

# Python 程序设计基础

## (上)

同济大学计算机基础教研室

2022 年 8 月

# 目 录

第一章 Python 语言概述 .....	1
1.1 程序与程序设计语言 .....	1
1.2 为什么要学习程序设计 .....	4
1.3 Python 语言简介 .....	5
1.4 简单的 Python 程序 .....	7
1.5 Python 开发环境 .....	9
1.5.1 IDLE 简介 .....	9
1.5.2 Anaconda 简介 .....	11
1.5.3 第三方库 .....	12
1.6 程序调试与获取帮助 .....	15
1.6.1 程序调试 .....	15
1.6.2 获取帮助 .....	18
习题 .....	19
第 2 章 语言基础 .....	21
2.1 数据类型 .....	21
2.1.1 引例—计算地球质量的问题 .....	21
2.1.2 内置数据类型 .....	22
2.1.3 常量、变量和标识符 .....	23
2.2 数值数据的表示和处理 .....	24
2.2.1 数值数据常量 .....	24
2.2.2 算术表达式 .....	25
2.2.3 常用数值类函数 .....	28
2.3 字符串类型和处理 .....	31
2.3.1 字符串类型 .....	31
2.3.2 字符串的处理 .....	32
2.4 布尔类型和处理 .....	36
2.4.1 关系运算 .....	37
2.4.2 逻辑运算 .....	37
2.5 组合数据类型简介 .....	39
2.5.1 列表和元组 .....	39
2.5.2 字典和集合 .....	40
2.6 数据的输入和输出 .....	42
2.6.1 数据的输入 .....	42
2.6.2 数据的输出 .....	43
2.6.3 格式化输出 .....	43
2.7 综合应用 .....	45
习 题 .....	47
第 3 章 基本的控制结构 .....	52
3.1 顺序结构 .....	52
3.1.1 引例—人民币与美元兑换 .....	52
3.1.2 简单程序的结构 .....	53

3.2 选择结构.....	53
3.2.1 引例—体质指数 BMI 的计算 .....	54
3.2.2 选择结构语句.....	55
3.2.3 选择结构的嵌套.....	58
3.3 循环结构.....	59
3.3.1 引例—批量数据的处理 .....	59
3.3.2 for 语句.....	60
3.3.3 while 语句.....	63
3.3.4 循环结构的嵌套.....	65
3.3.5 循环结构的其他语句.....	66
3.4 综合应用.....	69
习题.....	72
<b>第四章 函数.....</b>	<b>77</b>
4.1 函数定义与调用.....	77
4.1.1 引例-多边形面积的计算.....	77
4.1.2 函数的定义和调用.....	78
4.2 参数及传递.....	81
4.2.1 位置参数及默认值参数.....	81
4.2.2 可变长参数.....	82
4.2.3 形参对实参的影响.....	85
4.2.4 变量的作用域.....	86
4.3 lambda 函数.....	88
4.4 递归函数.....	89
4.4.1 什么是递归.....	89
4.4.2 递归函数的设计 .....	90
4.5 综合应用.....	94
习题.....	99
<b>实验一 Python 程序设计基础 .....</b>	<b>103</b>
<b>实验二 顺序结构.....</b>	<b>104</b>
<b>实验三 选择结构.....</b>	<b>106</b>
<b>实验四 循环结构.....</b>	<b>108</b>
<b>实验五 函数.....</b>	<b>111</b>

# 第一章 Python 语言概述

计算机是通过执行程序完成各种各样的任务，而程序是用程序设计语言编写的。自 1954 年 FORTRAN 语言诞生以来，程序设计语言姹紫嫣红、繁花似锦，据统计发布了上千种语言，Python 是其中最华美绚丽的的一朵。

## 1.1 程序与程序设计语言

### 1. 程序

计算机系统之所以能完成各种各样的工作，是因为有了程序，那么什么是程序？程序又是如何设计的？

在日常生活中，大家都知道做任何事情要有个先后次序，这些按一定的顺序安排的工作被称为程序。例如，某学校颁奖大会的程序：

- ① 主持人宣布颁奖会开始，介绍出席颁奖会的领导
- ② 校长讲话
- ③ 领导宣布获奖名单
- ④ 领导颁奖
- ⑤ 获奖代表发言
- ⑥ 主持人宣布大会结束

可见，程序的概念是很普遍的，例如用汉语撰写的菜谱、会议的日程等。随着计算机的出现和普及后，程序这一概念成了计算机的专用名词，用于描述了计算机解决问题的过程。

计算机程序是为了解决某个特定问题使用某种程序设计语言编写的一组指令序列。例如，用 Python 编写的计算机角度转换弧度程序如下：

```
import math           #导入 math 模块
angle=float(input("请输入角度: ")) #输入角度，并且转换为 float 类型
radian=math.pi/180*angle #引用 math 库中的 pi，计算弧度
print("弧度=",radian) #输出弧度
```

尽管上述程序非常简单，但是反映了程序的基本运算模式：数据输入、数据处理、数据输出，如图 1.1.1 所示。不管是小程序还是大程序，一般都遵循这种朴素的**运算模式**，称为 IPO 模式。

#### (1) 数据输入

数据输入是一个程序的开始。一个程序的数据输入途径有：控制台输入、随机产生、图形用户界面输入、文件输入等。

#### (2) 数据处理

数据处理是一个程序最重要的部分，处理的方法被称为算

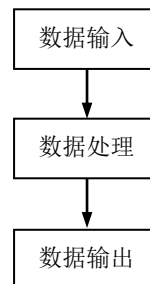


图 1.1.1 程序的基本运算模式

法。

### (3) 数据输出

数据输出是程序不可或缺的部分，没有输出的程序是没有意义的。数据输出有控制台输出，图形用户界面输出、文件输出等。

## 2. 程序设计语言

自然语言是人与人之间进行交流的语言，如汉语、英语、日语等。迄今为止，计算机技术以及人工智能的发展还没有达到人与计算机之间完全使用自然语言进行交流的水平。

程序设计语言是人与计算机进行交流的语言。利用程序设计语言编写计算机程序实现人与计算机的交流。若没有程序设计语言，则计算机几乎没有任何价值。

### (1) 程序设计语言的分类

发布的程序设计语言有上千种，最常用的不过十多种。按照程序设计语言发展的过程，程序设计语言大概分为三类：机器语言、汇编语言和高级语言。

#### ① 机器语言

机器语言是由 0 和 1 二进制代码按一定规则组成的、能被机器直接理解和执行的指令集合。如某型机器上由机器语编写的计算  $15+10$  的程序：10110000 00001111 00101100 00001010。

机器语言编写的程序像“天书”，程序设计成了一件复杂而只能由专家完成的工作，现在已经没有人用机器语言直接编程了。

#### ② 汇编语言

为了克服上述机器语言缺点，人们用英文助记符来表示机器指令。例如，计算  $15+10$  的汇编语言程序如下：

MOV	A, 15	: 把 15 放入累加器 A 中
ADD	A, 10	: 10 与累加器 A 中相加

用英文助记符表示的指令称为汇编指令，汇编指令的集合称为汇编语言，用汇编语言编写的程序称为汇编语言程序。汇编语言程序不能直接在计算机上运行，需要将其翻译成为机器语言程序才能执行。如图 1.1.2 所示。

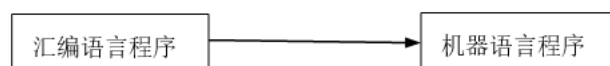


图1.1.2 汇编语言程序翻译成机器语言程序

汇编语言在一定程度上克服了机器语言难读、难改的缺点，但是还没有从根本上解决问题，这推动了高级语言的出现。

#### ③ 高级语言

为了从根本上改变上述缺陷，使计算机语言更接近于自然语言并力求使语言脱离具体机器，达到程序可移植的目的，在 20 世纪 50 年代出现了高级语言。高级语言是一种接近自然语言和数学公式的程序设计语言。高级语言之所以“高级”，就是因为它使程序员可以不用与

计算机的硬件打交道，可以不必了解机器的指令系统，这样，程序员就可以集中精力来解决问题本身而不必受机器制约，极大地提高了编程的效率。图 1.1.3 表示了上述三类语言在机器和人的自然语言之间的紧密关系程度，通常机器语言和汇编语言称为低级语言。从计算机技术和程序设计语言发展的角度来看，程序设计语言的目标是让计算机直接理解人的自然语言，不需要计算机语言，但这个过程是漫长的。

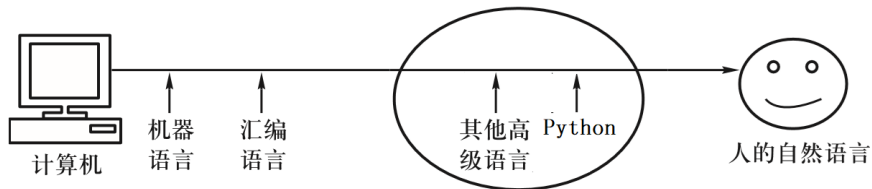


图 1.1.3 程序设计语言在机器和人的自然语言之间的紧密关系程度

(2) 解释方式和编译方式

高级语言翻译程序是将高级语言编写的源程序翻译成目标程序的工具。翻译程序有两种工作方式：解释方式和编译方式，相应的翻译工具也分别称为解释程序和编译程序。

① 解释方式

解释方式是对源程序进行逐句解释执行，即翻译一句立即执行一句，如同“同声翻译”。解释方式不生成目标程序，其工作过程如图 1.1.4 所示。采用解释方式的有 Python、Visual Basic 等。

② 编译方式

编译方式是将整个程序翻译成目标程序，这种方式如同“笔译”。目标程序还不能直接执行，因为程序中还会调用一些其他语言编写的程序或库函数，所有这些程序通过连接程序将目标程序和有关的程序库组合成一个完整的可执行程序。整个过程如图 1.1.5 所示。编译方式执行速度快，但每次修改源程序，都必须重新编译。大多数高级语言都是采用编译方式，如 C/C++、Java 等。

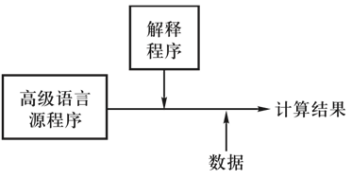


图 1.1.4 解释方式的工作过程

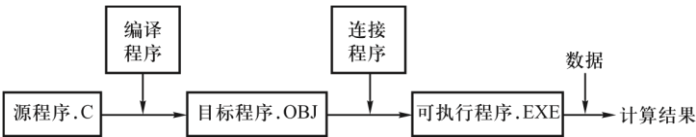


图 1.1.5 C 语言编译方式的工作过程

(3) 面向过程语言与面向对象语言

从描述客观系统来看，程序设计语言可以分为面向过程语言和面向对象语言。

① 面向过程程序设计语言

以“数据结构+算法”程序设计范式构成的程序设计语言，称为面向过程语言。典型的有 BASIC、C 语言等。

### ③ 面向对象程序设计语言

面向对象程序设计语言是一类以对象作为基本程序结构单位的程序设计语言，设计的程序是以对象为核心，在程序运行时对象是基本成分。

面向对象程序设计语言提供了对象、类、方法、实例、变量、消息、派生类和继承等成分。

面向对象程序设计语言在目前属于主流，典型的有 Python、C++、Visual Basic、Java 等。

## 1.2 为什么要学习程序设计

程序设计又称编程，是计算机为解决某个问题而使用某种程序设计语言编写程序代码的过程。近年来，随着人工智能、大数据等新一代信息技术融合到各个领域，以 Python 为代表的高级程序设计语言越来越火热，越来越多的学校开设了相应的课程，甚至编程也将纳入中小学课程的设置。那么为什么从前相对比较小众的编程得到了如此强烈的追捧呢？作为大学生又为什么要学习编程呢？

### 1. 培养逻辑思维能力

程序设计与逻辑思维息息相关。我国著名数理逻辑学家莫绍揆曾经指出：“事实上，它们（程序设计）或者就是数理逻辑，或者是用计算机语言书写的数理逻辑，或者是数理逻辑在计算机中的应用。”荷兰著名计算机科学家 E.W.Dijkstra 曾感慨地说：“搞了这么多年软件，错误不知犯了多少，现在觉悟了。我想，假如我早年在数理逻辑上好好下点功夫的话，我就不会犯这么多的错误。”在使用程序设计语言进行程序设计时要把各种情况、各种可能性丝毫不漏地考虑到，防止在计算过程中出现错误，即使出现错误了也能很快地发现并及时改正，计算机才能依此运行，如稍有错误，轻则得到毫无意义的错误结果，重则产生严重的后果。由此可见，必须有足够的逻辑训练，熟悉推理过程，才能从事程序设计。

学习程序设计能反过来培养逻辑思维。程序设计的工作就是让计算机按照人的意愿去做事。在完成一件复杂的事情时，需要不断的分解、组装，这个过程中，只有自己的逻辑思维清楚了，计算机才能正确的实现你的命令。在计算机的世界中，一切都遵循着非 0 即 1 的基本原则，它教会我们当一个错误出现时，不要试图不加修正，重新执行一次程序，我们就会得到正确的结果，让我们不再心存侥幸，直面问题。

### 2. 提高计算思维能力

学习程序设计可以提高计算思维能力。美国计算机科学家周以真认为，“计算思维是运

用计算机科学的基础概念进行问题求解、系统设计、以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。计算思维的本质是抽象和自动化。”计算思维的根本目的是解决问题，而解决问题的核心是算法设计，这也是程序设计的主要任务。尽管计算思维不仅仅是程序设计，但是计算思维最终需要程序设计去实现。显然，学习程序设计时中，可以潜移默化提高计算思维能力。

### 3. 提高综合素质

近年来，STEAM 开始颠覆传统的教育。STEAM 由科学（Science）、技术（Technology）、工程（Engineering）、艺术（Arts）、数学（Mathematical）五个部分组成。STEAM 的主要理念就是通过多学科融合，培养人全方位的能力，充分挖掘人自己未知的潜力。通过学习程序设计就可以培养上面所有的能力。程序设计首先是一门工程学，把创意想法变成真实的作品需要学习科学技术，按照工程学的方法论规划实施，同样的，优秀的软件也离不开优雅的艺术设计，最后，如果我们想进一步探索计算机的奥秘，当我们深入其基本原理时，我们又必须提升自己的数学能力。

### 4. 提高专业能力

学习程序设计还可以提高专业能力。一方面，各专业与新一代信息技术深度融合，在知识结构上不断完善和拓展，特别是随着大数据技术逐渐开始落地应用，对于数据分析能力的要求也会越来越高，而掌握一定的程序设计技术对于提高数据分析能力是具有较大帮助的；另一方面，人工智能技术走进日常办公领域产生了智慧办公，智慧办公不仅能够降低工作难度，同时也能够提升办公效率，要想充分利用智慧办公系统，就需要具备一定的程序设计能力。学习程序设计不仅能提高自己的专业能力，并且也可以能更快的适应未来的生活和工作环境。

## 1.3 Python 语言简介

### 1. Python 的发展

Python 是由荷兰 Guido van Rossum 设计的解释性的、高级的、通用程序设计语言。

1989 年，Guido 为了开发一个新的脚本解释程序，在 ABC 语言基础上，吸收 Modula-3 优秀思想，结合了 Unix shell 和 C 的习惯而设计了 Python，1991 年发布第一版本。之所以取名 Python，是因为他所挚爱的电视喜剧《Monty Python's Flying Circus》。

2000 年 10 月，Python 2.0 发布。它修复了许多缺陷和错误，开始了广泛应用。

2008 年 12 月，Python 3.0 发布。它打破向后兼容性进行了革命性的修改，让 Python 向



着最好的程序设计语言迈出的坚实一步。

目前，Python 的两个系列版本 2.X 和 3.X 均在使用，但是 3.X 版本是主流。Python 2.X 的最后一个版本是 2.7，2020 年将被终止支持。

## 2. Python 的特点

Python 有 4 个显著的优点：

### （1）简单易学

Python 程序不需要类型说明，简化了 python 的语法，摒弃了 C 语言非常复杂的指针，特别容易学习，特别适合非计算机类专业的人士，是一个面向应用的设计语言。

### （2）简洁优美

Python 代码简洁、优美。Python 采用缩进来标识代码块，通过减少无用的大括号，去除语句末尾的分号等视觉杂乱，使得代码的可读性显著提高，也提高了开发效率。

### （3）自由 / 开源软件

Python 是自由 / 开源软件。也就是说，可以自由地发布它的拷贝、阅读它的源代码、对它做改动、把它的一部分用于新的自由软件中。

### （4）丰富强大的库

Python 不仅标准库庞大，而且还拥有大量第三方库的支持。例如经典的科学计算库：NumPy、SciPy 和 Matplotlib 以及著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK。

与 C/C++ 相比，Python 不可忽视的缺点运行速度慢。

## 3. 应用领域广泛

### （1）常规软件开发

Python 支持面向过程编程和面向对象编程，能够承担任何种类软件的开发工作，因此常规的软件开发、脚本编写、网络编程等可以使用 Python。

### （2）科学计算

随着 NumPy, SciPy, Matplotlib, Enthoughtlibrarys 等众多程序库的开发，Python 越来越适合于做科学计算、绘制高质量的 2D 和 3D 图像。和科学计算领域最流行的商业软件 Matlab 相比，Python 是一门通用的程序设计语言，比 Matlab 所采用的脚本语言的应用范围更广泛，有更多的程序库的支持。

### （3）自动化运维

这几乎是 Python 应用的自留地，作为运维工程师首选的编程语言，Python 在自动化运维方面已经深入人心，比如 Saltstack 和 Ansible 都是大名鼎鼎的自动化平台。

### （4）云计算

开源云计算解决方案 OpenStack 就是基于 Python 开发的，搞云计算的同学都懂的。

### （5）WEB 开发

基于 Python 的 Web 开发框架不要太多，比如耳熟能详的 Django，还有 Tornado，Flask。其中的 Python+Django 架构，应用范围非常广，开发速度非常快，学习门槛也很低，能够帮助你快速的搭建起可用的 WEB 服务。

#### （6）网络爬虫

网络爬虫是大数据行业获取数据的核心工具。没有网络爬虫自动地、不分昼夜地、高智能地在互联网上爬取免费的数据，许多大数据公司将无法生存。能够编写网络爬虫的编程语言有不少，但 Python 绝对是其中的主流之一，其 Scrapy 爬虫框架应用非常广泛。

