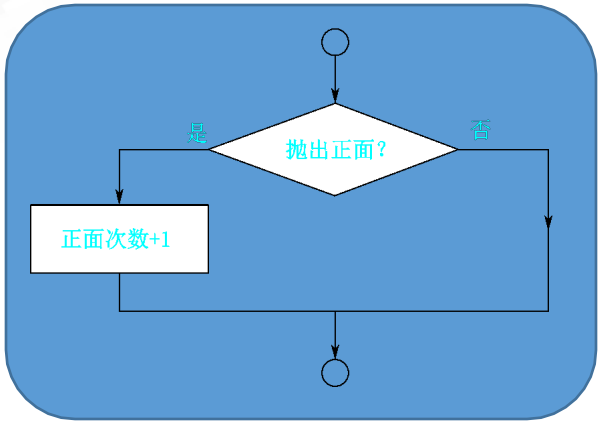
（1）条件控制语句。

条件判断在我们的生活中无处不在，在“抛硬币”问题中：

如果抛出正面，

则正面次数+1。

用程序流程图来表示这个逻辑，如图所示。



在Python中，有一个特殊语句—条件控制语句，用于进行条件判断。条件语句就像一个岔路口，程序会首先在岔路口进行条件判断，然后根据条件判断的结果进入相应的道路，控制程序执行相应的语句。

在Python中，有一个特殊语句—条件控制语句，用于进行条件判断。条件语句就像一个岔路口，程序会首先在岔路口进行条件判断，然后根据条件判断的结果进入相应的道路，控制程序执行相应的语句。

① 条件控制语句的构成。

条件控制语句由关键字、判断条件，以及子句构成。if、elif和else都是Python中的关键字，所以当输入if、elif和else时，可以看到它们的颜色为橙色（不同的编辑器颜色可能不同）。在if语句、elif语句和else语句后有一个冒号“:”标志接下来缩进的语句为其子句。

```
if 条件1:
    <语句块1>

elif 条件2:
    <语句块2>

……

else:
    <语句块n>
```

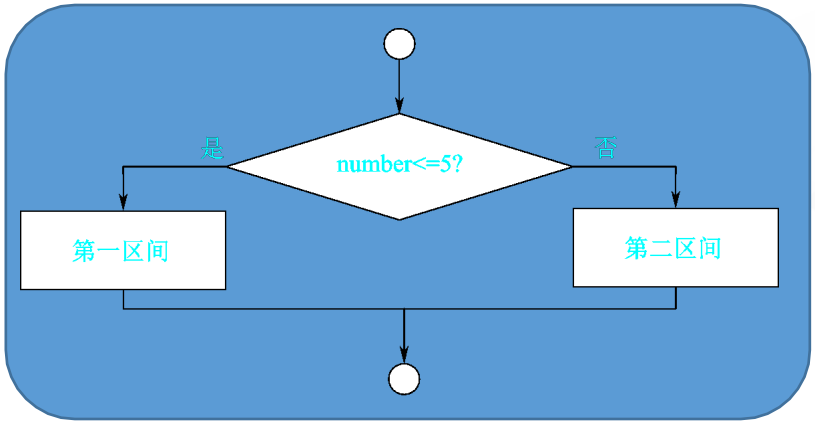
若条件 1 成立(条件 1 的逻辑值为 True):执行语句块 1 中的语句。

若条件 1 不成立，继续判断条件 2 是否成立，如果条件 2 成立(条件 2 的逻辑值为 True)：执行语句块 2 中的语句。

若前面的所有条件都不成立：执行语句块 n 中的语句。

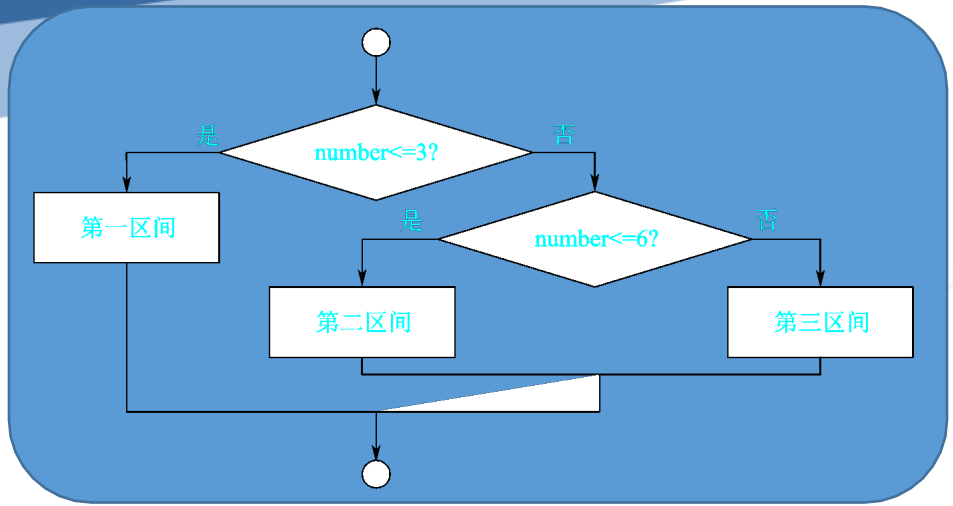
条件控制语句的运行让程序不再只有一条道路可以走，它能给程序提供多个选择，并根据不同的条件让程序走向不同的道路，得出不同的结果。在程序运行时，总是先判断if语句的条件，若if语句的条件不成立，再顺次判断后面的elif条件是否成立，若所有条件都不成立，才会执行else语句中的内容。

例1：将1～10的整数分为1～5、6～10这2个区间，获取1～10的随机数，并判断随机数所属区间。流程图如图所示。




```
checkNumber1.py
01.import random
02.number = random.randint(1,10)
03.if number<=5:
04.    print("第一区间")
05.else:
06.    print("第二区间")
```

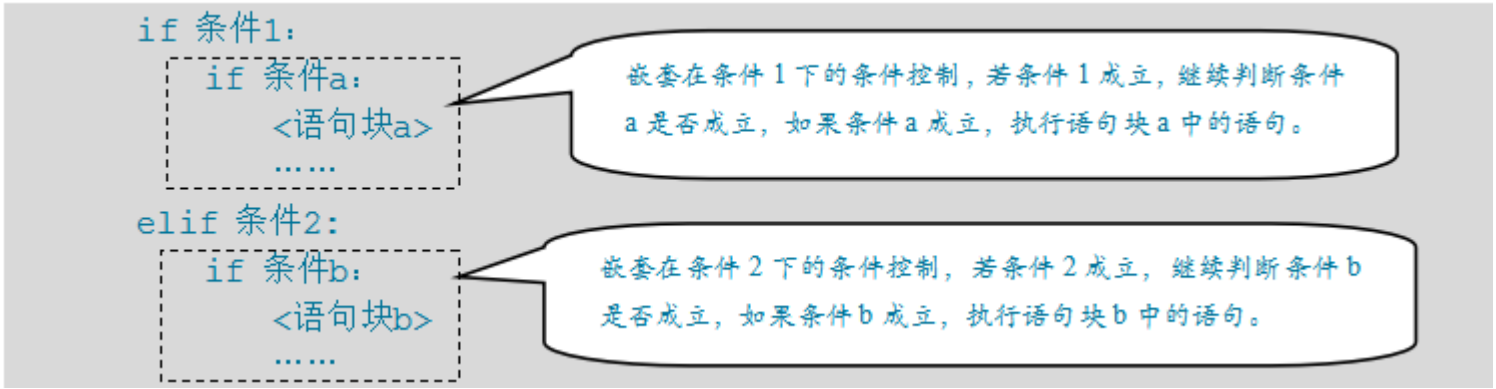
例2：将1～10的整数分为1～3、4～6、7～10这3个区间，获取1～10的随机数，并判断随机数所属区间。流程图如图所示。



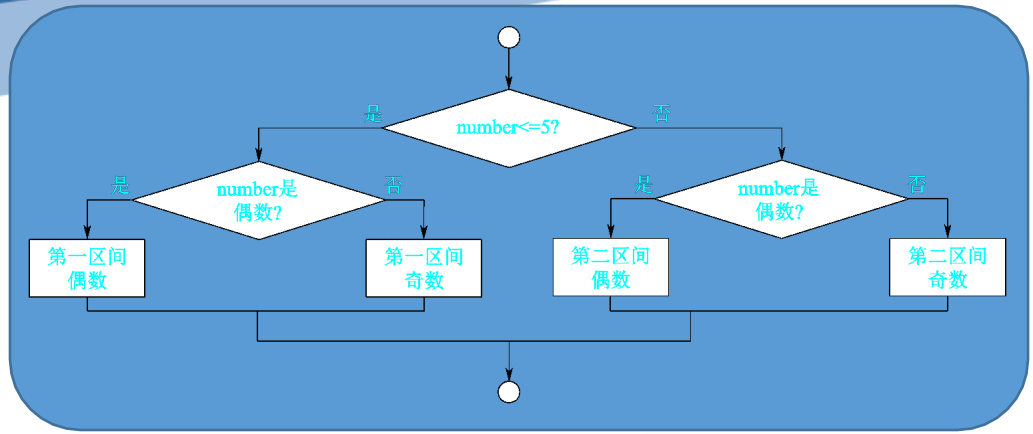

```
checkNumber2.py
01.import random
02.number = random.randint(1,10)
03.if number<=3:
04.    print("第一区间")
05.elif number<=6:
06.    print("第二区间")
07.else:
08.    print("第三区间")
```

②条件控制语句的嵌套。

像树枝从主干分了岔之后，在枝干上也可以继续分出更小的枝干一样，在复杂的条件判断中，有时需要在某个判断条件下“嵌套”新的条件判断。所谓“嵌套”，就像俄罗斯套娃一样，把一个条件控制语句嵌套在另一个条件语句块里，嵌套在里面的条件控制语句看作一个整体。在Python中，根据代码的缩进量来控制语句的嵌套层级。



例3：将1～10的整数分为1～5、6～10两个区间，获取1～10的随机数，并判断随机数所属区间及其奇偶性。流程图如图所示。




```
checkNumber3.py
01.import random
02.number = random.randint(1,10)
03.if number<=5:
04.    if number%2 == 0:
05.        print("第一区间的偶数")
06.    else:
07.        print("第一区间的奇数")
08.else:
09.    if number%2 == 0:
10.        print("第二区间的偶数")
11.    else:
12.        print("第二区间的奇数")
```