#### （7）数据分析

在大量数据的基础上，结合科学计算、机器学习等技术，对数据进行清洗、去重、规格化和针对性的分析是大数据行业的基石。Python 是数据分析的主流语言之一。

#### （8）人工智能

Python 在人工智能大范畴领域内的机器学习、神经网络、深度学习等方面都是主流的编程语言，得到广泛的支持和应用。

## 1.4 简单的 Python 程序

下面通过一个简单的实例，说明 Python 程序的基本组成。

**【例 1.1】**角度转换弧度程序。输入角度，求弧度。

程序：

```
import math                #导入 math 模块
angle=float(input("请输入角度: "))  #输入角度，并且转换为 float 类型
radian=math.pi/180*angle    #引用 math 库中的 pi，计算弧度
print("弧度=",radian)       #输出弧度
```

这是一个简单的 Python 源程序，运行结果如图 1.1.6 所示。在图中，数据 45 是用户从键盘输入的，而结果 0.7853981633974483 是程序输出。

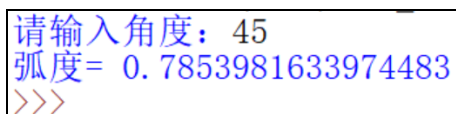


图 1.1.6 程序运行结果

说明：

① 一行中以“#”开始的内容称为注释。它的作用是对程序进行说明，提高程序的可读性。

② 语句 `import math` 用于导入 `math` 模块，因为下面的代码需要使用 `math` 模块中定义的符号常量 `pi`，代表 3.141592653589793。

③ `input` 函数用于读入角度，读到的数据是字符串类型，使用 `float` 函数转换成 `float` 类型。

④ 语句 `radian=math.pi/180*angle` 的作用是使用 `math` 中的符号常量 `pi` 根据公式计算弧

度，并且赋值给变量 `radian`。

⑤ 语句 `print("弧度=",radian)` 输出计算结果，输出格式为：

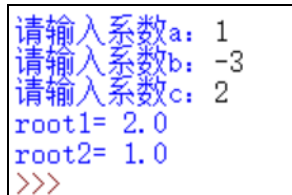
弧度=XXX.XXXXXXX

**【例 1.2】** 求一元二次方程  $ax^2+bx+c=0$  的根。

程序：

```
import math                #导入 math 模块
a=float(input("请输入系数 a: "))    #输入系数 a，转换成 float 类型，并且赋值给变量 a
b=float(input("请输入系数 b: "))    #输入系数 b，转换成 float 类型，并且赋值给变量 b
c=float(input("请输入系数 c: "))    #输入系数 c，转换成 float 类型，并且赋值给变量 c
delta=b*b-4*a*c              #计算 b2-4ac，并且赋值给变量 delta
if delta>=0:                  #判断 delta 是否 ≥ 0，若是则执行下面缩进的语句
    root1=(-b+math.sqrt(delta))/(2*a)    #计算第 1 个根，并且赋值给变量 root1
    root2=(-b-math.sqrt(delta))/(2*a)    #计算第 2 个根，并且赋值给变量 root2
    print("root1=",root1,"\nroot2=",root2) #输出两个根
else:
    print("方程没有实根。")            #若 delta ≥ 0 不成立，若是输出“方程没有实根。”
```

程序运行结果如图 1.1.7 所示。其中 1、-3 和 3 是用户输入的数据。



```
请输入系数a: 1
请输入系数b: -3
请输入系数c: 2
root1= 2.0
root2= 1.0
>>>
```

图 1.1.7 程序运行结果

说明：

① 若 `delta>=0` 成立，则执行下面缩进的四条语句。语句增加缩进表示语句块的开始，减少缩进表示语句块的结束，相同的缩进表示语句处于同一个级别。若后面的三条语句不缩进表示与 `if` 语句同级别，语法上会出错。

② 若 `delta>=0` 不成立，则执行 `else` 下面缩进的 `print("方程没有实根。")` 语句。

③ `sqrt` 函数是 `math` 模块中的求平方根函数。

从上面两个实例，可以看到 `Python` 程序的组成及书写规则为：

(1) `Python` 程序由一个或多个语句组成，从上到下按次序执行。

(2) 注释是从“`#`”开始到本行结束的内容。除了增加程序的可读性之外，它没有其它任何作用。若是多行注释，则应用三个单引号“`'''`”括起来。例如，

```
'''
这是一个验证程序，用于验证用多行注释命令
文件名为 Test.py
'''

print("Hello,World!")
```

(3) 一般情况下，一行只写一个语句。若一行太长需要换行，则应在换行处加上“`\`”表示换行。一行写多个语句时，语句之间“`;`”分隔。

(4) 缩进是 `Python` 简洁的特征之一。虽然可以缩进，但也应按编辑器的指示规范缩进，而不能任意缩进。

(5) Python 中大小写字母是有区别的，这一点初学者要特别注意。

## 1.5 Python 开发环境

在 Windows 下，Python 的常用开发环境（Integrated Development Environment，简称 IDE）有五个：

- ① IDLE Python 内置的一个简洁的集成开发环境，使用第三方库时需要安装；
- ② Anaconda 严格来说，Anaconda 是一个开源项目，其中的 Spder 才是集成开发环境。由于使用 Anaconda 主要是使用 Spder，所以常常将 Spder 等同于 Anaconda。Anaconda 下已经集成了大量常用的第三方库，一般情况不需要安装新的第三方库。
- ③ Jupyter Notebook 基于网页的用于交互计算的应用程序，可以随 Anaconda 一起安装。可以网页的形式打开、编写和运行代码，并且运行结果也会直接在代码块下显示。
- ④ PyCharm 由 JetBrains 开发的 Python IDE，支持 Django 和 Google App Engine。
- ⑤ VSCode 是 Visual Studio Code 的简称，微软开发的现代 Web 和云应用的跨平台源代码编辑器，在安装相应的扩展包后可具备强大的编程开发能力。

本书推荐使用 IDLE 和 Anaconda。

### 1.5.1 IDLE 简介

IDLE 是一个 Python 自带的集成开发环境。安装 Python 时，IDLE 会自动安装。

#### 1. IDLE 下载

直接在官网（<https://www.python.org/>）选择合适的版本下载安装包，下载完成后根据提示直接安装。图 1.1.8 是在 Python 官网主页中鼠标指向“Downloads | Windows”后看到的界面，在其中可以下载最新版本的 Python for Windows。

下载安装包时需要考虑三个因素：一、操作系统是 Windows、Mac OS X 还是其他操作系统；二、下载 32 位还是 64 位的 Python。若操作系统是 32 位的，则 Python 只能下载 32 位的，否则下载 32 位和 64 位的都可以；三、Python 版本号，并不是最新版本是最好的，某些情况下旧的版本可能更适合自己的操作系统。

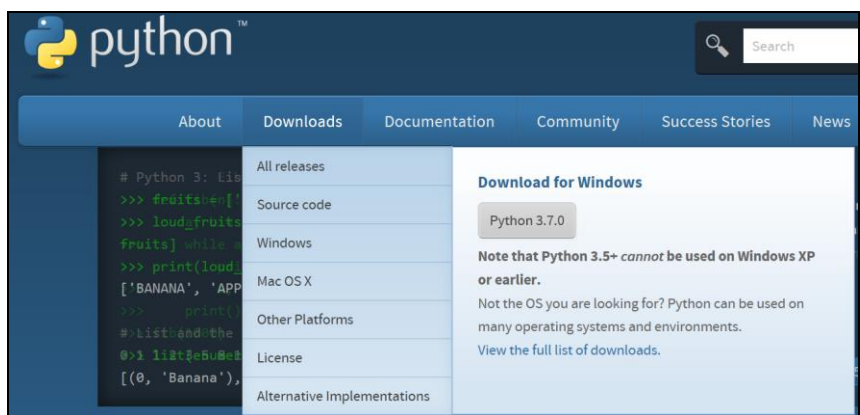


图 1.1.8 Python 官网 “Downloads” 主页

安装时应选定 “Add Python 3.7 to PATH” 复选框，如图 1.1.9 所示，将 Python 的安装目录路径添加到环境变量 PATH 中，方便以后安装第三方库。安装后在 Windows “开始” 菜单中出现了 Python 程序组，如图 1.1.10 所示。选择程序组中 “IDLE” 命令就进入了开发环境。

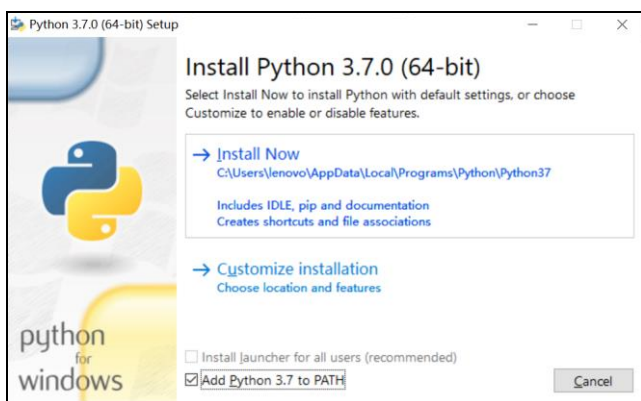


图 1.1.9 Python 安装界面



图 1.1.10 “开始”菜单中的 Python 程序组

## 2. 运行方式

进入 IDLE 开发环境后就可以编辑和运行 Python 程序。有两种运行方式：Shell 交互方式和文件直接执行方式。

### ① Shell 交互方式

Shell 实质是一种交互式的命令解释器。它的执行过程为先输入一条命令，解释器解释并执行这条命令，执行完毕就显示相应的结果。例如，图 1.1.11 是输入 “print(“同学，你好!”)” 命令后，Python 解释器执行并显示结果：同学，你好!。

需要注意的是 “>>>” 是命令提示符。

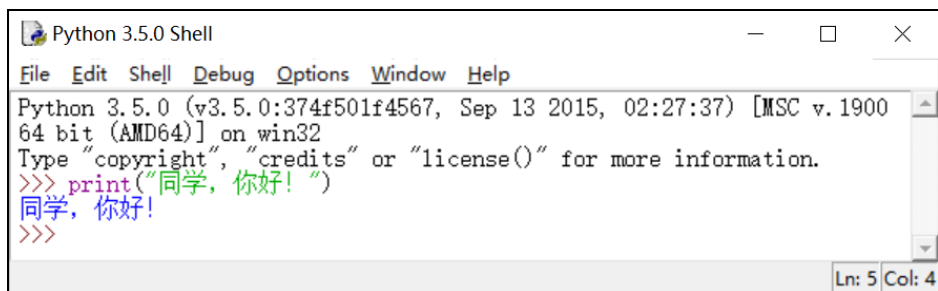


图 1.1.11 Shell 交互方式

## ② 文件执行方式

这种方式是创建一个以.py 为扩展名的 Python 程序文件，Python 解释器执行后在 Shell 窗口中显示结果。

在 Shell 中选择“File | New File”命令，出现如图 1.1.12 所示的文件编辑窗口，在其中输入程序代码，然后执行“Run | Run Module”命令，则程序执行结果如图 1.1.13 所示。

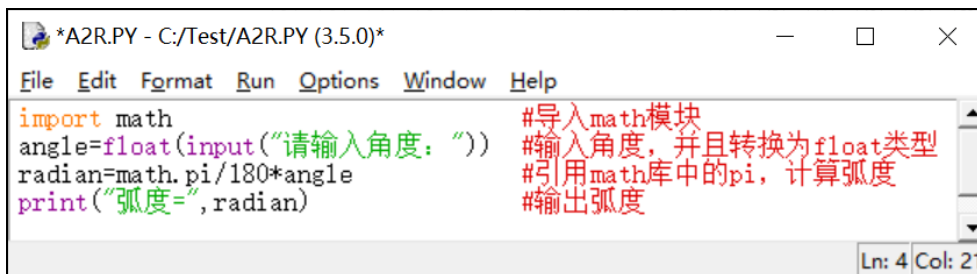


图 1.1.12 文件编辑窗口

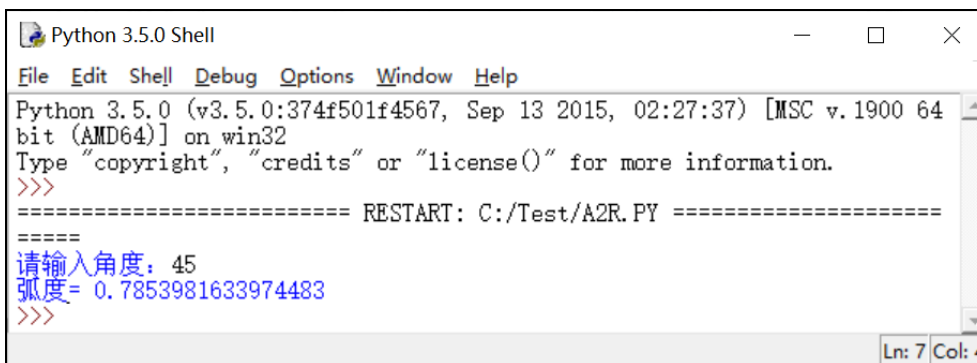


图 1.1.13 Shell 窗口中显示程序文件执行结果

在程序执行开始前，会提示保存程序文件。数据输入也是在 Shell 窗口中进行的。

## 1.5.2 Anaconda 简介

Anaconda 集成了大量常用的第三方库，安装后一般不再需要特别去安装，避免了安装第三方包的麻烦，深受用户欢迎。

### 1. Anaconda 的安装

直接在官网 (<https://www.anaconda.com/>) 选择合适的版本下载 Anaconda 安装包，下载完成后根据提示直接安装，安装后在 Windows 菜单中出现如图 1.1.14 所示的程序组，其中 Spyder 就是 Anaconda 的集成开发环境。



图 1.1.14 “开始”菜单中的 Anaconda 程序组

## 2. 执行方式

选择“开始”菜单中的“Anaconda | Spyder”命令后出现如图所示 1.1.15 所示的界面，在其中就可以编辑、执行程序。

Anaconda 程序的执行方式也分为两种：一、Shell 执行方式。窗口左下角的 Console 窗口类似 IDLE 的 Shell，在其中可以一条一条输入、运行命令。命令提示符不是“>>>”，而是 In[n]，其中“n”是一个数字，代表命令的序号；二是文件执行方式。首先在左边的编辑窗口中输入程序，然后执行“Run | Run”命令（或者直接选择工具栏上的“Run file”按钮，图标为“▶”），就可以执行程序了。

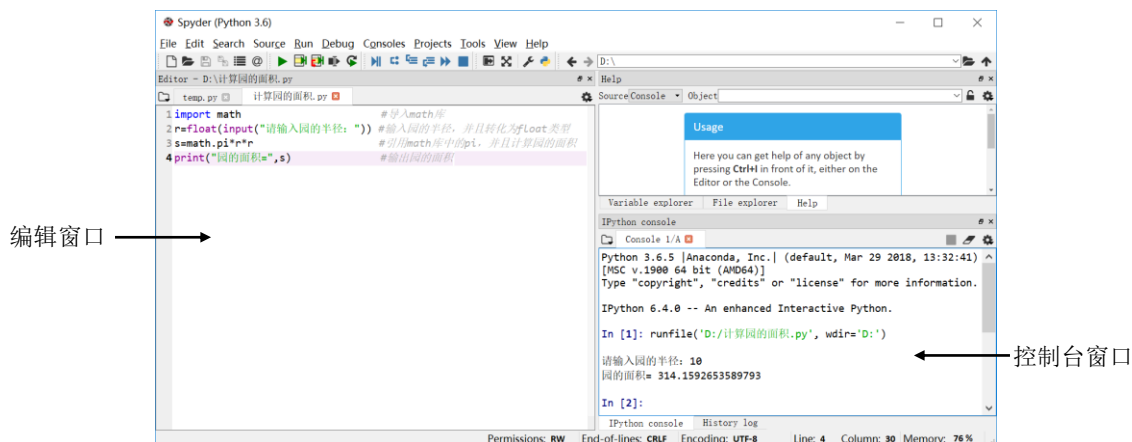


图 1.1.15 Spyder 界面

## 1.5.3 第三方库

Python 除了拥有随 Python 解释器一起安装的标准库之外，还拥有丰富的第三方库，这是 Python 功能强大而深受欢迎的原因之一。

### 1. 第三方库的安装

Python 的第三方库在使用之前必须安装。一般来说，有两种安装方法：直接安装和下

载 whl 文件后安装。

(1) 使用 pip 命令直接安装

使用 pip 命令直接安装，语法格式如下：

```
pip install jieba
```

#假定需要安装的第三方库为 jieba

判断是否已经安装的方法是执行 import 命令尝试导入，若没有显示错误信息则表示已经安装成功。例如，下面的信息表示 jieba 没有安装成功。

```
>>> import jieba
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import jieba
ModuleNotFoundError: No module named 'jieba'
```

1) IDLE 中安装第三方库

由于 IDLE 仅仅安装了标准库，若要使用第三方库则一定要安装。假如 Python 安装时选定了“Add Python 3.7 to PATH”复选框已经将安装目录路径添加到了环境变量 PATH 中，则安装步骤为：

- ① 进入命令提示符界面；
- ② 执行安装命令，如图 1.1.16 所示。

```
C:\Users\lenovo>pip install jieba
```

#假定需要安装的第三方库为 jieba



图 1.1.16 安装第三方库（选定“Add Python 3.7 to PATH”情况下）

假如 Python 安装时没有选定“Add Python 3.7 to PATH”复选框，则安装有点麻烦。具体过程是：

① 设法确定 Python 的安装目录。假定 Python 的安装目录是 C:\Python37，pip.exe 在 Python37 文件夹下 Scripts 子目录中。

- ② 进入命令提示符界面；
- ③ 使用命令进入 C:\Python37\Scripts 目录：

```
C:\Users\lenovo>CD C:\Python37\Scripts
```

- ④ 执行命令，如图 1.1.17 所示。

```
C:\Python35\Scripts>pip install jieba
```

#假定需要安装的第三方库为 jieba



图 1.1.17 安装第三方库（没有选定“Add Python 3.7 to PATH”情况下）



需要注意的是：有可能因为 pip 版本太低而安装失败，此时应先升级，升级的具体命令：

```
python -m pip install --upgrade pip
```

pip 升级命令在安装失败时会提示，只要仔细观察即可。

## 2) Anaconda 中安装第三方库

由于 Anaconda 已经集成了常用的第三方库，一般不需要安装。若要安装特殊的库，则执行“开始”菜单中的“Anaconda | Anaconda Prompt”命令后，然后运行 pip 命令，如图 1.1.18 所示。