内层判断
内层判断
外层判断

（2）循环语句。

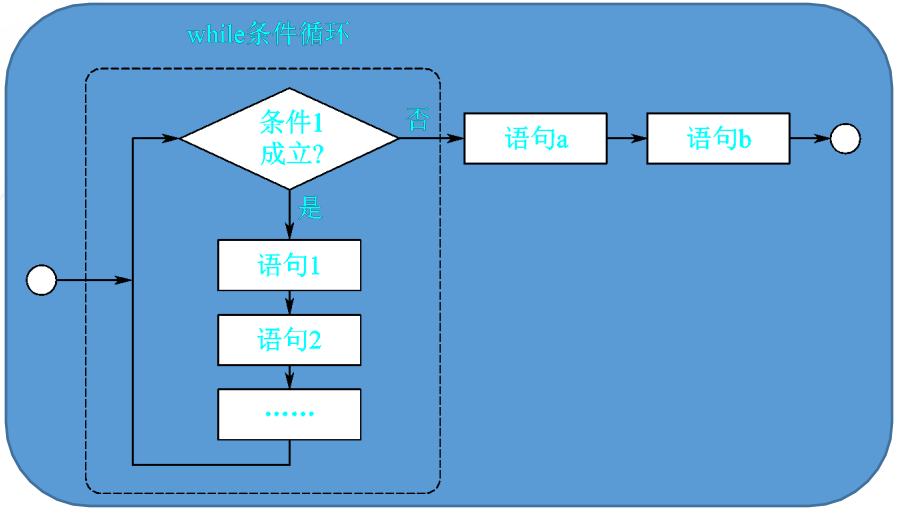
① while条件循环语句。

条件循环语句由while关键字、判断条件及循环体组成，其流程图如图所示。在while语句后面，以一个冒号“:”标志其子句的开始。

```
while 条件1:
        语句1
        语句2
        ……
语句a
语句b
……
```

循环体

循环条件：条件满足方可执行继续循环，条件不满足时结束循环



while是“当……时”的意思。当程序运行到while语句时，首先判断条件1是否成立，若条件1成立，则执行一次循环体（语句1→语句2→……）。循环体执行一次之后，再次判断条件1是否成立，若条件1依然成立，则再执行一次循环体，如此循环往复。直到条件1不被满足时，程序不再执行循环体，结束循环，继续执行后面的语句（语句a→语句b→……）。

在“抛硬币”问题中，抛100次硬币就是一个循环任务，循环条件为“实验次数是否达到100次？”，在开始循环之前，将实验次数cnt变量初始化为0，进入循环后，每次抛硬币cnt都自增1，因此，循环次数为100次。

tossCoin.py

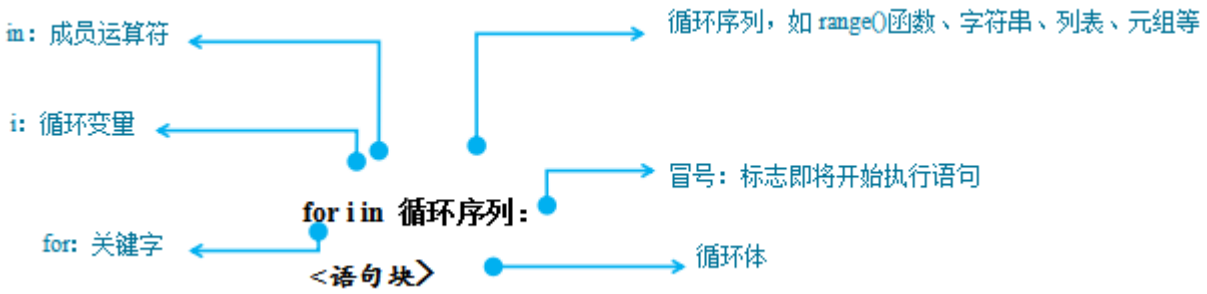
```
01.import random

02.up_n = 0 #记录扔出正面的次数
03.total_n = 100 #代表实验总次数
04.cnt = 0 #记录实验次数

05.while cnt<100:
06.    result = random.randint(0,1)
07.    print(result)
08.    if result==1:
09.        up_n = up_n +1
10.    cnt = cnt + 1

11.p = up_n/total_n
12.print('扔出正面的频率为: '+str(p))
```

② for计数循环语句。

for循环和while循环一样，都能控制一个程序段重复连续执行多次。但for循环的循环条件比较特殊，它根据循环执行的次数来判断是否继续循环，因此for循环也称为计数循环。

for循环由关键字for、循环变量、循环序列和循环体组成，其结构如图所示。

✓ 循环序列：for循环中的“储物柜”中有序存放着一组数据。每次循环，循环序列中的数据将被依次取出，因此，循环执行的次数就是循环序列中元素的个数。序列型数据都可以作为循环序列，如range()函数、字符串、列表、元组等，其中，range()函数是Python的内置函数，它将产生一个数字序列。



✓ 循环变量：for循环中的“计数员”，本质是一个变量。每次循环时，都从循环序列中取出下一个元素，并将该元素赋值给循环变量i，当序列中的元素取完时，返回False，循环结束。

✓ 循环体：for循环中的“执行者”，每次循环中被执行的语句（块）。

在“抛硬币”问题中，由于“抛硬币”的过程要执行100次，即以循环次数来作为循环的判断条件，因此，也可以用for计数循环来实现：

tossCoin.py

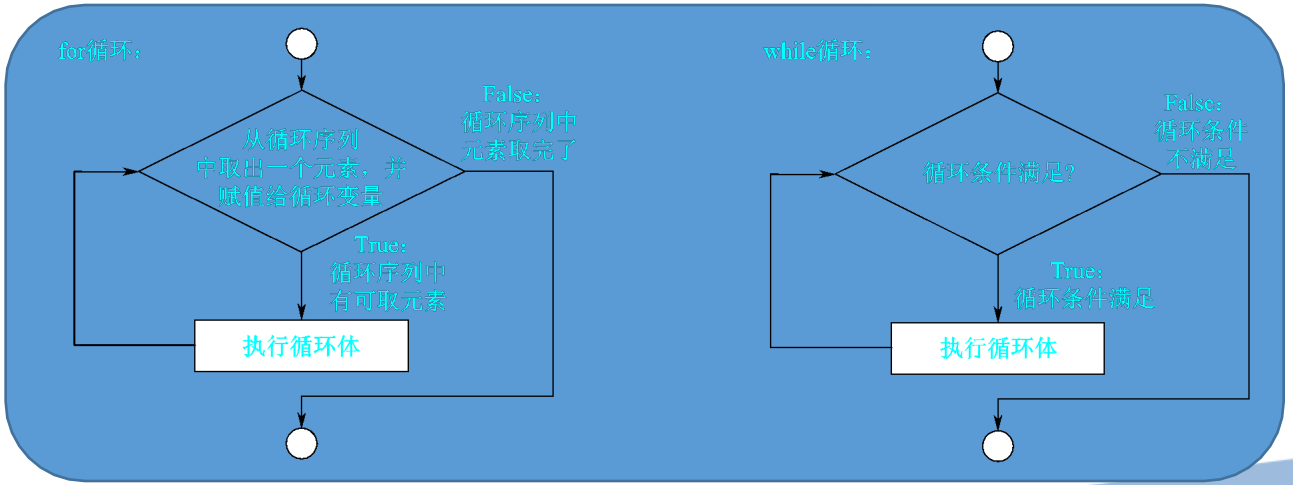
```
01.import random

02.up_n = 0 #记录扔出正面的次数
03.total_n = 100 #代表实验总次数

04.for i in range(100):
05.    result = random.randint(0,1)
06.    print(result)
07.    if result==1:
08.        up_n = up_n +1

09.p = up_n/total_n
10.print('扔出正面的频率为：'+str(p))
```

③ while条件循环与for计数循环。

for循环和while循环都属于循环语句，它们有何共同之处？有何不同之处？其适用场景分别是怎样的？

while条件循环和for计数循环都属于循环结构，都能控制程序循环执行一段代码，且对程序的控制都依赖于条件判断，若条件满足则执行循环体，否则循环结束。

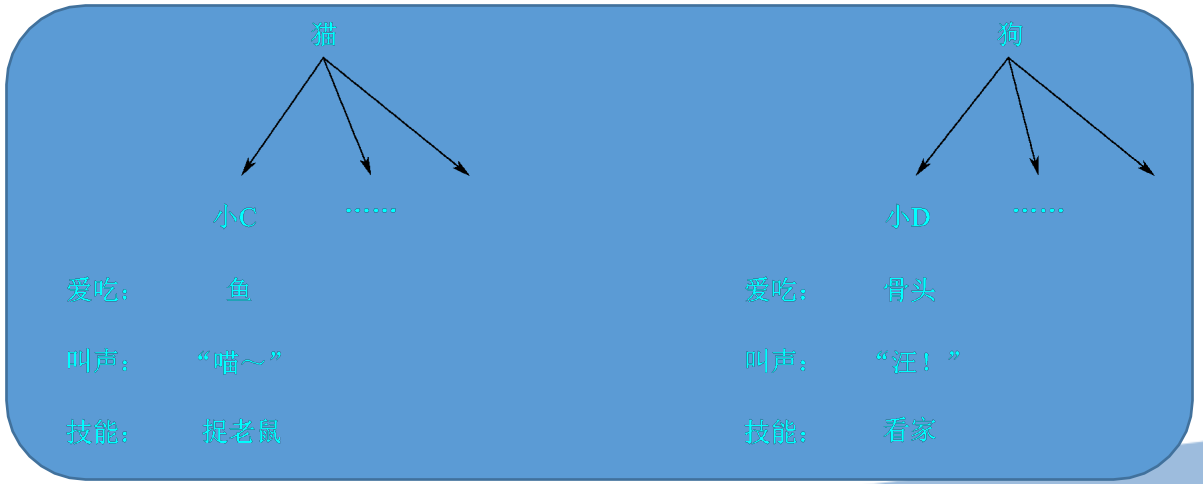
不同的是，for循环需要对序列进行遍历，且其条件判断比较特殊，为“循环序列中的元素是否被取完？”；而while循环可以对任何条件进行判断，应用更为广泛。两种循环语句的对比如图所示。



6．面向对象

（1）类和对象。

类和对象的概念来源于我们的生活。例如，小C是一只猫，它喜欢吃鱼，叫声是“喵~”，会捉老鼠；小D是一只狗，它喜欢啃骨头，叫声是“汪！”，会看家。而世界上除了小C和小D，还有其他的猫和狗，并且所有的猫都有相同的属性（如喜欢吃鱼、叫声是“喵~”）和技能（如会捉老鼠），所有的狗也都有相同的属性（如喜欢啃骨头，叫声是“汪！”）和技能（如会看家）。其示意图如图所示。



可以说，“猫”是一个“类”，小C是这个类中的一个“对象”或“实例”；“狗”也是一个“类”，小D是这个类中的一个“对象”或“实例”。因此，类（class）是具有相同属性和相同操作的一些对象的“模板”，对象（object）则是根据类这个模板创造出来的一个具体的“实例”。

Python中每个数据都可看作一个对象，而每个数据的类就是它所属的数据类型，不同类型的数据属于不同的类，具有不同的特征。比如，123是一个整数，它是整数这个类的一个对象，可以进行数学运算；'123'是一个字符串，它是字符串这个类的一个对象，可以进行文本操作。