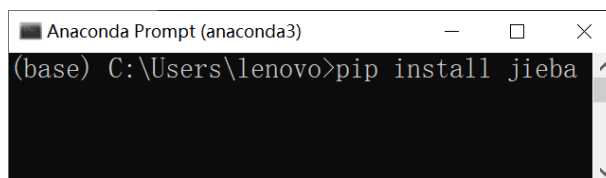


图 1.1.18 Anaconda 中安装第三方库

### (2) 安装 whl 文件

许多 Python 第三方库是以 whl 文件的形式提供的，此时就需要下载 whl 文件后再安装，具体步骤如下：

#### ① 下载 whl 文件。

whl 文件主要有两个网站：

◆ pypi 网站：<https://pypi.python.org/pypi/>

◆ 国内 whl 集合网：<https://www.lfd.uci.edu/~gohlke/pythonlibs/>

② 若是 Anaconda 则，执行“Anaconda Command Prompt”；若是 IDLE，则进入命令提示符界面（路径添加到环境变量 Path 中）。

#### ③ 执行 pip 安装 whl 文件，使用形式为

```
pip install whl 文件名
```

例如，若有 MyLib.whl 文件放在 D:\Test 盘跟目录下，则命令为：

```
pip install D:\Test\MyLib.whl
```

## 2. 常用的第三方库

对 Python 初学者来说，常用的第三方库如表 1.1.1 所示。

表 1.1.1 常用的第三方库

第三方库	说 明
Jieba	中文分词库
wordcloud	词云图生成器
PIL	图像处理库
Matplotlib	数据可视化，可绘制各种图形
Numpy	数值计算库
Pandas	数据分析包
Scikit-learn	机器学习库

TensorFlow	深度学习库
win32com	有关 Windows 系统操作、Office（Word、Excel、Access 等）文件读写等的综合应用库

### 3. 查看已安装的第三方库

有时需要查看 Python 环境中安装了什么库，常用方法是在命令提示符界面中执行命令：

```
C:\Users\lenovo>pip list
```

结果如图 1.1.19 所示。

```

C:\>pip list
Package            Version
-----
cyclor             0.10.0
jieba              0.39
kiwisolver         1.0.1
matplotlib         2.2.2
numpy              1.14.2
pip                19.2.3
pygame            1.9.3
pyparsing          2.2.0
python-dateutil    2.7.3
pytz               2018.4
pywin32            224
setuptools         18.2
six                1.11.0
xlrd               1.1.0
xlutils            2.0.0
xlwt               1.3.0
C:\>

```

图 1.1.19 查看已安装的第三方库

## 1.6 程序调试与获取帮助

### 1.6.1 程序调试

在程序设计过程中，调试是一个不可缺少的环节。俗话说，“三分编程七分调试”，说明调试的工作要比编程大得多。

程序的错误可以分为两种：一、语法错误，如变量名大小写错误，解释器可以发现，用户可以根据提示信息修改；二、逻辑错误，如将 `c=a+b` 写成 `c=a-b`，解释器发现不了，用户往往需要通过设置断点，跟踪程序的运行过程，才能发现错误。

#### 1. 语法错误

程序若有语法错误，系统则会在输出窗口中显示错误信息。错误信息的形式为：

```
File "文件名", line 行号, in <module>
```

错误代码  
错误类型: 错误内容

例如,

```
File "C:/Test/A2R.PY", line 1, in <module>
import Math                      #导入 math 模块
ModuleNotFoundError: No module named 'Math'
```

表示在“C:/Test/A2R.PY”文件的第 1 行处有一个 `ModuleNotFoundError` 错误, 没有 `Math` 模块。因为 Python 中大小写是敏感的, 即大小写字母是不同的, 因此 `math` 不能写成 `Math`。

常见错误有:

① 变量没有定义

```
x=2019                # 此处 x 是小写
print(X)              # 此处 X 是大写, 与 x 是不同的, 导致出现变量没有定义错误
```

错误信息为: `NameError: name 'X' is not defined`

② 没有指定引用库

```
import math
angle=float(input("请输入角度: "))
radian= pi/180*angle
print("弧度=",radian)
```

错误信息为: `NameError: name 'pi' is not defined`, 实质是没有指定引用库。

③ 没有圆括号

```
print "弧度=",radian    #输出弧度
```

④ 没有导入模块错误

```
x=sqrt(5)              # 使用 sqrt 之前没有导入 math 模块
print(x)
```

⑤ 数据类型错误

```
x= input("请 x 的值: ")
y=x*x                  # 通过 input 输入的数据是一个字符串, 应转换成数值类型
print(y)
```

此错误信息为: `TypeError: can't multiply sequence by non-int of type 'str'`

修改语法错误时要注意:

① 有时候系统并不能很准确地定位错误所在的位置, 这是因为代码中的某个错误, 在解释器看来实际是另外一种错误。一般在定位到的位置如果没有发现错误的话, 这个错误应该就在附近。

② 一个错误可能会产生若干条错误信息, 但是第一条信息最能反映错误的位置和类型, 所以务必先根据第一条错误信息进行修改, 修改后立即进行执行是否还有错误信息。即使有多个错误, 也要一个个地修改, 即修改一处执行一次。

## 2. 逻辑错误

一般来说, 逻辑错误很难发现, 常用的有效方法三种: 添加 `print` 语句、设置断点、单步运行。下面 Anaconda 的调试方法。

### (1) 添加 `print` 语句

在怀疑可能是错误的前后，添加 `print` 语句输出变量或表达式的值，根据输出情况发现错误。

## (2) 设置断点

设置断点并让程序运行到该断点，然后在“Variable explorer”窗口观察各变量的值，从中发现错误。

### ① 设置断点方法

在需要设置断点的行首双击设置断点，或者首先将光标移动到需要设置断点的行上，然后选择“Debug | Set/Clear breakpoint”命令。断点设置后，行首有一个实心圆点。

### ② 运行到断点

选择“Debug | Debug”命令或者选择  按钮，进入调试状态，标志是 Console 窗口出现“ipdb>”提示符。

在“ipdb>”下可以使用下列命令：

- `c` 命令：程序执行到下一个断点
- `q` 命令：退出调试
- `!(python 语句)`：在 `pdb` 提示符下执行 `python` 语句，可以用来查看变量值或者给变量临时指定值

## (3) 单步执行

程序进入调试状态后，可以单步执行语句。单步运行命令如图 1.1.20 所示。

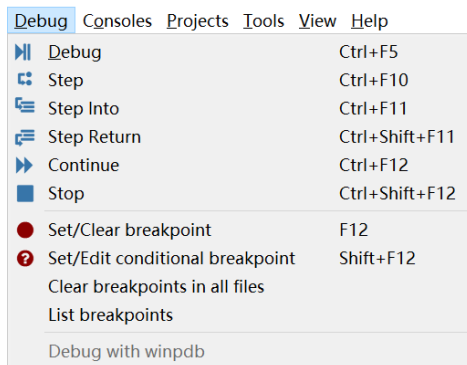


图 1.1.20 Debug 菜单

Debug：开始调试

Step：开始单步执行

Step Info：单步运行进入函数体内

Step Return：单步运行跳出函数体

Continue：继续

Stop：停止调试

一般来说，逻辑错误解释器很难发现的，但对程序来说很可能是致命的。为了避免这样的错误，这就要求编写代码的时候要遵循一定的规范，并且要对程序做足够的测试工作。

## 1.6.2 获取帮助

程序设计时，即使是高级程序员也会遇到问题，并且需要得到帮助。Python 中获取帮助可以有两个途径：一、访问 <https://docs.python.org>，它提供了各个版本、各种语言的帮助文档；二、Python 自带了一个内置的帮助系统，通过 `help`、`dir`、`__doc__` 获得帮助。

### 1. help 函数

通过 `help` 函数可以获对象的信息，其使用形式为

`help(对象)`

例如，如果要了解 `sqrt` 的功能和用法，则执行下面的语句：

```
>>> import math                      # 导入 math 模块后才能查看 math 对象的帮助
>>> help(math.sqrt)

Help on built-in function sqrt in module math:

sqrt(x, /)                            # “/” 表示函数参数 x 不可以是负的

    Return the square root of x.
```

### 2. dir 函数

`dir` 函数可以用来查看对象的所有属性及方法，其使用形式为：

`dir(对象)`

说明：在 python 中任何东西都是对象，一种数据类型，一个模块等，都有自己的属性和方法，都可以通过 `dir` 函数查。例如，如果要了解 `math` 中所有的函数和常数，则执行下面的语句。

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan',
'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow',
'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

### 3. \_\_doc\_\_

Python 中有一个特色——文档字符串（DocStrings），是指模块、类、函数中等添加的说明性文字，可以通过 Python 自带的标准方法——`__doc__` 将这些信息输出。其使用方法为

`对象.__doc__`

例如，如果得到关于 `math` 的描述性文字信息，则执行下面的语句：

```
>>> import math
>>> math.__doc__
'This module provides access to the mathematical functions\ndefined by the C standard.'
```

注意：当不是函数、方法、模块等调用 `doc` 时，而是具体对象调用时，会显示此对象从属的类型的构造函数的文档字符串。

## 习题

### 一、选择题

1. 在下列程序设计语言中，不属于面向对象程序设计语言的是\_\_\_\_\_。  
A. Python  
B. VB.NET  
C. C  
D. Java
2. 在下列程序设计语言中，属于解释执行的\_\_\_\_\_。  
A. Python  
B. C++  
C. C  
D. Java
3. Python 自带的开发环境是\_\_\_\_\_。  
A. PyCharm  
B. TensorFlow  
C. Anaconda  
D. IDLE
4. 在下列程序设计语言中，属于自由/开源软件（Free/Libre and Open Source Software，简称 FLOSS）的是\_\_\_\_\_。  
A. C/C++  
B. Python  
C. VB.NET  
D. Java
5. 执行下列程序时输入 1 和 2，则输出是\_\_\_\_\_。  

```
a=input()
b=input()
c=a+b
print(c)
```

  
A. 3  
B. 12  
C. 4950  
D. 99
6. 下列不属于调试手段的\_\_\_\_\_。  
A. 谨慎地设置输出语句  
B. 设置断点  
C. 单步执行  
D. 查阅文献

## 二、填空题

1. 能被机器直接理解和执行的语言是\_\_\_\_\_。
2. 程序的基本运算模式：数据输入、数据处理、数据输出，这种朴素的运算模式被称为\_\_\_\_\_。
3. \_\_\_\_\_是 Python 内置的一个简洁的集成开发环境。
4. 选定“Add Python 3.7 to PATH”复选框，可以将 Python 的安装目录路径添加到环境变量中。

5. 第三方库最常见的安装命令是\_\_\_\_\_。
6. Python 功能强大而深受欢迎的原因之一是拥有\_\_\_\_\_、第三方库和自定义模块。
7. 查看已安装的第三方库的完整命令是\_\_\_\_\_。
8. 程序的错误可以分为语法错误和\_\_\_\_\_。

### 三、问答题

1. Python 有什么特点？
2. 什么是开源软件？什么是自由软件？
3. Python 程序的书写格式有什么特点？
4. 请简述 IDLE 和 Anaconda 两个编程环境的主要区别。
5. 请查阅有关资料，说明自己所在学科常用的 Python 库有哪些？
6. 程序错误分成哪几类？在 Python 中如何调试程序？

## 第 2 章 语言基础

计算机程序是通过计算来解决实际问题的。程序是用程序设计语言所描述的解决问题的方法和步骤。从组成上讲，程序包括数据和对数据的操作两部分。数据是程序加工处理的对象，操作则反映了对数据的处理方法，体现了程序的功能。计算机能处理的数据是多种多样的，程序设计语言定义了不同的数据类型，规定了各种类型数据的存储和所能进行的处理。

本章介绍 Python 语言提供的数据类型、各类型数据的表示方法和运算规则等语言基础知识。

### 2.1 数据类型

#### 2.1.1 引例—计算地球质量的问题

**【例 2.1】** 利用计算机来解决初等数学问题。编写一个计算地球质量的程序，地球体积的计算公式为： $V=\frac{4}{3}\pi r^3$ ，地球半径  $r$  为 6356.91 千米，平均密度为每立方米 5.52 吨。要求地球半径  $r$  从键盘输入，地球密度为常量。

程序如下：

```
r=input("输入地球半径 r: ")
v=4/3*3.14159*(r*1000)**3          # 体积，单位为立方米
w=v*5.52                            # 求得地球质量，单位为吨
print("地球质量 w=",w)
```

程序运行和输入：

```
Traceback (most recent call last):
  File "D:/py/2-1.py", line 2, in <module>
    v=4/3*3.14159*(r*1000)**3          # 体积，单位为立方米
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>>
```

程序第 2 行有错误，原因是变量  $r$  在第 1 行通过 `input` 函数输入后获得的是字符串类型“6356.91”，字符串类型是不能进行乘方（\*\*）运算。

程序代码若修改为：

```
r=int(input("输入地球半径 r: "))      # int()函数将输入的数字字符串转换成整数类型
```

程序运行输入 6356，能有运行结果。但程序运行若输入 6356.91，则出现如下错误信息：

```
输入地球半径 r: 6356.91
Traceback (most recent call last):
  File "D:/py/2-1.py", line 1, in <module>
    r=int(input("输入地球半径 r: "))
```



```
ValueError: invalid literal for int() with base 10: '6359.91'
```

原因是输入的 6356.91，而 `int()` 函数只能将整数形式的字符串转化为整数。

要解决此问题，则要调用 `float()` 函数，即 `r=float(input("输入地球半径 r: "))`，程序运行结果如下：

```
输入地球半径 r: 6356.91
地球质量 w= 5.939713652967326e+21
```

说明：对于大数，系统自动以指数形式显示，最后的“e+21”表示 10 的 21 次方；若要保留三位小数，则可用如下格式输出：

```
print("地球质量 w=%5.3e"%(w))    # 以指数形式显示，保留 3 位小数
```

显示：

```
地球重量 w=5.940e+21
```

从上面引例中可看出：在程序设计语言中，对要处理的数据规定数据的类型、所能进行的运算、数据的输入和输出的方式和格式控制等，在本章将作逐一介绍。

## 2.1.2 内置数据类型

每一种程序设计语言都会预先设置一些数据类型，称为内置数据类型，在程序中可以直接使用。Python 提供的内置数据类型如图 1.2.1 所示，包括数值型（整型、浮点型和复数类型）、字符串型、布尔型等一般语言都具有的数据类型；还提供了列表、元组、字典、集合等 Python 特有的组合数据类型。数据类型可分为两类：基本数据类型和组合数据类型。

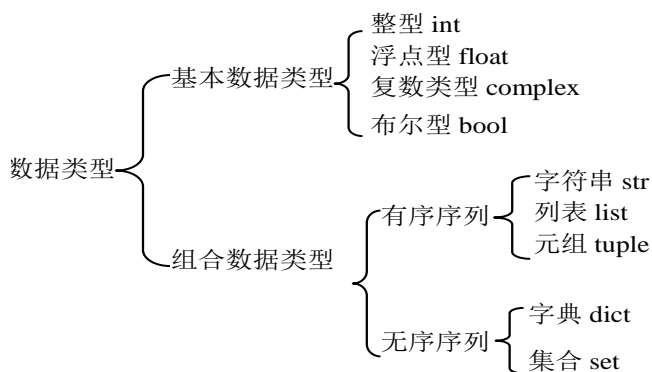


图 1.2.1 Python 的内置数据类型

说明：

（1）有序序列是一维向量，元素之间存在次序关系，可通过索引访问元素。字符串可以方便地表示文本数据，列表和元组可表示数据的序列。

（2）无序序列是指元素之间不存在次序关系，因此不能用索引来访问。集合也可表述数据的组合，但是其中的元素不重复出现；字典使用键-值(key-value)进行存储，查找速度极快，即通过键来获得对应值。

## 2.1.3 常量、变量和标识符

计算机程序是通过计算来解决实际问题的。计算中涉及的数据在程序中以常量或变量的形式出现。变量的值可以变化，而常量则表示固定不变的数据。如在例 2.1 中求圆体积  $v$  的公式为：

$$v = 4/3 * 3.14159 * (r * 1000) ** 3$$

其中  $r$ 、 $v$  是变量，4、3、3.14159 和 1000 是常量。假如当  $r$  的值为 6356.91 时， $v$  获得相应的计算结果  $1.0760350820592982e+21$ 。

### 1. 常量

常量是在程序运行中固定不变的数据，通过常数值直接反映了其数据类型。例如，123 是整数、123.0 是浮点数、'123' 是字符串，[123] 是列表等。

### 2. 变量

变量是在程序运行过程中其值可以变化的量。变量具有变量名、类型和值三要素。变量通过变量名访问，变量名必须满足标识符命名规则。在 Python 中，不需要事先声明变量名及类型，通过赋值运算符可以创建任意类型的变量，其数据类型与所赋的值一致，通过 `type(变量)` 函数获得变量的数据类型；不仅变量的值可以改变，变量的数据类型也可改变。

为什么变量的数据类型可以变化？Python 作为面向对象的编程语言，一切都是对象，包括数据类型、变量、函数、方法、类等。Python 中变量并不直接存储数据而是个引用到实际对象的名称，赋值运算只是改变了变量的引用对象，可以使用函数 `id(变量)` 获取引用对象的存储地址。因此 Python 属于动态类型的程序设计语言。

### 2. 变量

变量是在程序运行过程中其值可以变化的量。变量名必须符合标识符的命名规则。在 Python 中变量不需要事先声明类型就可以使用，而且变量类型是可以变化的。例如：

```
x=10                                # 没有声明直接使用，变量为整型
x="Tongji University"              # 变量从整型改变为字符串类型
```

为什么变量的数据类型可以变化？因为在 Python 中变量与值的关系是引用关系，赋值运算只是改变了变量的引用对象。Python 因为变量类型可以改变所以属于动态类型程序设计语言，如图 1.2.2 所示。