（2）面向对象和面向过程。

“面向对象”和“面向过程”本质都是解决问题的一种思想方法。相比之下，面向过程关注解决问题的一系列步骤；面向对象则是一种以事物为中心的思想方式，事物就是对象，具有一定的属性和方法。但在解决实际问题时，面向对象的方法也含有面向过程的思想，可以说面向过程是一种基础的方法，它考虑的是实际的实现。

举个生活中常见的例子：玩五子棋游戏。面向过程的设计思路是先分析问题的每个步骤：开始五子棋→黑子先走→黑子落子→判断输赢→白子落子→判断输赢→黑子落子→判断输赢……最终结束。将每个步骤用不同的方法来实现，这就是面向过程。

如果使用面向对象的方法来解决问题，整个五子棋便可以分为：黑方、白方、棋盘、裁判这四个对象。其中，黑方和白方负责接收两名选手的输入，并告知棋盘，棋盘负责显示当前棋局中棋子的布局情况，同时，裁判来对棋局的胜负进行判定。

说一说

请结合“双十一”网络购物满减活动，谈谈程序设计思想的应用。

5.2.2 编辑、运行和调试简单程序

掌握了程序设计的基础知识后，小华找到堂兄，想要学习怎样设计程序来解决他还没想明白的打折问题。堂兄告诉小华，和之前的“抛硬币”问题一样，要用程序设计来解决打折问题，首先得对这个问题进行分析，然后设计相应的算法，并根据所设计的算法来编写程序。运行程序之后，还可以对问题进行进一步的反思和迁移，对程序进行优化和改进，并将解决问题的办法迁移到其他问题的解决中。这一节里，我们就来编写程序解决打折问题，体会通过程序设计解决问题的过程和方法。

1．问题描述

假设，某商店出售的某件商品，成本为4.5元，售价为15元，在第一周的营业中，销售量为100个。在接下来的一周中，将进行打折活动。经过市场调查，发现商品每降价1元，顾客就会增加20%的购买意愿，这意味着商品每降价1元，销售量就可能增加20%。

若最低折扣为5折，在接下来一周的打折活动中，为让商店获得最高利润，应该打几折?

2．问题分析

商品每降价1元，销售量就增加20%，根据这一市场调查结果，可以根据以下公式的计算步骤，计算出每种打折情况下活动期间的总利润：

折后价=原价×折扣×0.1

活动期间销售量=上周销售量×[1+20%×（原价-折后价）]

活动期间总利润=（折后价–成本）×下周销售量

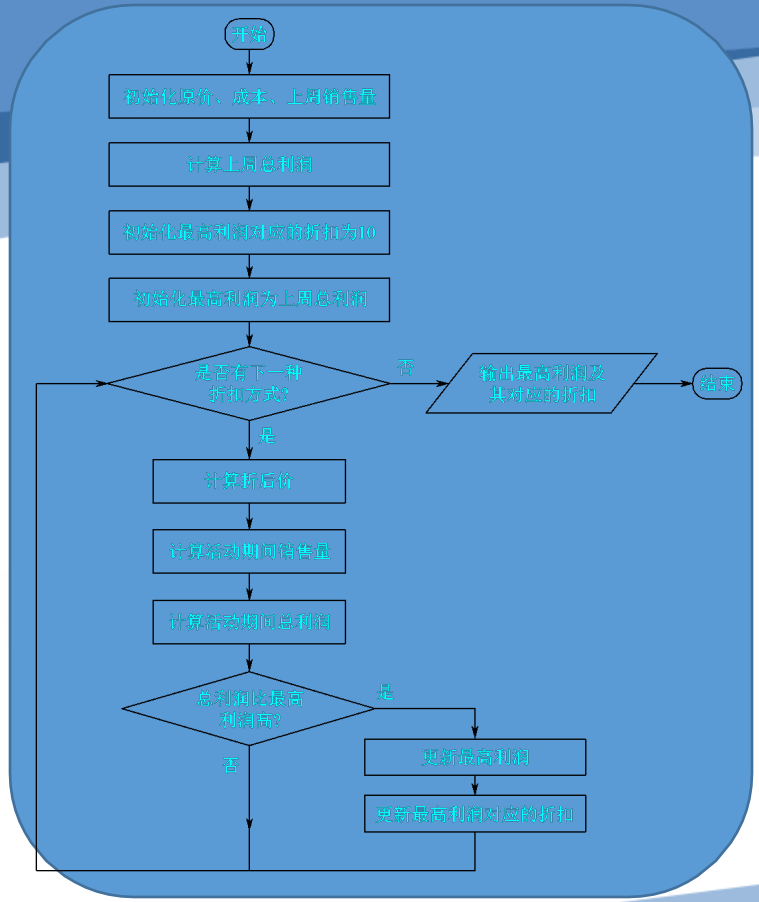
为各个数据创建变量见表所示。

数　据	变　量
原价	price
成本	cost
上周销售量	sellNum
折扣	discount
折后价	discPrice
活动期间销售量	discSellNum
活动期间总利润	discTotalProfit

知道了活动期间每种打折情况下总利润的计算方法，接下来，只需要计算出所有打折情况下，活动期间的总利润，并找到其中总利润最高的打折方法。

3．算法设计

在拍卖活动中，工作人员会先给出一个底价，然后让所有参与者在此基础上报价，每当出现比当前最高价更高的价格时就更新当前最高价……直到没有人再报出更高的价格，就以当前最高价成交。寻找最佳打折方法的过程也一样，首先创建两个变量分别存储最高利润和最高利润对应的折扣，将最高利润对应的折扣初始化为10，表示不打折，最高利润初始化为不打折时的一周利润。然后从5折开始递增折扣，依次计算每种打折方式下的总利润。如果某种折扣方式下，总利润比最高利润高，则更新最高利润和对应的折扣……直到计算并比较完所有的折扣方式下的总利润，也就找到了所有方式下的最高利润及其对应的折扣方式。将这一算法用流程图表示，如图所示。


开始
初始化原价、成本、上周销售量
计算上周总利润
初始化最高利润对应的折扣为10
初始化最高利润为上周总利润
是否有下一种折扣方式?
否
输出最高利润及其对应的折扣
结束
是
计算折后价
计算活动期间销售量
计算活动期间总利润
总利润比最高利润高?
是
否
更新最高利润
更新最高利润对应的折扣

可以看出，计算每种打折情况下的总利润的过程是一个循环结构，并且在每一次循环中，还嵌套了一个选择结构，用于判断是否需要更新最高利润及其对应的折扣。

接下来，就可以根据所设计的算法，编写程序来解决打折问题了，示例程序如下。

```
discount.py01.price = 1502.cost = 4.503.sellNum = 100
04.totalProfit = (price - cost) * sellNum 05.discOfBigProfit
= 1006.bigProfit = totalProfit 07.for discount in
range(5,10):08.    newPrice = price * discount * 0.109.
newSellNum = sellNum * (1 + 0.2 * (price - newPrice))10.
newTotalProfit = (newPrice - cost) * newSellNum11.    if
newTotalProfit > bigProfit:12.        bigProfit =
newTotalProfit13.        discOfBigProfit = discount 14.print
('打'+ str(discOfBigProfit) + '折，预计利润最高：' +
str(bigProfit) + '元。')
```

注：第7行，range(a,b)函数是Python的内置函数，将返回一个[a,b)左闭右开的整数序列，如range(5,10)将返回数字序列5,6,7,8,9。在for循环的遍历中，循环变量discount将依次被赋值为5,6,7,8,9。这样，就能通过for循环遍历每种折扣，并计算和比较该折扣方式下的总利润了。

反思与迁移

在打折问题中，商品只有1项，但如果商店有多个商品，每种商品的成本和原价都各不相同，如何计算不同打折情况下，所有商品的总利润呢？假设：商店有4种商品，每种商品的成本、原价，以及上周销售量见表5-11，请设计算法编写程序计算出商店上周营业的总利润。

商　品	成本（元）	原价（元）	上周销售量（个）
商品1	4.5	15	100
商品2	8	20	120
商品3	8.5	20	150
商品4	9	22	120

最简单的办法，我们可以依次计算出4种商品的利润，然后将4种商品的利润相加。如下所示：

```
calProfit.py01.price1 = 1502.price2 = 2003.price3 =
2004.price4 = 22 05.cost1 = 4.506.cost2 = 807.cost3 =
8.508.cost4 = 9 09.sellNum1 = 10010.sellNum2 = 12011.sellNum3
= 15012.sellNum4 = 120 13.totalProfit1 = (price1 - cost1) *
sellNum114.totalProfit2 = (price2 - cost2) *
sellNum215.totalProfit3 = (price3 - cost3) *
sellNum316.totalProfit4 = (price4 - cost4) *
sellNum417.totalProfit = totalProfit1 + totalProfit2 +
totalProfit3 + totalProfit418.print('上一周总利润为：' +
str(totalProfit))
```

可以看出，这种方法虽然比较直观，但是当商品数量更多时，代码会很长，不利于进行程序的编写和维护。因此，不妨将每种商品的各项信息都存储在列表里，使信息的表达和处理都更加方便。示例程序如下：

注：第5行中，range(4)是range(0,4)的简写，将返回一个长度为4的整数序列0,1,2,3。第5～7行，将循环4次，每次循环中，用oneProfit变量存储单个商品的利润，并将单个商品的利润叠加到总利润中（见第7行）。这样，当循环结束时，totalProfit的值就是所有商品的总利润。

```
calProfit.py01.prices = [15, 20, 20, 22]02.costs = [4.5, 8,
8.5, 9]03.sellNums = [100, 120, 150, 120] 04.totalProfit =
005.for i in range(4):06.    oneProfit = (prices[i] -
costs[i]) * sellNums[i]07.    totalProfit = totalProfit +
oneProfit 08.print('上一周总利润为：' + str(totalProfit))
```

说一说

在拍卖活动中，拍卖流程是如何的呢？类比该问题，请描述一个寻找最佳打折方法的解决办法。

5.2.3 了解典型算法

1．枚举算法

在解决打折问题的时候，我们实际上对每种打折方式进行了遍历，即计算了每种打折方式下的总利润，并判断该利润是否是最高利润。像这样将所有可能的情况遍历的方法，就是枚举，枚举算法是解决问题的一种典型算法，举例如下：