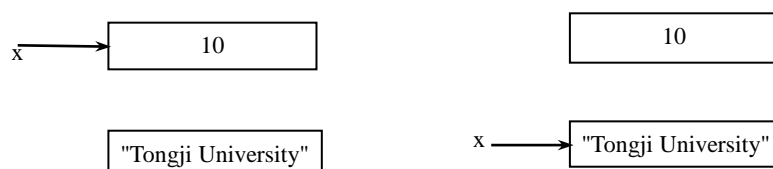


图 1.2.2 Python 动态类型变量示意

### 3. 标识符

标识符是指在程序中为一些要处理的对象起个名字，包括变量名、函数名、类名等。

在 Python 中，标识符的命名规则：以字母或下划线开头，后面可跟字母、数字和下划线的序列，大小写区分。在中文系统中，标识符中任意位置可以出现汉字。

例如：area, score\_list, x123、学号、姓名等都是合法的标识符；而 3x, x-y, x y 等是非法的标识符。

注意：在 Python 中大小写区分的，即 x123、X123 是两个不同的变量名；关键字不能用于命名变量；也要避免将 Python 的内置的数据类型、函数名等用作变量名，以免引起混淆和带来不必要的麻烦。

#### 4.关键字

Python 语言保留某些单词用做特殊用途，这些单词被称为关键字，也叫作保留字；在不同的编程开发环境中系统会以不同的颜色标识关键字来突出语法表示。用户定义的标识符（变量名、函数名等）不能与关键字相同，否则在运行时会出现异常。

**【例 2.2】**显示 Python 中用到的关键字。

要显示 Python 中用到的关键字，必须导入 keyword 库。

```
>>> import keyword          #导入 keyword 库
>>> print(keyword.kwlist)
```

['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

Python 3.X 拥有 33 个关键字，可以通过 help()命令进入帮助系统查看相关关键字说明。

```
>>> help()                  #进入帮助系统
help> if                    #查看 if 关键字说明，help>是帮助系统提示符
help>quit                  #退出帮助系统
```

## 2.2 数值数据的表示和处理

### 2.2.1 数值数据常量

Python 的数值数据包含了整数、浮点数和复数。常量表示形式见表 1.2.1

表 1.2.1 数值类型常量举例

数据类型	表示形式	说明	举例
整型	①十进制：±n	n 为 0~9 数字组成	1234、-101
	②二进制：0Bn 或 0bn	0B 或 0b 二进制标记，n 为 0~1 数字组成	0B1110100
	③八进制：0O n 或 0on	0O 或 0o 八进制标记，n 为 0~7 数字组成	0O144
	④十六进制：0X n 或 0x n	0X 或 0x 十六进制标记，n 为 0~9 数字、A~F 或 a~f 字母组成	0X5B
浮点型	①小数形式 ±m.n	m、n、k 为 0~9 数字组成	123.45
	②指数形式：±m[.n]E±k	E 或 e 表示指数符号	0.12345E+3
复数型	实部+虚部 J	实部（real）和虚部（imag）可以是整数或浮	12.3+45.6J

		点数, J 或 j 是后缀	
--	--	---------------	--

### 1. 整数

整数是不带小数点和指数符号的数。整数有四种表示形式，分别为：十进制、二进制整数、八进制整数和十六进制整数，除十进制整数外都有标记，如表 1.2.1 所示，常用十进制整数。

例如，123、+123、0B1111011、0O173 和 0X7B 都是合法的整数，后三个分别为二、八、和十六进制整数，并与十进制整数 123 等值。

整数特点数值精确表示；在 Python 中，整数类型没有取值范围的限制。

### 2. 浮点数

浮点数又称实数，有两种表示形式，分别为：小数形式和指数形式，如表 1.2.1 所示。

例如，123.0、0.123e3、1.23E2、123e0 等都是合法的浮点数，后三个是指数形式并与 123.0 小数形式等值。一般使用小数形式比较直观，而对于特大或特小的浮点数使用指数形式。

要注意的是浮点数特点数值不能精确表示，这样在计算时可能会出现误差。

#### 【例 2.3】浮点数运算精度问题和解决的方法

```
>>> x=1.2
>>> x-1.0
0.19999999999999996
>>> 1.2-1.0==0.2           # 两浮点数相减判断结果不正确，这是浮点数精度引起的
False
>>> abs(x-1.0-0.2)<1e-5    # 解决的方法是将相等比较改为绝对值之差是否足够小来比较
True
```

### 3. 复数

Python 提供了复数类型，复数由实部和虚部构成，形式：实部+虚部 j 或 J。通过表 1.2.4 中所列出的 complex(x1[,x2])函数可以将 1 个或者 2 个数值转换成复数类型。

#### 【例 2.4】复数表示。

```
>>> c=12.3+45.6j
>>> c           # 显示 (12.3+45.6j)
>>> c.real      # 显示 12.3
>>> c.imag      # 显示 45.6
>>> c1=complex(10)
>>> c1          # 显示 (10+0j)
>>> c2=complex(10.0,20.2)
>>> c2          # 显示 (10+20.2j)
```

## 2.2.2 算术表达式

要进行各种复杂的运算，就需要各种运算的符号，通过运算符和操作数组合成表达式，

实现程序编制中所需的大量操作。

### 1.算术运算符

在 Python 中,为数值类型提供了通常的算术运算和运算规则,运算符及其含义如表 1.2.2 所示(若已知  $a=3$ )。

表 1.2.2 算术运算符

运算符	含义	优先级	实例	结果	说明
-	负号	1	-a	-3	单目运算
**	幂运算	2	27**(1/3)	3.0	乘方
*	乘	3	a*a*a	27	
/	除	3	10/a	3.3333333333333335	整数相除结果为浮点数
//	整除	3	10//a	3	结果为不大于商的最大整数
%	取余数	3	10 % a	1	也称为模运算
+	加	4	10.0+a	13.0	浮点数与整数运算结果为浮点数
-	减	4	a-10	-7	

说明:

- (1) “-”是单目运算,对操作数取负;其余都是双目运算,即运算符旁有两个操作数。
- (2) 运算优先级表示当表达式中含有多个操作符时,先执行哪个操作符。
- (3) 与 C 语言不同,Python 中两个整数相除返回的类型是浮点数。如:  $4/2$ , 结果为 2.0, 而 “//”是整数相除得整数,  $5//2$  得 2。
- (4) 整数和浮点数运算,结果类型为浮点数。如:  $10+5.0$  结果为 15.0。

**【例 2.5】**利用运算符,输入秒数,以小时、分、秒形式显示。

```
x=int(input("输入秒数"))
s = x % 60          # 求得秒
m = x //60 % 60     # 求得分钟
h = x // 3600       # 求得小时
print(h,":",m,":",s)
```

程序运行:

```
输入秒数 200000
55:33:20
>>>
```

思考: 如何将一个三位数倒置,例如  $n=345$ , 倒置后为 543。

### 2.赋值运算符

程序中的变量在进行具体的运算之前,都需要有明确的值,Python 中对变量的赋值提供了多种方式。

- (1) 简单赋值

这是一般程序设计语言最常见和常用的格式，即：

变量=表达式

作用：计算右边表达式的值，赋值给左边的变量。

例如：y=x\*\*2+6

注意：赋值号的左边只能是变量，不能是常量、表达式。下面均为错误的赋值语句：

```
>>> sin(x)=x+y      #左边是表达式（函数）
>>> 5=sqrt(s)+x+y    #左边是常量
>>> x+y=z            #左边是表达式
```

## （2）复合赋值

Python 中还为上述运算符（表 1.2.2 中除负号单目运算符外）提供了与之对应的复合赋值（或称增强赋值）运算符：+=、-=、\*=、/=、//=、%=、\*\*=。复合赋值语句的形式为：

变量 复合赋值运算符 表达式

作用：计算赋值运算符右边表达式的值，然后与左边的变量进行相应的运算，最后赋值给变量。例如：

a\*=b+3      等价于：a=a\*(b+3)

复合赋值常用于累加、计数和连乘运算，例如：

①累加 sum += x    等价于 sum = sum+x

②计数 n += 1      等价于 n = n+1

③连乘 t \*=x       等价于 t=t\*x

## （3）链式赋值

链式赋值将同一个值赋给多个变量。例如：

x=y=z=10.3                      # x、y、z 三个变量获得相同值 10.3

## （4）序列解包赋值

序列解包赋值指用同一个赋值号“=”，给不同的变量分别赋值。例如：

x,y,z=1,2,3                      # x、y、z 依次获得 1、2、3 的值

序列解包还可以方便实现交换变量的值，例如：

x,y=y,x                          # 变量 x 与 y 交换值

## 3.数据类型的转换

在算术运算中，如果操作数具有不同的数据类型，则使用隐式或显式进行类型转换。

### （1）隐式转换

在源代码中不需要任何特殊语法自动进行，称隐式转换。隐式转换规则是整型向浮点型转换、浮点型向复数类型转换，可避免在转换时造成的误差损失。

### （2）显式转换

利用转换函数来实现类型强制转换，数值类型转换函数使用的是 int()、float()和

complex(), 见表 1.2.4 所示。

**【例 2.6】** 如下代码显示了隐式转换与显式转换。

```
>>> 3.0+4          # 浮点数与整数运算，整数类型隐式转换成浮点数类型
7.0
>>> int(7.5)       # 通过 int()函数，将浮点数转换成整数
7
```

## 2.2.3 常用数值类函数

程序设计语言中的函数如同数学函数的概念，完成某个特定功能并获得某种返回。

Python 功能强大之处在于提供了丰富的库函数，以模块的方式，便于管理和使用。

Python 中的函数分为四类：

- 内置函数：内置模块 `__builtins__` 中，可以直接调用；
- 标准库函数：先导入模块再调用函数，常用的标准库有 `math` 库和 `random` 库；
- 第三方库：先安装库再导入模块然后才能调用函数，例如 `numpy` 库等；
- 用户自定义函数：是用户自己编写的函数，这将在第 6 章介绍。

本章主要介绍前两类函数，使用函数提高代码重用和编程效率。函数调用形式：

函数名（参数列表）

注意：可以使用 `help` (函数名) 查看某个函数的详细用法。

### 1. 内置的数值类型函数

内置的数值类型函数见表 1.2.3 所示：

表 1.2.3 内置的数值类型函数

函数	含 义	实 例	结 果
<code>abs(x)</code>	取 x 的绝对值	<code>abs(-3.5)</code>	3.5
<code>divmod(x,y)</code>	结果为一元组：(x//y,x%y)	<code>divmod(10,3)</code>	(3,1)
<code>pow(x,y)</code>	幂运算，即 x 的 y 次幂	<code>pow(3,4)</code>	81
<code>round(x[,n])</code>	对 x 四舍五入，保留 n 位小数 n 可以缺省，表示无小数位。	<code>round(10.4456)</code>	10
<code>max(x1,x2,...xn)</code>	获得所有参数的最大值	<code>max(13,56,89,33,65)</code>	89
<code>min(x1,x2,...xn)</code>	获得所有参数的最小值	<code>min(13,56,89,33,65)</code>	13

### 2. 内置的数据类型转换函数

内置的数据类型转换函数见表 1.2.4 所示。

表 1.2.4 内置的数据类型转换

函数名	描述	实例	结果
<code>int(x)</code>	将 x 浮点数取整，即去除小数部分	<code>int(2.5)</code>	2
		<code>int("23")</code>	23
<code>float(x)</code>	将 x 转换成浮点数	<code>float(3)</code>	3.0
		<code>float("23.4")</code>	23.4

<code>complex(x1[,x2])</code>	将 x1、x2 转换成复数 缺省 x2，虚部为 0	<code>complex(3,4)</code> <code>complex(3)</code>	(3+4j) (3+0j)
<code>str(x)</code>	将 x 转换成字符串	<code>str(23.4)</code> <code>str(True)</code>	'23.4' 'True'
<code>bool(x)</code>	将 x 转换成布尔类型。若 x 为数值， 非 0 为 True，0 为 False；若 x 为字符串， 非空为 True，空为 False。	<code>bool("23.4")</code> <code>bool(0)</code> <code>bool("")</code>	True False False
<code>eval(str)</code>	将字符串 str 当成有效表达式来求值， 并返回计算结果。	<code>eval('3+4')</code> <code>eval('True')</code> <code>eval('3,4,5')</code>	7 True (3,4,5)

说明：`eval(str)`是功能很强的函数，根据其函数的特点功能，可拓展如下常见的应用。

(1) 利用该函数的功能，计算表达式的值。

**【例 2.7】**利用 `eval` 函数编写通用、简单计算的程序。

```
from math import *
x=eval(input("输入表达式: "))
print("计算结果=",x)
```

程序运行：

```
输入表达式: log(10)
计算结果= 2.302585092994046
>>>
输入表达式: sin(3.14/2)
计算结果= 0.9999996829318346
>>>
```

(2) 用作类型转换，即将任意字符串智能转化为其合理的对应值。

```
>>> eval('True')           # True      将字符串转换为布尔类型
>>> eval('1.2e4')          # 12000.0   将字符串转换为浮点数
```

(3) 多个数值型变量的输入

```
>> x,y,z=eval(input("输入 x,y,z: "))
输入 x,y,z: 3,4,5
>>> x,y,z
(3, 4, 5)
```

### 3. math 库

使用数学库函数，需导入 `math` 模块。

**【例 2.8】**输入角度，显示对应的正弦值和余弦值。

```
import math
x=int(input("输入角度:"))
y= math.sin(x*math.pi/180)      # 输入的是角度，函数要求是弧度
print("x=",x,"sin(x)=",y)
```

程序运行：

```
输入角度:30
x= 30 sin(x)= 0.49999999999999994
```

注意：若用 `from math import *` 方式导入 `math` 模块，则调用时可省略模块名，即：



$$y=\sin(x*\pi/180)$$

常用的数学函数和常数如表 1.2.5 所示。

表 1.2.5 常用的数学函数和常数

函数	含 义	实 例	结 果
pi	常数 $\pi$	pi	3.141592653589793
e	常数 e	e	2.718281828459045
cos(x)	返回 x 弧度的余弦值	cos(0)	1.0
exp(x)	返回以 e 为底的幂，即 $e^x$	exp(3)	20.085536923187668
gcd(m,n)	返回 m,n 两数的最大公约数	gcd(12,8)	4
log(x)	返回自然对数（以 e 为底）	log(10)	2.302585092994046
log10(x)	返回常用对数（以 10 为底）	log10(10)	1.0
sin(x)	返回 x 弧度的正弦值	sin(0)	0.0
sqrt(x)	求 x 的平方根	sqrt(9)	3.0
tan(x)	返回 x 弧度的正切	tan(0)	0.0

#### 4. random 库

random 库中包含了可以生成不同类型、范围的随机数函数，表 1.2.6 列出常用的随机数生成函数。同样调用 random 库中函数，必须导入 random 模块。

表 1.2.6 常用的随机数生成函数

函数	描述	示例	结果
random()	随机产生[0,1)区间内的一个浮点数	random()	0.64125066849 65525
randint(a,b)	随机产生[a,b]区间内的一个整数	randint(65,90)	75
randrange(m,n[,k])	随机产生[m, n)之间以 k 为步长的随机整数，缺省 k 为 1	randrange(1,101,10)	41
sample(seq,k)	从序列 seq 中随机抽取 k 个元素组成列表， $k \leq \text{len}(\text{seq})$	sample(range(1,10),5)	[3, 2, 1, 9, 5]

**【例 2.9】**随机产生 10 个大写字母。

分析：大写字母的编码值区间为 65~90，已知编码值转换成字母的函数是 chr()。

程序如下：

```
from random import *
lst=[]
for i in range(0,10):
    lst+=chr(randint(65,90))
print(lst)
```

运行三次结果：

```
['E', 'O', 'O', 'Y', 'F', 'W', 'M', 'X', 'L', 'N']
['N', 'Z', 'Z', 'C', 'O', 'T', 'V', 'Y', 'M', 'T']
['B', 'V', 'T', 'Z', 'M', 'X', 'S', 'W', 'N', 'M']
```

说明：每次运行程序，产生不同的序列，若要为了程序调试需要，产生相同的序列，在程序的循环结构前加语句：`seed(任意整数)`，表示种子数不变。则运行三次结果：

```
['E', 'O', 'O', 'Y', 'F', 'W', 'M', 'X', 'L', 'N']
['E', 'O', 'O', 'Y', 'F', 'W', 'M', 'X', 'L', 'N']
['E', 'O', 'O', 'Y', 'F', 'W', 'M', 'X', 'L', 'N']
```

函数形式：`seed([n])`，作用初始化随机数种子。`n` 为整数，生成序列固定；缺省参数 `n` 或参数为 `None`、非整数，则为当前系统时间，生成序列可变。

## 2.3 字符串类型和处理

### 2.3.1 字符串类型

字符串是程序设计中非常常见的数据类型，它是由字符组成的序列。例如，"Python 程序设计"、'2020'等。

Python 中全面支持中文，在统计字符串长度时，中文字符和西文字符都视为一个字符来对待；默认对字符串采用的是 Unicode 编码，任一字符用两字节表示。

#### 1. 字符串常量

字符串常量是由字符串定界符括起的字符序列，在 Python 中，字符串定界符由三种表示形式：

- (1) 用一对单引号包括的单行字符序列。
- (2) 用一对双引号包括的单行字符序列。
- (3) 用两边都是 3 个单引号或 3 个双引号包括的多行字符序列。

对于单行字符串用单引号或双引号没有区别，主要是若字符串中包含了双引号，则用一对单引号括起字符串，反之亦然。

例如：'他说："Python 语言学习很容易！"'、"what's your name?"等都是字符串常量。

#### 2. 转义字符

问题的提出：除了可显示的字符外，还有一些控制字符如何表示？若单引号作为定界符，而字符序列中有单引号又如何表示？

例如：'what's your name?' 怎么书写？

为表示控制符等一些具有特殊功能的符号，Python 提供了一种称为转义表示的方法：就是以反斜杠加上其它特定字符来表示一个特殊功能符号，称该符号为转义字符。例如，

'what\'s your name?', 这里 “\’” 就是转义字符，表示单引号。

常用的转义字符如表 1.2.7 所示：

表 1.2.7 常用的转义字符

转义字符	描述	转义字符	描述
\\	反斜杠符号	\n	换行
\'	单引号	\r	回车
\"	双引号	\t	横向制表符

说明：

（1）对于 “\t” 横向制表符所占的列数与开发环境有关。例如 IDLE 环境占 8 列、Spyder 占 4 列；所以同样的代码在不同的开发环境输出的效果将不同。