【例1】要将100元兑换成10元或者5元的币值，共有哪些兑换方式？

【例2】一个两位数密码，每一位数可以取0～9的任意数字，如何找到真正的密码？

【例3】田忌赛马问题中，让三个不同等级的马以什么顺序出场有可能获胜？

【例4】到达目的地有多条路径，哪条路径用时最短？

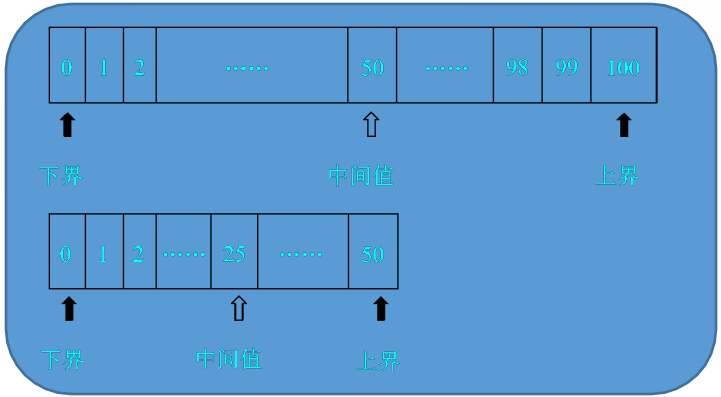
……

适用情况：研究对象可数且有范围，最优解可从该范围中逐一列举检验得到。

枚举算法的基本结构：确定范围—列举—检验。

2．二分查找法

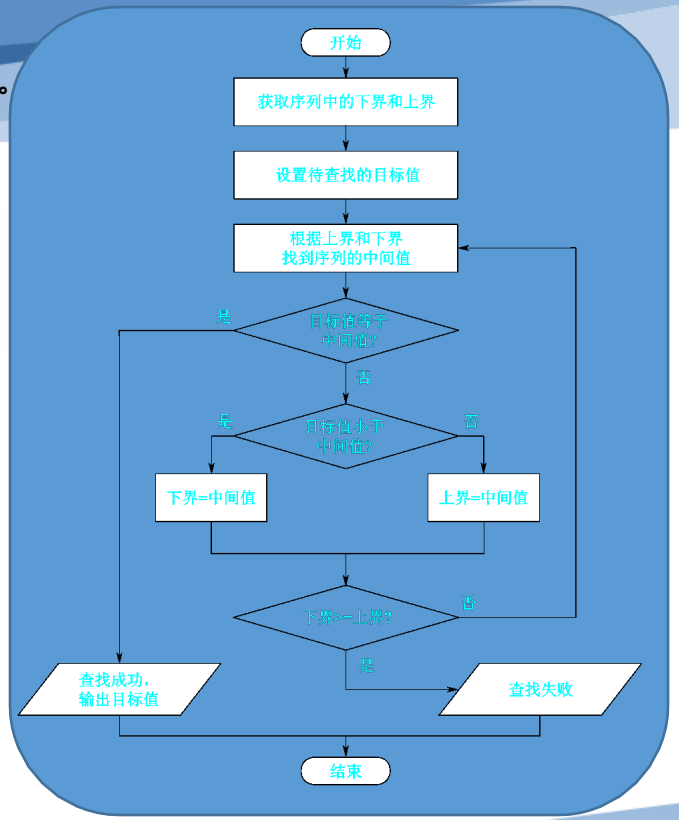
二分查找法也称折半查找，通常用于从一组有序数据中查找目标数据，利用二分查找法可以提高查找的效率。二分查找法的基本思路和玩猜数游戏的思路类似：假设要猜测的数字是0～100的一个数，最好的办法就是从0～100的中间数50开始猜，若目标数字小于50，就猜0～50中间的数字25；若目标数字大于50，就猜50～100中间的数字75；若目标数字刚好是50，结束猜数。重复此过程，不断缩小目标数字的范围，直到猜出正确的数字。猜数游戏示意图所示。



任务情景

临近节日，小华发现很多商店都在打折，促销活动吸引了很多的顾客去购买商品。小华心想：虽然打折活动让每个商品的价格降低了，但是销售量也增多了，那么商家最后获得的利润是比平时更高了还是更低了呢？如果他将来也开一家店，到了打折季的时候，为了获得最高的利润，怎么决定打几折呢？怀着这些疑问，小华找到了堂兄。

二分查找法的基本思路与此类似，用程序流程图表示该算法，如图所示。



说一说

逐个列举判断，看似简单，但有时也确不失为一种好办法。在生活中，我们也常常运用枚举的思想解决问题，比如轮胎漏气了，得一点一点挨着去找漏气的位置；为了挑出一碗红豆中的绿豆，也得一个一个去挑……而当我们把这种方法运用到程序编写中时，计算机凭借自己的高超的计算速度和准确度，可以帮我们解决很多问题。请想一想枚举算法还可以解决学习和生活中的什么问题呢？枚举算法的使用情况和基本结构是怎样的呢？

5.2.4 使用功能库扩展程序功能

1．拓展任务描述

请设计程序，让用户输入商店一周中每一天的销售量，并绘制柱状图分析商店一周内的销售情况。

2．问题分析和算法设计

通过input()函数可以获取用户的输入，对于一周七天的输入，可以通过for循环实现七天销售量的连续输入。

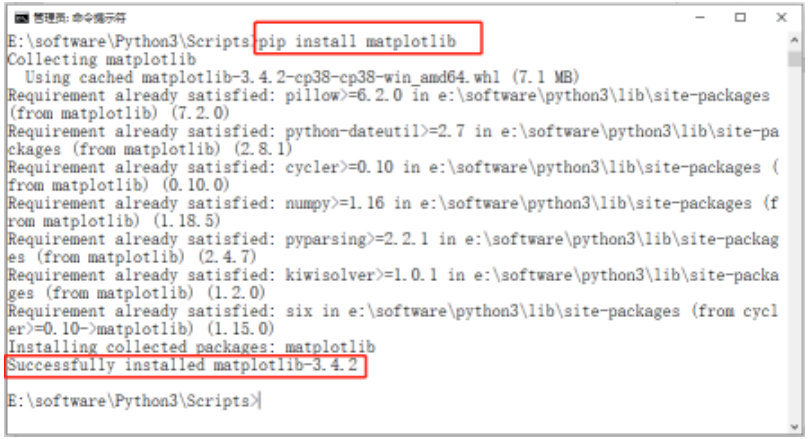
而要绘制一周销售量的柱状图，可以通过Python的第三方功能库matplotlib（2D绘图功能库，非常适合绘图）来实现，周一到周日作为柱状图的横轴数据，每天的销售量作为柱状图的纵轴数据。