（2）Python 语言还允许在字符串前加 r，即：r"内容"或 r'内容'，表示字符串内的内容不转义。

【例 2.10】 应用举例。在 IDLE 开发环境下，利用字符串常量、转义字符和操作符将学生信息以表格形式输出。

```
print(25*" ") # “*” 为对字符串复制，即显示 25 个下划线字符
print("|学号\t姓名\t计算机\t|") # “\t” 跳到下一个制表位（默认一个制表位占 8 列）
print(25*" ")
print("|180001\t张三\t优秀\t|")
print(25*" ")
print("|180002\t李小明\t良\t|")
print(25*" ")
print("|180003\t王平平\t不及格\t\r")
print(25*" ", "\n") # 本身 print 结束要换行，加了 “\n” 实质是空一行
print(r"|180003\t王平平\t不及格\t\r") # r"字符串"内容不转义，照样输出
```

程序运行如图 1.2.3 所示：

学号	姓名	计算机
180001	张三	优秀
180002	李小明	良
180003	王平平	不及格
180003\t	王平平\t	不及格\t\r

图 1.2.3 转义符应用举例

### 2.3.2 字符串的处理

字符串的处理包括对字符串的基本操作、常用函数和方法。

#### 1. 字符串基本操作符

Python 语言提供了 5 个字符串的基本操作，如表 1.2.8 所示。

假定 `st1="cde",st2="abcdefg"`

表 1.2.8 字符串的基本操作符

操作符	描述	实例	结果
<code>s1+s2</code>	连接两个字符串	<code>st1+st2</code>	<code>'cdeabcdefg'</code>
<code>n*s</code> 或 <code>s*n</code>	对字符串 <code>s</code> 复制 <code>n</code> 次	<code>3*st1</code> 或 <code>st1*3</code>	<code>'cdecdecde'</code>
<code>s1 in s2</code>	子串测试。 <code>s1</code> 若是 <code>s2</code> 的子串 返回 <code>True</code> ,否则 <code>False</code>	<code>st1 in st2</code>	<code>True</code>
<code>s[i]</code>	索引操作,取索引为 <code>i</code> 的字符	<code>st2[3]</code> 或 <code>st2[-4]</code>	<code>'d'</code>
<code>s[m:n:k]</code>	切片操作,返回索引号 <code>m</code> 到 <code>n</code> 但不包括 <code>n</code> 的子串;省略 <code>m</code> 表示 从头,省略 <code>n</code> 表示取到结束; <code>k</code> 为步长,省略为 1	<code>st2[:4]</code> <code>st2[6:2]</code> <code>st2[3:]</code> <code>st2[:]</code>	<code>'abc'</code> <code>'ace'</code> <code>'defg'</code> <code>'abcdefg'</code>

说明:

(1) 索引

字符串中的每个字符都是有索引号的，通过索引号和索引操作符“`[]`”可以获取单个字符。有两种表示索引方式：正向递增索引和反向递减索引，习惯用正向。例如 `st2` 变量的字符串，正向和方向索引如图 1.2.4 所示。`st2[3]`与 `st2[-4]`都表示通过索引获取到字符：‘d’。

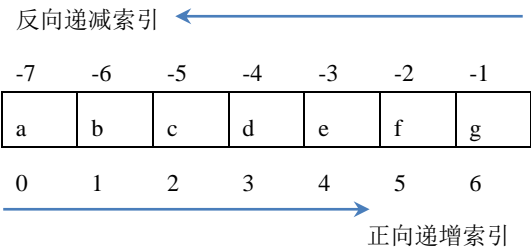


图 1.2.4 序列的正向和反向索引

(2) 切片

利用`[m:n:k]`操作符，可以对字符串任意切片获得所需的子串。

例如：`st2[3:]`、`st2[3:7]` 都表示通过切片获取到子串：‘defg’；而 `st2[:3]`结果为：‘abc’。

注意：切片操作常用于取子串，一般缺省 `k` 为 1，取`[m:n)`区间的子串，即左闭右开区间的子串。

例如：`st2[:3]`结果为‘abc’、`st2[:6:2]`的结果为‘ace’、`st2[3:4]`的结果为‘d’、`st2[3:3]`的结果为“空串”。

**【例 2.11】**提取星期几。已知 s= "星期一星期二星期三星期四星期五星期六星期日"，根据输入的数字（1-7），输出星期几。

分析：s 字符串是有规律的，每个星期几占 3 位；根据输入数字，确定取的起始位即可获取对应的星期。

代码如下：

```
s="星期一星期二星期三星期四星期五星期六星期日"
n=int(input("请输入数字 1-7: "))
start=(n-1)*3
weekDay=s[start:start+3]
print(weekDay)
```

程序运行：

```
请输入数字 1-7: 3
星期三
```

## 2.内置的字符串函数和方法

### （1）字符串函数

Python 提供了一些内置的字符串函数，常用的函数见表 1.2.9 所示：

表 1.2.9 常用内置函数

函数	描述	实例	结果
len(s)	返回 s 字符串长度	len("abcdefg")	7
chr(n)	返回字符编码 n 对应的单字符	chr(65)	'A'
ord(s)	返回 s 单字符对应的编码	ord('A')	65

说明：

①在 Python 中，默认对字符串采用的是 unicode 编码的 str 类型来表示，任一字符用两字节表示。

②常用的西文字符：'0'~'9' 编码值为 48~57，'A'~'Z' 编码值为 65~90，'a'~'z' 编码值为 97~122。

**【例 2.12】**输入一字符串，显示其对应的 Unicode 码，再根据 Unicode 码显示其字符作验证。

分析：利用 len()函数获得输入的字符串长度，然后利用循环结构依次取出每个字符，调用 ord()函数获得 Unicode 码，再调用 chr()函数作验证。

代码如下：

```
s=input("请输入字符串: ")
n=len(s)                # 调用 len()函数求得字符串长度
for i in range(n):       # 利用循环结构依次取每一个字符
    c_ord=ord(s[i])      # 调用 ord()函数获得字符对应的编码值
    ord_c=chr(c_ord)     # 已知编码值调用 chr()函数获得对应的字符
```

```
print(s[i], " ord=", c_ord, " ord_c=", ord_c) # 显示字符、字符对应的编码值和对应字符
```

程序运行如图 1.2.5 所示。

```
请输入字符串: Python程序设计
P  c_ord= 80  ord_c= P
y  c_ord= 121 ord_c= y
t  c_ord= 116 ord_c= t
h  c_ord= 104 ord_c= h
o  c_ord= 111 ord_c= o
n  c_ord= 110 ord_c= n
程 c_ord= 31243 ord_c= 程
序 c_ord= 24207 ord_c= 序
设 c_ord= 35774 ord_c= 设
计 c_ord= 35745 ord_c= 计
```

图 1.2.5 运行效果

从该例看出 `ord()` 函数与 `chr()` 函数是一对反函数，即 `chr(ord("p"))="p"`，同样 `ord(chr(65))=65`。

(2) 字符串方法

Python 中每一个数据类型都有一个相对应的内置类，每新建一个数据，实际就是初始化并生成一个对象。系统为对象提供了一些方法便于功能处理。

方法引用形式：字符串.方法（参数）

要注意：方法作用于对象，方法引用结果获得一个处理结果，但不改变引用对象。

系统提供了很多方法，在编辑环境中，在字符串变量的后面输入“.”后，稍等会自动弹出字符串对象的所有方法，如图 1.2.6 所示，移动光标选择所需的方法。

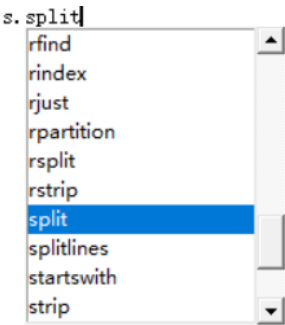


图 1.2.6 字符串对象的方法

常用的方法见表 1.2.10 所示。假定如下举例中均以 `s` 变量操作：`s1="ABC DEFG BC"`，“实例”列是对方法的引用，返回结果见最右列，不改变原字符串对象 `s` 的值。

表 1.2.10 常用内置方法

方法	描述	实例	结果
<code>find(s)</code>	在字符串中找第一个出现 <code>s</code> 子串的位置，找不到返回 -1	<code>s1.index("BB")</code>	-1
<code>lower()</code>	全部转换成字母小写	<code>s1.lower()</code>	'abc defg bc'
<code>isnumeric()</code>	字符串全部是数字返回 True，否则 False	<code>s1.isnumeric()</code> <code>"123".isnumeric()</code>	False True
<code>isalpha()</code>	字符串全部是字母返回 True，否则 False	<code>s1.isalpha()</code> <code>"asDF".isalpha()</code>	False True
<code>index(s)</code>	在字符串中找第一个出现 <code>s</code> 子串的位置，找不到抛出异常	<code>s1.index("BC")</code>	1
<code>replace(old,new)</code>	将字符串中用 <code>new</code> 子串替换 <code>old</code> 子串	<code>s1.replace("BC","WYZ")</code>	'AWYZ DEFG WYZ'

方法	描述	实例	结果
split([s])	将字符串按照 s 分隔符分离，返回结果是列表。 缺省 s 按空格、制表符或换行符分隔	若 s2="a 12 34" s2.split() s2.split(" ")	[ 'a', '12', '34' ] [ 'a', '', '12', '34' ]
startswith(s [,start[,end]])	字符串(在可选的范围内)是否是以 s 开头，若有 end，表示不包含 end 的整数	s1.startswith("AB") s1.startswith("BC,1,3")	True True
strip(s)	去掉字符串两旁的 s 子串，一般用于去掉空格	" 123 ".strip()	'123'
upper()	全部字母大写	"abcd".upper()	'ABCD'

说明：对于 split() 函数，若以 split(" ") 单空格分离，当字符串中连续出现多个空格时，分离的效果比较差。改进的方法为缺省参数，系统就会优化处理，即当连续出现多个空格作为一个空格分离，效果比较好。

**【例 2.13】** 对 s1 字符串引用 replace 方法，运行如下：

```
>>> s1="ABC DEFG BC"
>>> s1.replace("BC","WYZ")
'AWYZ DEFG WYZ'
>>> s1
'ABC DEFG BC'
```

**【例 2.14】** 利用 split 方法对输入的时间按小时、分、秒分离。

```
s=input("输入时间：小时：分：秒 ")
hour,minute,second=s.split(":")
print(hour,"小时，",minute,"分，",second,"秒")
```

程序运行：

```
输入时间：小时：分：秒 12:23:54
12 小时， 23 分， 54 秒
```

## 2.4 布尔类型和处理

布尔数据类型用于描述判断的结果，具有真和假两个值。在 Python 中用 True 和 False 表示真和假，注意首字母大写其余小写。

以真和假为值的表达式称为布尔表达式，用于表示某种条件成立与否，一般在选择结构和循环结构中用于分支和循环条件判断。

注意：在 Python 中，布尔类型值也可以当作数值来使用。当参与数值表达式计算时，True 自动转化为 1，False 转化为 0。例如：

```
>>> x=3+True
>>> x
4
```

## 2.4.1 关系运算

关系运算的作用是判断两个表达式值得大小关系，，结果为布尔类型 `True` 或 `False`。

关系运算表达式形式：

操作数 1 关系运算符 操作数 2

操作数：可以是数值型和字符串，按其数值或字符串的编码值大小比较。

关系运算符有：`==`、`>`、`>=`、`<`、`<=`、`!=`，分别表示等于、大于、大于等于、小于、小于等于、不等于。

注意：

(1) 等号“`==`”与赋值号“`=`”的区别

```
>>> x=20          # x 通过赋值号运算获得值 20
>>> x==20         # 根据 x 的值与 20 进行比较是否相等？结果为 True 或 False
```

(2) 浮点数的相等比较

计算机中的浮点数有时是近似值存储数据（因浮点数十进制与二进制互转因素），当两个浮点数进行相等比较时结果会不正确，应采用两个数之间的差值小于给定的精度。

**【例 2.15】** 两浮点数相等比较。

```
>>> 3.5-2.7==0.8
False
>>> 3.5-2.7
0.7999999999999998
>>> abs((3.5-2.7)-0.8)<0.000000001
True
```

(3) 字符串的比较

两个字符串进行比较，则按字符的 `ASCII` 码值从左到右逐一进行比较，即首先比较两个字符串中的第 1 个字符，其 `ASCII` 码值大的字符串为大；如果第 1 个字符相同，则比较第 2 个字符，以此类推，直到出现不同的字符时为止。

**【例 2.16】** 字符串的比较例。

```
>>> "ABCDE" > "ABR"
False
>>> "BC" < "bc"
True
```

## 2.4.2 逻辑运算

常用的关系运算只能进行简单的逻辑判断。当判断的条件复杂时，需要逻辑运算符将多个条件进行组合。逻辑表达式通过逻辑运算符，可以将关系表达式、布尔型常量或变量进行逻辑运算，运算结果是 `True` 或 `False`。除 `not` 是单目运算符外（即仅右边有操作数），其余是双目运算符。表 1.2.11 列出 Python 中的常用逻辑运算符。



表 1.2.11 常用逻辑运算符

运算符	含义	优先级	举例	说 明
not	取反	1	not x	当 x 的值为 False 时，逻辑表达式的值为 True 当 x 的值为 True 时，逻辑表达式的值为 False
and	与	2	x and y	当 x、y 有一个 False 时，逻辑表达式的值为 False 只有当 x、y 均为 True 时，逻辑表达式的值才为 True
or	或	3	x or y	当 x、y 有一个 True，逻辑表达式的值为 True 只有当 x、y 均为 False 时，逻辑表达式的值才为 False

注意：

(1) 在 Python 中也支持数值或字符串的表达式参与逻辑运算，规则是对数值型非 0 为 True，0 为 False；对于字符串非空为 True，空为 False。同样，当逻辑量进行算术运算时，True 自动转换成 1，False 转换为 0，然后参与运算。

例如：

```
>>> True+3
4
>>> False+3
3
```

(2) 对数学上常用于表示值域范围的表达式，在 Python 的逻辑运算中，则可采用与数学上相同书写的逻辑表达式。

例如，要表示 x 的值域范围  $0 \leq x < 10$ ，则简化的逻辑表达式为： $0 \leq x < 10$ ，这在其他程序设计语言中是不支持的，必须写成： $0 \leq x$  and  $x < 10$ 。

**【例 2.17】**输入成绩，判断该生成绩是否优秀，优秀生成绩为 90~100。

```
>>> mark=int(input("输入成绩: "))
输入成绩: 93
>>> 90<=mark<=100
True
```

**【例 2.18】**输入三角形三边，判断其能否构成三角形。

```
>>> a, b, c = eval(input("输入 a,b,c: "))
输入 a,b,c: 2, 4, 5
>>> a+b>c and b+c>a and a+c>b
True
```

说明：eval 函数将字符串中去掉引号后，其中的内容看作表达式进行计算，其实质是前面赋值运算符中介绍的序列解包赋值。该函数是 Python 中很有特色的函数，作用见表 1.2.4 所示，在以后多变量的输入中常使用。

(3) 运算优先级

当一个表达式中，多种不同类型的运算符出现时，运算符优先级如下：

算术运算符>关系运算符>逻辑运算符

当然，算术运算符、逻辑运算符各自都有不同的优先级，关系运算符优先级相同。

注意：对于多种运算符并存的表达式，可增加圆括号，改变计算的顺序也可使表达式更清晰。

**【例 2.19】** 选拔优秀学生。若选拔优秀生的条件为：年龄(Age)小于 19 岁，三门课总分(Total)高于 285 分且其中有一门为 100 分，如果其表达式写为：

```
age<19 and total>285 and mark1 ==100 or mark2 ==100 or mark3==100
```

有何问题？应如何改正？请读者考虑。

## 2.5 组合数据类型简介

为了更好地描述和处理客观世界中的成批数据，满足不同的需要，Python 还提供了列表、元组、字典和集合等组合数据类型。本节简要介绍这些数据类型的创建，便于大家建立对 Python 数据类型的整体认识和使用，更详细的用法在第 4 章中介绍。

### 2.5.1 列表和元组

Python 中的字符串、列表和元组是 Python 中常用的有序类型，它们的每个元素按照位置编号即索引来顺序存取，如同其他程序设计语言的数组。但数组通常只能存储相同数据类型的元素，而列表和元组则可以存储不同类型的元素。

列表和元组在很多操作上是一样的，如索引、切片、各种数据的统计等，不同的是表示形式不同，前者是方括号“[]”，后者是圆括号“()”，更大区别是列表是可修改的，元组是不可修改的，元组可以看成是只读的列表。

#### 1. 创建列表

创建列表最简单的方法是利用列表表示形式直接创建。

**【例 2.20】** 利用列表表示形式直接创建。

```
>>> fibs=[0,1,1,2,3,5,8,13]          # 建立了一个斐波那契数列列表
>>> stud=[["黄小明",87],["李平",79],["章一",99]]  # 建立由 3 名学生的姓名成绩构成的嵌套列表
>>> num1=["一二三四五六"]           # 建立了仅有一个元素的列表["一二三四五六"]
>>> nl=[]                             # 建立空列表
```

#### 2. 创建元组

创建元组的方法类似创建列表，当利用元组表示形式直接创建时只要用圆括号(可以缺省)代替方括号。

**【例 2.21】** 直接建立元组。

```
>>> tfibs=(0,1,1,2,3,5,8,13)          # 建立元组：(0,1,1,2,3,5,8,13)
>>> tfibs=0,1,1,2,3,5,8,13           # 与上式等价，建立元组：(0,1,1,2,3,5,8,13)
>>> tstud=(("黄小明",87),("李平",79),("章一",99))# 建立由 3 名学生的姓名成绩构成的嵌套元组
>>> tnl=()                            # 建立空元组
```

注意：当建立的元组仅有一个元素时，要在后面加逗号，不然建立的就不是元组 `tuple`。

例如：

```
>>> tone=(123,)          # 是元组：(123,)
>>> tone=(123)           # 不是元组，是整数 123
```

### 3.通用序列操作

通用序列操作指的是对有序序列类型包括列表、元组和字符串其操作是一致的。

这些操作符包括索引、切片、序列相加、乘法运算，在字符串类型节已经介绍，如图

1.2.4 和表 1.2.8 所示。

**【例 2.22】** 列表的序列操作。

```
>>> fibs=[0,1,1,2,3,5,8,13]    # 建立 fibs 列表：[0,1,1,2,3,5,8,13]
>>> fibs[4]                    # 通过索引访问列表中元素值：3
>>> fibs+[21]                  # 两个列表加操作结果：[0, 1, 1, 2, 3, 5, 8, 13, 21]
>>> fibs*2                     # 将列表复制 2 次：[0, 1, 1, 2, 3, 5, 8, 13, 0, 1, 1, 2, 3, 5, 8, 13]
>>> fibs[3:5]                  # 通过切片访问列表中元素：[2, 3]
```

### 4.元组应用场合

元组除了用于表达固定数据外，还常用如下场合，例如多变量同步赋值即序列解包、函数返回多个值等。

说明：把一个序列（列表、元组、字符串等）直接赋给多个变量，此时会把序列中的各个元素依次赋值给每个变量，但是元素的个数必须和变量个数相同，这称为序列解包。

**【例 2.23】** 元组常用场合示例。

```
>>>x,y,z=(10,20,30)          # 元组序列赋值给多个变量，括号可省
>>>x,y=y,x                    # 元组(y,x)的元素依次赋值给 x,y 变量，实现两数交换
>>>print("x=%d,y=%d"%(x,y))  # 格式输出中的输出元素是元组
>>>def f(x):
    return x**2,x**(1/2)      # 函数返回两个值，默认是元组类型
```

## 2.5.2 字典和集合

字典和集合都属于无序的数据类型，因此不能像列表和元组通过位置索引来访问数据元素。

### 1.字典

实际使用的字典工具书以字或词为单位以及含义信息组成，按一定顺序排列，通过字或词获得其含义信息。Python 中的字典与字典工具书类似，由 `key-value` 键值对组成，通过 `key` 可以快速地找到其对应的值。

#### （1）创建字典

通过字典表示形式直接创建，即由大括号“`{}`”括起、逗号分隔、键值对构成的，键必须唯一。键值对形式为“键:值”。

**【例 2.24】** 建立各星期与其英文缩写的字典。

```
>>> week1={1:'Mon',2:'Tues',3:'Wed',4:'Thur',5:'Fri',6:'Sat',7:'Sun'}      #字典表示形式创建
>>> week1
{1: 'Mon', 2: 'Tues', 3: 'Wed', 4: 'Thur', 5: 'Fri', 6: 'Sat', 7: 'Sun'}
```

## (2) 字典应用

Python 中的字典是内置的映射类型，即通过键获得对应的值。利用这样的特点，可以方便地进行字典中元素的检索、插入、删除和统计等操作。

在例 2.24 中，若要显示周三的英文缩写，则语句如下：

```
>>> week1[3]
'Wed'
```

## 2.集合

集合类型与数学中的集合概念是一致的，其主要作用是消除重复元素、进行集合运算。

### (1) 创建集合

通过集合表示形式直接创建，即大括号“{}”括起、逗号分隔的数据项。

**【例 2.25】直接建立集合。**

```
>>> sfibs={0,1,1,2,3,5,8,13}      # 建立 sfibs 集合{0, 1, 2, 3, 5, 8, 13}
>>> sh={"hello!"}                # 直接建立 sh 集合{"hello!"}
>>> sh2=set("hello!")            # 利用函数建立 sh2 集合{'!', 'h', 'l', 'e', 'o'}
```

**注意：**字典和集合表示形式都是一对“{}”括起，那么如下语句，到底建立的是空字典还是空集合：

```
>>> week1[3]
'Wed'
>>> s={}
>>> s
{}
>>> type(s)
<class 'dict'>
>>>
```

通过交互方式运行可知，创建的是空字典，若要创建空集合，必须通过 set() 函数，如下：

```
>>> s=set()
>>> type(s)
<class 'set'>
>>> s
set()
```

## (2) 集合应用

**【例 2.26】利用集合特点过滤重复数据。**

```
>>> score=["计算机","数学","外语","计算机","物理"]      # 建立课程 score 列表
>>> score
['计算机', '数学', '外语', '计算机', '物理']
>>> s=set(score)      # 列表转化为集合，过滤掉重复的课程
```

```
>>> s
{'数学', '计算机', '外语', '物理'}
```

说明：组合数据类型可以通过类型转换函数进行相互转换，类型转换函数分别为 `list()`、`tuple()`、`set()`和 `dict()`。

**【例 2.27】** 已知建立了斐波那契数列列表，分别进行类型转换，代码和执行效果如下：

```
>>> fibs=[0,1,1,2,3,5,8,13]
>>> tuple(fibs)                # 列表转换成元组
(0, 1, 1, 2, 3, 5, 8, 13)
>>> set(fibs)                  # 列表转换成集合
{0, 1, 2, 3, 5, 8, 13}
>>> dict(fibs)                 # 列表转换成字典出错，银字典每个元素为键值对
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    dict(fibs)
TypeError: cannot convert dictionary update sequence element #0 to a sequence
>>>
```

## 2.6 数据的输入和输出

程序的一般结构是输入数据，处理数据、输出结果。程序的输入/输出分为两大类：一类是人—机交互，把人们可以识别的形式（字符串、数）进行输入和输出；另一类是程序之间以文件形式数据传送。本节介绍第一类。

### 2.6.1 数据的输入

Python 语言利用 `input()`函数实现输入，调用形式为：

```
input([提示字符串])
```

说明：`input` 函数返回的是字符串类型。可通过 `int()`、`float()`函数进行所需类型转换；也可通过 `eval()`函数对多个变量进行输入。

**【例 2.28】** 各种类型数据的输入

```
>>> x=input("输入 x:")          # x 获得的是字符串类型
输入 x:123
>>> x
'123'
>>> y=int(input("输入 y:"))      # 通过 int()函数强制类型转换，y 获得的是整数类型
输入 y:123
>>> y
123
>>> n,x,s=eval(input("请输入 n,x,s:"))  # eval()根据输入的值智能转换成相应的类型
请输入 n,x,s:10,67.8,"成绩"
```

```
>>> n,x,s
(10, 67.8, '成绩')
```

## 2.6.2 数据的输出

Python 语言有两种输出方式：表达式方式和 `print()` 函数方式。

### 1. 表达式的方式

在 IDLE 交互状态下，输入表达式可直接输出表达式的值，这种方式常用于数据的检查。

例如：

```
>>> x=100

>>> x/3

33.333333333333336
```

### 2. `print()` 函数方式

使用 `print` 函数方式可以控制输出的格式，这是常用的方式。形式：

```
print([输出项列表], sep=分隔符[, end=结束符])
```

`print()` 函数参数全部可以缺省，若没有参数，则输出一个空行。参数说明如下：

- 输出项列表 是以逗号分隔的表达式；
- `sep` 表示各输出项间的分隔，缺省为空格；
- `end` 表示输出的结束符，默认值为换行符，也就是 `print` 默认输出后换行。

**【例 2.29】** 输出 1~9 的数及平方数。

```
>>> for i in range(1,10):                                # for 是循环结构语句，range(1,10)产生[1~9]间的序列
    print(i,i*i,sep=" ",end=";")
1,1;2,4;3,9;4,16;5,25;6,36;7,49;8,64;9,81;
```

若要每行输出一对，用默认方式空格分隔、每对结束换行，输出 3 对，则代码如下：

```
>>> for i in range(1,4):
    print(i,i*i)

1 1
2 4
3 9
```

若要每个数据项占 5 位，如何实现？则涉及到下面介绍的格式化输出。

## 2.6.3 格式化输出

为了使得输出的结果更美观、规范化，例如保留小数位数、每项输出宽度等，可使用格式化控制输出的格式。Python 语言有两种方式控制格式输出：

- (1) “%” 方式（如同 C 语言的格式控制符）；

(2) format()方法。

Python 后续的版本中不再使用 “%” 方式，将主要使用 format()方法。

## 1. “%” 方式

**【例 2.30】**输入 a、b、c 三个系数，计算一元二次方程的两个实根，保留三位小数。

```
from math import *
a,b,c=eval(input("输入 a,b,c: "))
d=b*b-4*a*c
if d>=0 :
    x1=(-b+sqrt(d))/(2*a)
    x2=(-b-sqrt(d))/(2*a)
    print("x1=%8.3f\nx2=%8.3f"%(x1,x2))      # %8.3f 浮点数输出宽度 8 位，小数位数 3 位
else:
    print("无实根")
```

程序运行：

```
输入 a,b,c: 2,7,5
x1=  1.000          # 表示空一格
x2=-2.500
```

从上例可以看出，格式化输出时用操作符 “%” 分隔格式字符串与输出项，即：

**格式控制字符串 % (输出项列表)**

其中：

① **格式控制字符串作为模板**，模板中有显示的原样字符（包括转义符）和格式说明符，**格式说明符**为输出值预留位置；输出序列将多个值依次传递给模板中对应格式说明符。

② **格式说明符常用形式如下：**

% [m][.n]类型符

其中：m 为输出最小宽度；n 为小数位数；类型符为对应输出数据类型。常用的**格式说明符**见表 1.2.12 所示：

表 1.2.12 常用的格式说明符

格式说明符	%d	%c	%s	%f 或 %F	%e 或 %E
类型符	d	c	s	f 或 F	e 或 E
对应输出数据类型	整数	单个字符	字符串	浮点数	指数形式

**【例 2.31】**字符串格式化操作符%输出应用。

```
已知 x=1234.567, n=1234567, s="pythonOk", 则:
>>>print("x=%6.2fs=%sn=%d"%(x,s,n))
x=1234.57s=pythonOkn=1234567
>>>print("x=%13.5fs=%13sn=%13d s[1]=%c"%(x,s,n,s[1]))
x=  1234.56700s=  1234567pythonOkn=  1234567  s[1]=y
```

注意：

(1) 当输出值整数位数多于最小宽度，按实际值输出；输出值小数位数多于精度位数，

则四舍五入按精度位数输出；输出小数位数少于精度位数，右补 0；输出值位数少于最小宽度，则右对齐左补空。

(2) 当输出项的数据类型与格式符不一致时，若格式符是 %s，输出项是非字符串类型，自动转换为字符串再输出；反之若输出项是字符串，而格式符是非 %s，则会产生错误信息。

例：

```
>>> print("x=%13.5fa=%13sn=%13d"%(x,n,a))          #a 字符串变量对应格式符%13d
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    print("x=%13.5fa=%13sn=%13d"%(x,n,a))
TypeError: %d format: a number is required, not str
```

## 2. format 方法

format()方法更清晰地表示对字符串进行格式化处理。形式：

**格式模板**.format (输出项列表)

**格式模板**中有显示的原样字符（包括转义符）和用若干个“{}”也称为“槽”作为格式占位符，为输出值预留位置；输出项列表将多个值依次传递给模板对应的“{}”。

“{}”的形式：{[序号:[m.n]类型符]}

其中：

①“{ }”中可以是带格式的串，也可以空；序号与后面的格式控制格式间用“:”分隔；同样 m 为输出最小宽度；n 为小数位数，类型符同表 1.2.12。

②序号为输出项列表的序号，第 1 个元素序号为 0，写在{ }中的起始位置；若无序号，则按参数顺序对应。

**【例 2.32】**format 方法输出应用。

已知 x=1234.567，n=1234567，s="pythonOk"，则：

```
>>> print("x={ }s={ }n={ }".format(x,s,n))          #最简单的占位符按默认格式输出
x=1234.567s=pythonOkn=1234567
>>> print("x={0:12.2f}s={1:10s}n={2:10d}".format(x,s,n))  #按占位符设置的格式输出
x=    1234.57s=pythonOk    n=    1234567
```

说明：从输出的两种方式看，“ % ”方式当设置的最小宽度大于输出值的宽度，数值和字符串类型默认均右对齐左补空；而 format()方式默认数值类型右对齐左补空、字符串类型默认左对齐右补空。

## 2.7 综合应用

本章我们围绕着数据和数据处理进行了介绍，包括数据类型、变量的声明、常量的表示；各种数据类型、运算符、函数和方法等。这些内容都是后面几章学习所必需的基础，对初学者一下子不可能理解和掌握这些内容，通过实践很多问题就迎刃而解。

**【例 2.33】**输入姓名，利用输出格式设计制作生日贺卡，程序运行如图 1.2.7 所示：





## 习 题

## 一、单选题

1.下面\_\_\_\_\_不是 Python 的基本数据类型。

- A.float                      B.type  
C.bool                        D.int

2.下面\_\_\_\_\_不是 Python 的组合数据类型。

- A.list 和 tuple                      B.str  
C.set 和 dict                         C.complex

3. 下面\_\_\_\_\_是 Python 合法的变量名。

- A.\_yz                      B.123xyz
- C.int                        D.x-y

4.下面\_\_\_\_\_是不合法的整型常数。

- [illegible]

5.下面\_\_\_\_\_是不合法的字符串常数。

- A.abc                      B."abc"
- C.'abc'                    D.""abc""

6. 数学关系  $3 \leq x < 10$  表示成 Python 表达式, 错误的是\_\_\_\_\_。

- A.  $3 \leq x < 10$                       B.  $3 \leq x$  and  $x < 10$   
C.  $10 > x \geq 3$                       D.  $3 \leq x$  or  $x < 10$

7. random()函数不可能产生的值是\_\_\_\_\_。

- |          |          |
|----------|----------|
| A.0      | B.1      |
| C.0.1234 | D.0.0005 |

8. 已知 s="12345678", 则表达式 s[3:6]的值为\_\_\_\_\_。

- A. "3456"                      B. "345"
- C. "456"                        D. "45"

9. input()函数返回值的类型是\_\_\_\_\_。

- A.整型数值                      B.字符串

C.由用户输入的值决定                      D.浮点数

10.下面\_\_\_\_\_不属于 print()函数中的格式类型符。

A.d    B.f 和 e

C. s    D.b

11. 以下注释语句中，不正确的是\_\_\_\_\_。

A. #Python 注释                              B. ''' Python 注释'''

C. """Python 注释"""                      D. // Python 注释

12. 表达式 'ab' + ''' 甲乙\new''' \* 2 的结果字符串长度是\_\_\_\_\_。

A. 12    B. 14

C. 16    D. 18

13. 以下关系表达式中，其值为 False 的是\_\_\_\_\_。

A."ABC">"AbC"                              B."PYTHON"=="python".upper()

C.True>False                                  D."123"<"23"

14. 下面正确的赋值语句是\_\_\_\_\_。

A.x+y=30                                      B.x,y,z=1,2,3

C.10=y=x                                      D.3y=x

15. 下列各项中，不能使用索引运算的是\_\_\_\_\_

A.列表    B.元组

C.集合    D.字符串

16.tuple(range(1,9,2))的返回结果是\_\_\_\_\_。

A.(1, 3, 5, 7)                                  B.[1, 3, 5, 7]

C.(2, 4, 6, 8)                                  D.(1, 3, 5, 7,9)

17.以下不能建立集合的语句是\_\_\_\_\_。

A.s=set()                                      B.s=set{ }

C.s= {1,2,3}                                  D.s=set([1,2,3])

18.以下不能建立字典的语句是\_\_\_\_\_。

A.d={ }    B.d={1:'甲',2:'T 乙',3:'丙',4:'丁'}

C.s=dict(1:'甲',2:'T 乙',3:'丙',4:'丁')    D.d=dict()

19.执行语句: x,y=eval(input( ))时，从键盘上输入数据格式错误的是\_\_\_\_\_。

A.10 20    B.10, 20

C.(10,20)    D.[10, 20]

20. 已知  $x=100, y=15$ , 执行语句: `print("x/y={0:5.2f}x % y={1:3d}".format(x/y, x % y))` 的结果是\_\_\_\_\_。

A.  $x/y = 6.67x \cup \% \cup y = 10$

B. 6.6710

C.  $x/y = \{0:5.2f\}x \cup \% \cup y = \{1:3d\}$

D.  $6.67 = 6.6710 = 10$

21. 假定变量有两个变量  $x$  和  $y$ , 它们的值分别为 20 和 30, 则下面语句中合法的是\_\_\_\_\_。

A. `print('x is %10d, y is %6d'%(x,y))`

B. `print('x is %10d, y is %6d',%(x,y))`

C. `print('x is %10d, y is %6d',x,y)`

D. `print('x is %10d, y is %6d',%x,%y)`

22. 执行下列语句后, 程序运行的结果是\_\_\_\_\_。

`s1="湖南人"`

`s2="不怕"`

`s3="辣"`

`print("{0}{2}{1}".format(s1,s2,s3))`

A. 湖南人不怕辣

B. 湖南人辣不怕

C. 辣不怕湖南人

D. 湖南人怕不辣

23. 若 `string='Hello_World'`, 则 `string[:-6]*2` 的值是\_\_\_\_\_。

A. 'Hello\_Hello\_'

B. 'World\_World\_'

C. 'HelloHello'

D. 执行时报错

24. `print(r'\\')` 和 `print('\')` 结果是\_\_\_\_\_。

A. \和\\

B. \\和\\

C. \和\

D. \\和\

## 二、填空题

1. 把下列数学表达式写成 Python 表达式。

(1)  $|x+y|+z^5$  的 Python 表达式为\_\_\_\_\_。

(2)  $\frac{10x+\sqrt{3y}}{xy}$  的 Python 表达式为\_\_\_\_\_。

(3)  $\frac{-b+\sqrt{b^2-4ac}}{2a}$  的 Python 表达式为\_\_\_\_\_。

(4)  $\sin 45^\circ + \frac{e^{10} + \ln 10}{\sqrt{x+y+1}}$  的 Python 表达式为\_\_\_\_\_。

2. 根据条件写一个 Python 表达式:

(1) 表示  $x$  是 5 或 7 的倍数的表达式为\_\_\_\_\_。

(2) 表示  $10 \leq x < 20$  的表达式为\_\_\_\_\_。

(3)  $x, y$  中有一个小于  $z$  的表达式为\_\_\_\_\_。

3. 已知 `st='Python Programming'`, 则

`st[3]`                      结果: \_\_\_\_\_

`st[-1]`                    结果: \_\_\_\_\_

`st[2:7]`                  结果: \_\_\_\_\_

`st[2:-1]`                结果: \_\_\_\_\_

`st[2:]`                   结果: \_\_\_\_\_

`st[:7]`                   结果: \_\_\_\_\_

`st[1::3]`                结果: \_\_\_\_\_

`st[4]*4`                结果: \_\_\_\_\_

4. 执行下列操作后, `lst2` 的值是\_\_\_\_\_

```
lst1=[1,,3,5,7]
```

```
lst2=lst1
```

```
lst1[2]=6
```

```
print(lst2)
```

三、程序填空题

1. 输入浮点数  $x$  的值, 计算  $y=\sin(x)+|2x|$ , 输出计算的结果。

```
import _____(1)_____
```

```
x=_____(2)_____(input("输入 x"))
```

```
y=_____(3)_____.sin(x)+abs(2*x)
```

```
print("y=",y)
```