3．matplotlib第三方功能库的安装和使用

（1）安装第三方功能库。

第三方功能库和内置模块不同，需要通过pip命令联网下载安装。搜索“cmd”，打开“命令提示符”窗口，如图所示。

接下来，将路径定位到Python文件夹下的Scripts文件夹下，例如，若Python安装在E盘下的“software”文件夹下，则首先在cmd命令窗口中将路径定位到“E:\software\Python3\ Scripts”。接着，输入pip安装命令“pip install 库名”，按回车键后即可开始进行第三方功能库的下载安装，直到提示“Successfully installed matplotlib-3.4.2”，表示安装成功。如图所示。



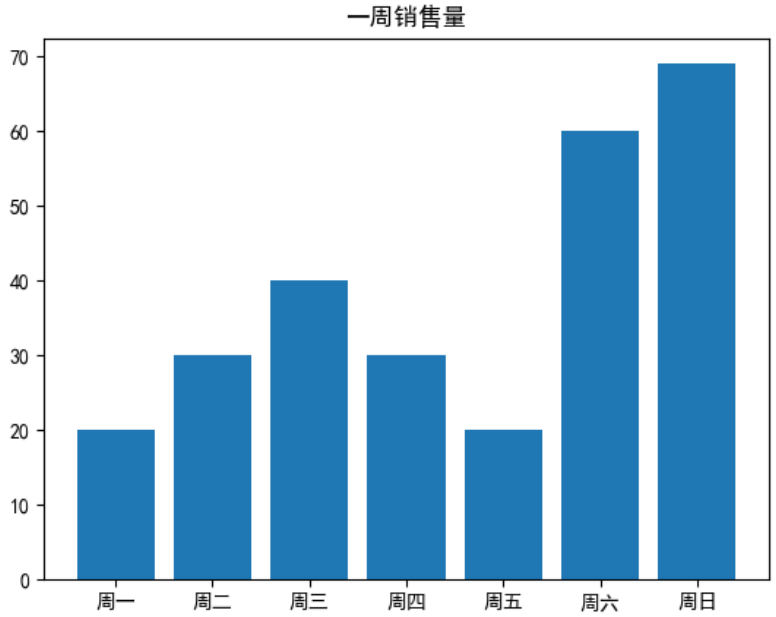
（2）使用matplotlib第三方功能库绘制柱状图。

Python第三方功能库的使用和内置模块一样，需要先将其导入程序中。通常，习惯在导入matplotlib功能库时为其取别名为mpl。另外，matplotlib库中包含多个子库用于不同的图形绘制，其中的pyplot子库是用于绘制柱状图的功能库，通常习惯为pyplot取别名为plt。

在绘制柱状图时，只需调用pyplot功能库中的bar()函数，并传入两个列表参数，分别作为柱状图的横轴和纵轴数据。例如，周一到周日的销售量分别为20、30、40、30、20、60、69，下面设计程序绘制这周的销售量柱状图。绘制出的柱状图如图所示.。

weekSell.py

```
01.import matplotlib as mpl
02.import matplotlib.pyplot as plt

03.mpl.rcParams['font.sans-serif']=['SimHei'] #用于正常显示中文标签

04.daySellNums = [20, 30, 40, 30, 20, 60, 69]
05.dayNames = ["周一", "周二", "周三", "周四", "周五", "周六", "周日"]

06.#绘制柱状图
07.fig=plt.figure()  #创建一个画布
08.plt.title("一周销售量")  #设置柱状图的标题
09.plt.bar(dayNames, daySellNums)
10.plt.show() #让图形显示出来
```



若需要由用户来输入每天的销售量，则初始化daySellNums列表为一个空列表，然后在for循环中获取用户输入的销售量，并将每天的销售量添加到daySellNums列表中，最后再利用matplotlib功能库绘制柱状图即可。

```
      weekSell.py 01.import matplotlib as mpl02.import
matplotlib.pyplot as plt 03.mpl.rcParams['font.sans-
serif']=['SimHei'] #用来正常显示中文标签 04.daySellNums =
[]05.dayNames = ["周一", "周二", "周三", "周四", "周五", "周六",
"周日"] 06.for day in dayNames:07.    daySellNum = int(input("
请输入" + day + "销售量："))08.
daySellNums.append(daySellNum)09.#绘制柱状图10.fig=plt.figure()11.plt.title
("一周销售量")12.plt.bar(dayNames, daySellNums)13.plt.show()
```

Python的一大特点是具有丰富的功能库，从上面的例子中可以感受到，利用功能库可以实现很多复杂的功能。所以，当我们需要实现一些复杂功能时，可以先了解是否已经有相关的功能库能够实现该功能，这样将大大提高编程效率。而学会检索和学习功能库的使用方法，是利用功能库解决问题的关键。

说一说

请说出利用Python功能库实现复杂功能的案例，如网络爬虫。