2. 输入一个 6 位的正整数, 将其分离成 3 个 2 位的整数。例如输入: 123456, 分离出 12, 34, 56。

```
n=int(input("输入一个 6 位的正整数 n"))
```

```
n3=n _____(1)_____
```

```
n2=_____(2)_____ % 100
```

```
n1=n // _____(3)_____
```

```
print(n1,n2,n3)
```

四、简答题

1. 简述 Python 标识符的命名规则。
2. 下列符号中，哪些是 Python 合法的变量名？

(1) a123	(2) a12_3	(3) 123_a
(4) a-123	(5) int	(6) XYZ
(7) False	(8) sin(x)	(9) sinx
2. Python 中基本的数据型有哪些？
3. Python 中组合数据类型有哪些？
4. 字符串的定界符有几种？
5. 转义符的作用是什么？
6. 格式化输出有几种形式？

## 第 3 章 基本的控制结构

结构化程序设计的基本思想之一是程序代码编写采用三种基本结构，即顺序结构、选择结构和循环结构组成；它们的特点都是“单入口和单出口”，即每种控制结构可用一个入口和一个出口的流程图表示。据此可编写出结构良好、易于阅读和调试的程序。

### 3.1 顺序结构

顺序结构是最基本和常用的程序结构，它最符合人们的书写习惯。Python 的 IDLE 交互状态下执行的程序实质就是顺序结构。顺序结构是按照语句出现的先后顺序依次执行，如图 1.3.1 所示。程序执行了“语句块 1”后再执行“语句块 2”。这里的语句块可以是简单语句、控制结构语句等。

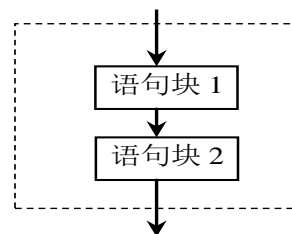


图 1.3.1 顺序结构流程图

#### 3.1.1 引例—人民币与美元兑换

**【例 3.1】**人民币与美元之间兑换，假定汇率为：1 人民币=0.1556 美元，1 美元=6.4250 人民币。要求：输入人民币，转换成美元，显示保留两位小数。

分析：该程序比较简单，通过前两章的学习编程是很容易的，关键要搞清楚语句顺序问题。

人民币兑换成美元的程序代码如下：

```
dollar=CNY*0.1556                                #语句①
CNY=int(input("输入人民币： "))                  #语句②
print("人民币： %d 元 兑换为美元为： %.2f"%(CNY,dollar))
```

程序运行：

```
Traceback (most recent call last):
  File "D:/py/3-1.py", line 1, in <module>
    dollar=CNY*0.1556
NameError: name 'CNY' is not defined
```

错误原因是语句①和②顺序不正确，程序执行到语句①，变量 CNY 没有定义没有值，不能参加运算。改正后程序为：

```
CNY=int(input("输入人民币： "))                  #语句①
dollar=CNY*0.1556                                #语句②
print("人民币： %d 元 兑换为美元为： %.2f"%(CNY,dollar))
```

程序运行：

```
输入人民币：100
人民币：100 元 兑换为美元为：15.56
```

对于美元兑换为人民币，请读者自行完成。

### 3.1.2 简单程序的结构

从上例可以看出，程序的执行顺序是自上而下，依次执行。简单的程序一般由三部分组成：输入数据、对数据进行处理、输出结果，如图 1.3.2 所示：

(1) 在 python 中，输入少量数据一般通过 `input()` 函数，测试程序的批量数据可以通过随机数生成，实际应用批量数据一般通过文件读入。

(2) 进行处理要看解决问题的复杂度。例 3.1 是最简单的计算处理，即将计算表达式的值赋值给变量。

(3) 输出结果通常使用 `print()` 函数。

以上这些语句都介绍过，读者可以编写完整的 Python 程序了。

**【例 3.2】** 输入三角形三条边长，计算面积并显示结果。

分析：已知三条边边长，计算三角形面积利用海伦公式即可求得。相应的程序代码和运行如下：

```
#输入三角形三条边的边长，求三角形的面积
import math                      # 导入 math 模块
a,b,c=eval(input("输入 a,b,c: "))
t=(a+b+c)/2
area=math.sqrt(t*(t-a)*(t-b)*(t-c))  # 调用 math 模块的 sqrt 函数
print("三角形面积=",area)
```

程序运行：

```
输入 a,b,c: 3,4,5
三角形面积= 6.0
输入 a,b,c: 1,2,3
三角形面积= 0.0
```

注意：当程序输入三条边长分别为 1,2,3 时，不能构成三角形，故显示面积为 0。为防止此类问题出现，好的程序应在计算面积前首先判断能否构成三角形。这就要涉及到下一节的选择结构。

## 3.2 选择结构

计算机要处理的问题往往是复杂多变的，仅采用顺序结构是不够的。必须利用选择结构等来解决实际应用中的各种问题。选择结构的特点是在程序执行时，根据不同的“条件”，选

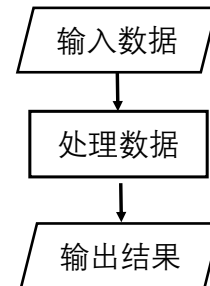


图 1.3.2 程序的基本结构



择执行相应的语句。

Python 中提供的 if 条件语句有多种形式：单分支、双分支和多分支等。

### 3.2.1 引例—体质指数 BMI 的计算

**【例 3.2】**体质指数 BMI(Body Mass Index)的计算公式为：

$$\text{BMI} = \text{体重(kg)} / \text{身高(m)}^2$$

根据 BMI 值可分成 4 类，如图 1.3.3 所示。

编写输入体重和身高，显示所属类型和提示信息。

分类 = {	偏瘦	bmi < 18.5
	正常	18.5 < bmi < 24
	偏胖	24 ≤ bmi < 28
	肥胖	bmi ≥ 28

图 1.3.3 BMI 分类

单分支实现程序代码如下：

```
w,h=eval(input("输入体重(kg)， 身高(m): "))
bmi=w/(h*h)
if 18.5<=bmi<=24:
    print("bmi=%5.2f,正常指数，恭喜你！"%(bmi))
```

程序运行：

```
输入体重(kg)， 身高(m): 61,1.6
bmi=23.83,正常指数，恭喜你！
>>>
```

程序运行：

```
输入体重(kg)， 身高(m): 62,1.6
>>>
```

即计算的 bmi 值不在正常范围内，程序没有任何信息显示，不够完整。

**【例 3.3】**要分别考虑在正常范围内和不在正常范围内两种情况和相应的处理。用双分支结构实现，程序代码如下：

```
w,h=eval(input("输入体重(kg)， 身高(m): "))
bmi=w/(h*h)
if 18.5<=bmi<=24:
    print("bmi=%5.2f,正常指数，恭喜你！"%(bmi))
else:
    print("bmi=%5.2f,不正常指数！"%(bmi))
```

**【例 3.4】**进一步思考，队计算的 BMI 值，希望能给出所属的分类信息。这就要多分支结构实现，程序代码如下：

```
w,h=eval(input("输入体重(kg)， 身高(m): "))
bmi=w/(h*h)
if bmi<18.5:
    print("bmi=%5.2f,低于正常指数范围 18.5~24，偏瘦！"%(bmi))
elif 18.5<=bmi<=24:
```

```

print("bmi=%5.2f,正常指数,恭喜你! "%(bmi))
elif 24<=bmi<=27:
    print("bmi=%5.2f,高于正常指数范围 18.5~24, 偏胖!"%(bmi))
else:
    print("bmi=%5.2f,高于正常指数范围 18.5~24, 肥胖!"%(bmi))

```

通过对 BMI 值的计算和分析，显示了 Python 提供的 if 语句构成的三种选择结构，即单分支、双分支和多分支。

## 3.2.2 选择结构语句

从上例可以看到，选择结构有三种形式。

### 1. 单分支结构—if 语句

语句形式：

if 表达式：

语句块

说明：

① 表达式表示条件，结果有两种情况 True 和 False；

表达式可以是算术、字符串、关系和逻辑表达式，若为算术或字符串表达式，则非零或非空为 True，零或空为 False。

② 语句块为 if 语句的内嵌语句，由一条或多条语句构成，同一语句块内的语句必须在同一列上进行缩进；否则表示语句块的结束。

③ if 语句的作用是当“表达式”值为 True 时执行“语句块”，若为 False，则跳过语句块；然后共同继续执行 if 语句的下一语句。流程图如图 1.3.4 所示。

**注意：**语句中用冒号表示下面是内嵌语句块，语句块由相同缩排的语句组成，空格数量必须相同（默认为 4 个），可以使用 Tab 键只要确保对齐即可。语句块的使用和书写规则在下面选择结构的 elif、else 子句，循环结构的循环体、函数定义中的函数体等都相同，不再重复说明。

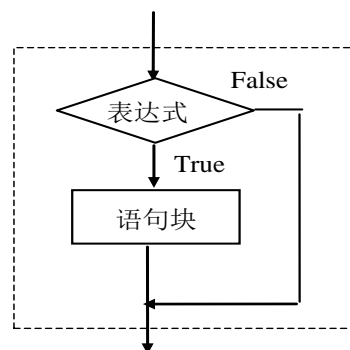


图 1.3.4 单分支结构

### 2. 双分支结构—if...else 语句

语句形式：

if 表达式：

语句块 1

else:

语句块 2

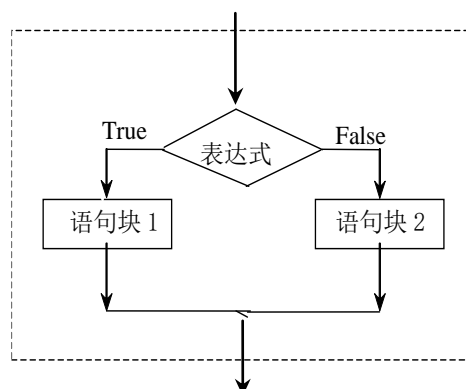


图 1.3.5 双分支 if 语句流程图

该语句的作用是当表达式的值为 **True** 时，执行语句块 1，否则执行 **else** 后面的语句块 2，其流程参见图 1.3.5。

$$y = \begin{cases} \sin x + \sqrt{x^2 + 1} & x \neq 0 \\ \cos x - x^3 + 3x & x = 0 \end{cases}$$

**【例 3.5】**已知分段函数。编程，输入任意  $x$ ，计算对应的  $y$  值。

分析：最直观的使用双分支结构实现，当然也可使用单分支结构实现。

双分支结构实现	单分支结构实现
<pre>from math import * x=float(input("输入 x: ")) if x!=0:     y=sin(x)+sqrt(x*x+1) else:     y=cos(x)-x**3+3*x print("x={ }y={:5.3f}".format(x,y))</pre>	<pre>from math import * x=float(input("输入 x: ")) y=cos(x)-x**3+3*x if x!=0:     y=sin(x)+sqrt(x*x+1) print("x={ }y={:5.3f}".format(x,y))</pre>

程序运行显示：

```
输入 x: 2.3
x=2.3y=3.254
```

思考：若单分支结构实现代码次序改为如下，请问能实现？为什么？

```
if x!=0:
    y=sin(x)+sqrt(x*x+1)
y=cos(x)-x**3+3*x
```

### 3. 多分支结构— **if...elif...else** 语句

双分支结构只能根据条件的 **True** 和 **False** 决定处理两个分支中的其中一个。当实际处理的问题有多种条件时，就要用到多分支结构。

语句形式如下：

```
if 表达式 1:
    语句块 1
elif 表达式 2:
    语句块 2
...
[else:
    语句块 n+1]
```

该语句的作用是根据不同的表达式值确定执行哪个语句块，测试条件的顺序为表达式 1、表达式 2...一旦遇到表达式值为 **True**，则执行该条件下的语句块，然后执行多分支结构的下一语句。其流程参见图 1.3.6。

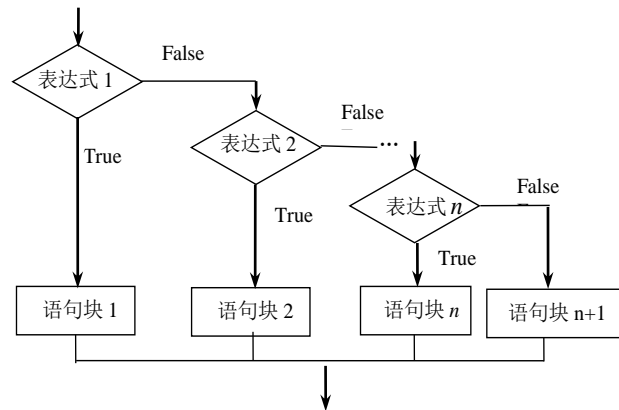


图 1.3.6 多分支结构

**【例 3.6】** 已知字符变量 `ch` 中存放了一个输入的字符，判断该字符是字母字符、数字字符还是其他字符，并作相应的显示，程序代码如下：

```

#输入字符，显示输入的是何种类型字符
ch = input("输入一个字符:")
if 'a'<=ch.lower() <= 'z':          # 大小写字母均考虑
    print(ch + "是字母字符")
elif '0'<= ch <= '9':                # 表示是数字字符
    print(ch + "是数字字符")
else:
    print(ch + "是其他字符")          # 除上述字符以外的字符
  
```

程序运行：

```

输入一个字符:e
e 是字母字符
  
```

注意：

- (1) 不管有几个分支，程序执行了一个分支后，其余分支不再执行。
- (2) 当多分支中有多个表达式同时满足，则只执行第一个与之匹配的语句块。因此，要注意多分支中表达式的书写次序，防止某些值的过滤。

**【例 3.7】** 计算轨道交通票价。某城市轨道交通实行按里程计价的多级票价，起步 6 公里内每人 3 元，6-12 公里每人 4 元，12-32 公里每 10 公里加 1 元，32 公里以上每 20 公里加 1 元，票价不封顶。公里，显示对应票价。程序代码如下：

```

km=eval(input("输入公里: "))
if km<=6:
    s = 3
elif km<=12:
    s = 4
elif km<=32:
    s = 4 + (km-12)/10
else :
    s=6+(km-32)/20          # 6 为 0~32 公里的票价
  
```

```

if int(s)!=s:                                # 考虑票价以元为单位
    s=int(s)+1
print("某城市地铁票价: ",s)

```

### 3.2.3 选择结构的嵌套

选择结构的嵌套是指把一个选择结构放入另一个选择结构之内,也就是选择结构中的语句块也是选择结构。

例如嵌套的形式如下:

<p>双分支均内嵌单分支</p> <pre> if 表达式 1:     if 表达式 11:         语句块 11 else:     if 表达式 21:         语句块 21 </pre>	<p>双分支的 if 块内嵌双分支</p> <pre> if 表达式 1:     if 表达式 11:         语句块 11     else:         语句块 12 else:     语句块 2 </pre>
---	---

使用嵌套的 if 结构时,一定要严格控制好不同层次语句块的缩进量,因为这决定了不同语句块的从属关系以及要处理的问题是否被正确地实现、是否能够被 Python 正确地理解和执行。

**【例 3.8】** 已知  $x$ 、 $y$ 、 $z$  三个数,比较它们的大小并进行排序,使得  $x>y>z$ 。

分析:在计算机中无法直接同时对三个数比较,只能依次通过多次两两相比较来实现。

下面程序是通过一个 if 语句和一个嵌套 if 语句来实现,流程图如图 1.3.7 所示。

程序如下:

```

x,y,z=eval(input("输入 x, y, z: "))
if x<y:
    x,y=y,x      # x<y, 交换, 使得 x>y
if y<z:
    y,z=z,y      # y<z, 交换, 此时 z 最小
    if x<y:      # 但还要考虑 x 与 y 大小
        x,y=y,x  # x<y, 交换, 使得 x>y
print(x,">",y,">",z)

```

程序运行:

```

输入 x, y, z: 3,45,9
45 > 9 > 3
>>>

```

思考:

(1) 若不用嵌套 if 语句,而用三个 if 语句,程序如何编写?

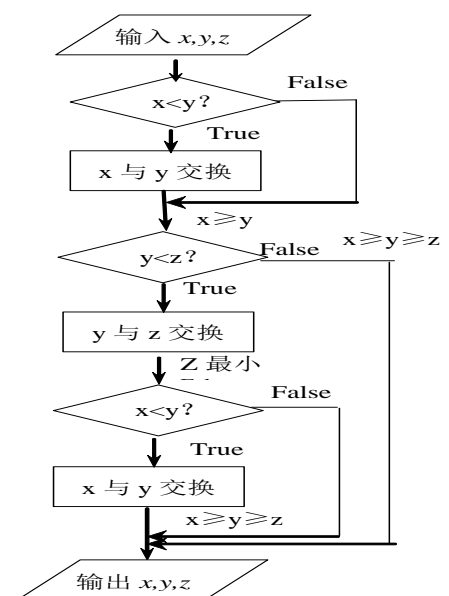


图 1.3.7 三个数排序流程图

(2) 系统提供了 `max()`、`min()` 函数，提供此两个函数是否不用 `if` 语句就可对 3 个数排序？

(3) 若要对四个数进行排序，程序又如何编写？若要对大量数据进行排序呢？

## 3.3 循环结构

计算机最擅长的功能之一就是按规定的条件，重复执行某些操作。例如，按照人口增长率统计人口数；根据各课程的学分、绩点和学生的成绩，统计每个学生的平均绩点等。这类问题都可通过循环结构来方便地实现。

### 3.3.1 引例—批量数据的处理

在例 3.4 中实现对 1 个人的 BMI 体质指数的计算和显示相应信息。若要对一个部门若干人的 BMI 体质指数计算，通常由以下两种方法。

【例 3.9】要统计分析一个部门 100 个职工的 BMI 体质指数，如何实现？则只要在计算一个人的基础上增加一个 `for` 语句来实现，程序如下：

```
for i in range(1,101):
    w,h=eval(input("输入第"+str(i)+"个人的体重，身高： "))
    bmi=w/(h*h)
    if 18.5<=bmi<=24:
        print("正常")
    elif bmi<18.5:
        print("偏瘦")
    else:
        print("偏胖")
```

程序运行：

```
输入第 1 个人的体重，身高： 62,1.62
正常
输入第 2 个人的体重，身高： 80,1.78
偏胖
输入第 3 个人的体重，身高： 78,1.87
正常
输入第 4 个人的体重，身高： 50,1.6
正常
输入第 5 个人的体重，身高： 50,1.7
偏瘦
输入第 6 个人的体重，身高：
.....
>>>
```

【例 3.10】若要处理若干个职工的信息，当输入体重或身高为 0 时结束，如何实现？则利用 `while` 语句来实现，程序如下：

```

i=1
w,h=eval(input("输入第"+str(i)+"个人的体重，身高："))
while w!=0 and h!=0:                                # 输入的体重和身高均不为 0，进行计算和显示
    bmi=w/(h*h)
    if 18.5<=bmi<=24:
        print("正常")
    elif bmi<18.5:
        print("偏瘦")
    else:
        print("偏胖")
    i=i+1                                              # 一个人处理完成
    w,h=eval(input("输入第"+str(i)+"个人的体重，身高："))    # 继续输入下一个人信息

```

程序运行：

```

输入第 1 个人的体重，身高： 61,1.7
正常
输入第 2 个人的体重，身高： 51,1.7
偏瘦
输入第 3 个人的体重，身高： 0,1.7
>>>

```

从引例中看到，Python 中提供了两种基本的循环结构：for 循环和 while 循环。for 循环常用于预知循环次数的场合；while 循环常用于未知循环次数场合。

### 3.3.2 for 语句

#### 1.for 语句

在一般程序设计语言中，for 语句常称为计数型循环语句；而在 Python 语言中，for 语句实质是一个通用的序列迭代器，可以遍历任何序列中的元素。for 语句形式如下：

```

for 变量 in 序列:
    语句块

```

该语句的作用是用序列中的元素逐一赋值给变量，对每一次的成功赋值都执行一次语句块（循环体）。当序列都遍历，即每一个元素都用过了，则循环结束，接着执行 for 语句的下一条语句。

注意：

（1）序列可以是：

常用的 range() 函数；组合数据类型，包括字符串、列表、元组、集合和字典等。

（2）循环的次数由序列中的成员个数决定。

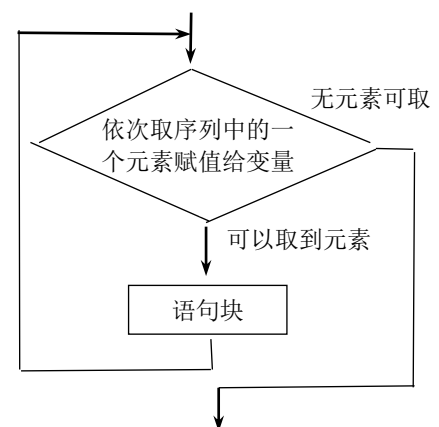


图 3-3-2 for 语句处理流程

(3) 书写规则序列后要有冒号、循环体要右缩排。

执行的流程图如图 1.3.8 所示。

**【例 3.11】** 循环控制的序列是 range()函数。将可打印的字符和其 ASCII 码制成表格输出，每行打印 7 个字符。

分析：

①ASCII 码中，空格到“~”是可以打印的字符，其余为不可打印的控制字符。可打印的字符的编码值为 32~126，利用 chr()函数获得编码值对应的字符。

②每行显示 7 个字符项，通过设置一个计数器变量 n，每打印一项加 1；当满了 7 项换行。

代码如下，运行结果如图 1.3.9 所示：

程序如下：

```
n=0
for asc in range(32,127):
    print('%s=%d'%(chr(asc),asc),end='\t')
    n=n+1
    if n % 7==0:print()
```

程序运行：

=32	!=33	"=34	#=35	\$=36	%=37	@=38
,=39	(=40	)=41	*=42	+ =43	, =44	-=45
. =46	/=47	0=48	1=49	2=50	3=51	4=52
5=53	6=54	7=55	8=56	9=57	: =58	;=59
<=60	=61	>=62	?=63	@=64	A=65	B=66
C=67	D=68	E=69	F=70	G=71	H=72	I=73
J=74	K=75	L=76	M=77	N=78	O=79	P=80
Q=81	R=82	S=83	T=84	U=85	V=86	W=87
X=88	Y=89	Z=90	[=91	\=92	] =93	^=94
_ =95	`=96	a=97	b=98	c=99	d=100	e=101
f=102	g=103	h=104	i=105	j=106	k=107	l=108
m=109	n=110	o=111	p=112	q=113	r=114	s=115
t=116	u=117	v=118	w=119	x=120	y=121	z=122
{=123	=124	}=125	~=126			

图 1.3.9 ASCII 码对照表

**【例 3.12】** 循环控制的序列是字符串。输入一字符串，检验其输入的是否全部是数字字符。

分析：从字符串中逐一取出字符，判断其若是非数字字符则该字符串是非数字字符串。取字符有两种方法：遍历法或索引法。

方法一、遍历法	方法二、索引法
<pre>str=input("输入字符串:") isnumeric=True for c in str:     if c not in "0123456789":         isnumeric=False if isnumeric:     print(str,"是数字字符串") else:     print(str,"含有非数字字符")</pre>	<pre>str=input("输入字符串:") isnumeric=True for i in range(len(str)):    #len()字符串长度     if str[i] not in "0123456789":         isnumeric=False if isnumeric:     print(str,"是数字字符串") else:</pre>



```
print(str,"含有非数字字符")
```

程序运行：

```
输入字符串:123e567
123e567 含有非数字字符
>>>
输入字符串:1234567
1234567 是数字字符串
>>>
```

**【例 3.13】**循环控制的序列是列表。随机生成 10 个 1~100 之间的整数，求最小、最大和平均值。

分析：利用 `sample()` 函数产生指定范围内、指定个数的随机数；然后通过遍历列表，逐一比较、累加，求得最小、最大和平均值。

```
from random import *
lst=sample(range(1,101),10)    #sample()生成 10 个在 1，100 之间的随机数
print(lst)
min1=100; max1=0
sum=0
for i in lst:
    if i>max1: max1=i
    if i<min1: min1=i
    sum+=i
print("max=",max1," min=",min1," avg=",sum/10)
```

程序运行显示：

```
[73, 59, 51, 36, 41, 81, 45, 97, 100, 39]
max= 100  min= 36  avg= 62.2
>>>
```

思考：为什么设置初值 `min1=100`；`max1=0`，而不是初值：`min1=0`；`max1=100`？

## 2.range 函数

`range()` 函数是 Python 的内置函数，用于创建一个整数序列表，一般用于 `for` 语句中，实现如同其他语言 `for` 语句的循环控制的功能。`range()` 函数有如下三种形式：

### (1) `range(start,end,step)`

整数序列表以起始值为 `start`、步长为 `step`、结束值为 `end`，但不包括 `end`。例：

```
>> list(range(1,10,3))    # 产生从 1 开始、以步长为 3、不到 10 的数字序列。
[1,4,7]
```

### (2) `range(start,end)` #缺省 `step` 为 1

```
>> list(range(1,10))    # 产生从 1 开始、以步长为 1、不到 10 的数字序列，例：
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

### (3) `range(end)` #缺省 `start` 为 0，缺省 `step` 为 1，例：

```
>>list(range(10))    # 产生从 0 开始到 9 的数字序列
```

### 3.3.3 while 语句

while 语句常用于控制循环次数未知的循环结构，语句形式为：

while 表达式:  
    语句块

语句作用：判断表达式的值，若为 **True**，则执行语句块(循环体)；否则退出循环，执行 while 语句的下一句语句。语句执行的流程如图 1.3.10 所示：

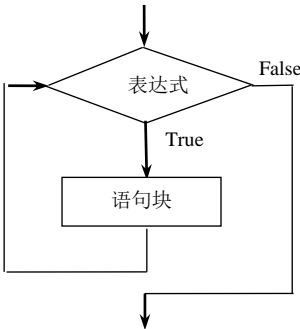


图 1.3.10 while 语句流程图

**【例 3.14】**据统计 2018 年年末我国总人口 13.95 亿人，自然增长率 3.81‰。若按此增长率，多少年后我国人口翻倍？

分析：解此问题两种方法：

（1）直接调用对数函数求得。

可根据公式： $27.9=13.95(1+0.00381)^n$ ，得： $n=\log(2)/\log(1.00381)$

调用内置数学函数对数可求得，但求得的年数不为整数，也得不到实际的人数。

（2）利用循环求得。

根据增长率求得每年的人数，当人口数没有翻倍重复计算。

程序代码和运行效果如下：

直接调用对数函数求得	利用循环结构求得
<pre>import math n=math.log(2)/math.log(1.00381) if int(n)!=n:     n=int(n)+1 #n 若非整数，取整加 1 年 print("用对数求得的年数为： %d"%(n))</pre>	<pre>x=13.95 n=0 while x&lt;27.9:     n+=1     x=x*(1+0.00381) print("循环求得年数为： %d 人数为： %f 亿"%(n,x))</pre>
程序运行： 用对数求得的年数为： 183	程序运行： 循环求得的年数为： 183 人数为： 27.977053 亿

**【例 3.15】**求两自然数 m 和 n 的最大公约数。

求最大公约数在数学上有多种方法，最方便的是短除法。计算机解决此问题也有多种方法。

(1) 枚举法。该方法求最大公约数比较简单且容易理解，可通过递增和递减方式实现。

递减算法思路

① 比较两个数  $m, n$ ，将小数放入变量  $\min\_mn$  中；

② 判断  $\min\_mn$  是否该两数的公因子，是则  $\min\_mn$  为两数的最大公约数，结束；否则  $\min\_mn$  递减重复判断是否是两数的公因子。程序如下：

```
m,n=eval(input("输入 m,n"))
min_mn = min(m,n)
while not(m % min_mn == 0 and n % min_mn == 0):      # gcd 不是 m、n 的公因子则循环
    min_mn = min_mn - 1
print("最大公约数是: ",min_mn)
```

程序运行：

```
输入 m,n 24,10
最大公约数是: 2
>>>
```

思考：递增法思想是从 1 开始依次递增求两个数的最大公因子，代码如何实现？

(2) 辗转相除法或相减法。

辗转相除法即欧几里德算法，特点效率高，算法思想是：

① 对于已知两数  $m, n$ ，使得  $m > n$ ；

②  $m$  除以  $n$  得余数  $r$ ；

③ 若  $r=0$  则  $n$  为求得的最大公约数，算法结束；若  $r \neq 0$ ，则令  $m \leftarrow n$ ， $n \leftarrow r$ ，重复执行步骤②。

流程图参见图 1.3.11(a)，程序如下：

```
m,n=eval(input("输入 m,n"))
if m<n:                                # 若 m<n，则交换，使得 m>=n
    m,n=n,m
r=m % n
while r!=0:
    m=n
    n=r
    r=m % n
print("最大公约数是: ",n)
```

程序运行：

```
输入 m,n 24,10
最大公约数是: 2
```

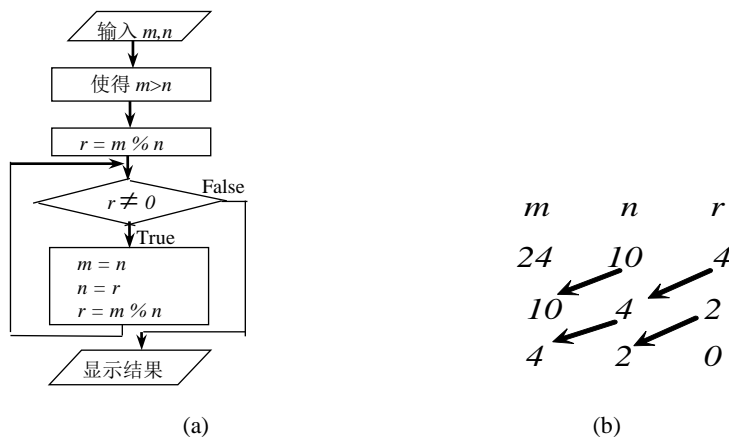


图 1.3.11 辗转相除法求最大公约数流程图和示例

当  $m, n$  输入的值分别为 24, 10 时, 显示最大公约数为 2, 如图 1.3.11(b)所示。

注意: python 的 `math` 库中提供求最大公约数 `gcd()` 函数, 调用并加以验证自编的求最大公约数程序。

### 3.3.4 循环结构的嵌套

在一个循环体内又包含了完整的循环结构称为循环结构的嵌套。循环结构的嵌套对 `for` 语句和 `while` 语句均适用。

**【例 3.16】** 打印九九乘法表(运行界面参见图 1.3.12)。

分析: 打印九九乘法表, 利用两重循环来方便地实现。其中: 外循环控制变量  $i$  作为被乘数, 也决定了乘法表的行数; 内循环控制变量  $j$  作为乘数, 也决定了乘法表的列数; 同时还要考虑输出的形式, 即一行输出后要换行。程序如下:

```

for i in range(1,10):
    for j in range(1,10):
        print("{}*{}={:2d}".format(i,j,i*j),end=" ")
    print()
  
```

运行结果:

```

1×1= 1 1×2= 2 1×3= 3 1×4= 4 1×5= 5 1×6= 6 1×7= 7 1×8= 8 1×9= 9
2×1= 2 2×2= 4 2×3= 6 2×4= 8 2×5=10 2×6=12 2×7=14 2×8=16 2×9=18
3×1= 3 3×2= 6 3×3= 9 3×4=12 3×5=15 3×6=18 3×7=21 3×8=24 3×9=27
4×1= 4 4×2= 8 4×3=12 4×4=16 4×5=20 4×6=24 4×7=28 4×8=32 4×9=36
5×1= 5 5×2=10 5×3=15 5×4=20 5×5=25 5×6=30 5×7=35 5×8=40 5×9=45
6×1= 6 6×2=12 6×3=18 6×4=24 6×5=30 6×6=36 6×7=42 6×8=48 6×9=54
7×1= 7 7×2=14 7×3=21 7×4=28 7×5=35 7×6=42 7×7=49 7×8=56 7×9=63
8×1= 8 8×2=16 8×3=24 8×4=32 8×5=40 8×6=48 8×7=56 8×8=64 8×9=72
9×1= 9 9×2=18 9×3=27 9×4=36 9×5=45 9×6=54 9×7=63 9×8=72 9×9=81
  
```

图 1.3.12 九九乘法表运行界面

**思考:** 若要分别打印呈下三角和上三角的九九乘法表, 如图 1.3.13 和 3.14 的结果, 程序如何作微小的改动?

```

1X1= 1
2X1= 2 2X2= 4
3X1= 3 3X2= 6 3X3= 9
4X1= 4 4X2= 8 4X3=12 4X4=16
5X1= 5 5X2=10 5X3=15 5X4=20 5X5=25
6X1= 6 6X2=12 6X3=18 6X4=24 6X5=30 6X6=36
7X1= 7 7X2=14 7X3=21 7X4=28 7X5=35 7X6=42 7X7=49
8X1= 8 8X2=16 8X3=24 8X4=32 8X5=40 8X6=48 8X7=56 8X8=64
9X1= 9 9X2=18 9X3=27 9X4=36 9X5=45 9X6=54 9X7=63 9X8=72 9X9=81

```

图 1.3.13 呈下三角的九九乘法表

```

1X1= 1 1X2= 2 1X3= 3 1X4= 4 1X5= 5 1X6= 6 1X7= 7 1X8= 8 1X9= 9
2X2= 4 2X3= 6 2X4= 8 2X5=10 2X6=12 2X7=14 2X8=16 2X9=18
3X3= 9 3X4=12 3X5=15 3X6=18 3X7=21 3X8=24 3X9=27
4X4=16 4X5=20 4X6=24 4X7=28 4X8=32 4X9=36
5X5=25 5X6=30 5X7=35 5X8=40 5X9=45
6X6=36 6X7=42 6X8=48 6X9=54
7X7=49 7X8=56 7X9=63
8X8=64 8X9=72
9X9=81

```

图 1.3.14 呈上三角的九九乘法表

提示：外循环行数不变，内循环改变循环的初值或终值、输出格式控制就可。

多重循环的循环次数等于每一重循环次数的乘积。对于多重循环的每一个循环变量变化规律的理解，读者只要以手表上时、分、秒三根针构成的三重循环的变化模拟，即当内循环秒针走满一圈时，分针加一，秒针又从头开始走；当分针走满一圈时，时针加一，分针、秒针从头开始；以此类推，时针走满一圈，即 12 小时，循环结束；整个循环执行的次数为：12\*60\*60。

对于循环语句使用，要注意以下事项：

- ①内循环变量与外循环变量不能同名。
- ②外循环必须完全包含内循环，不能交叉。
- ③若循环体内有 if 语句，或 if 语句内有循环语句，也不能交叉。

### 3.3.5 循环结构的其他语句

循环结构的其他语句表示在循环体内可以出现的语句，分为跳转类语句和 else 子句。

#### 1. 跳转语句

在 Python 循环体结构中，可以使用跳转语句来改变循环执行的路径，分别是 break 语句、continue 语句和 pass 语句。

##### (1) break 语句

break 语句的作用是结束循环，执行循环语句后的下一个语句。break 语句有时又称为循环的断路语句。

##### (2) continue 语句

continue 语句的作用是跳过本次循环体内 continue 后面的语句，并开始下一次循环。continue 语句有时又称为循环的短路语句，是指只对本次循环短路，并不终止循环。

两者语句出现在循环体内，常和 if 语句一起使用实现有条件地跳转，两者区别见流程图 1.3.15(a)和(b)所示。

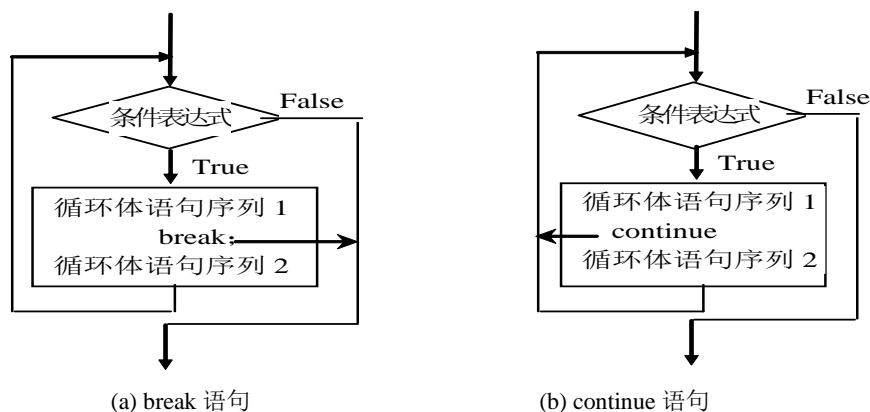


图 1.3.15 循环结构中两种语句区别图示

**【例 3.17】**表 3.1 的两个程序段，显示了 break 和 continue 语句的区别，读者可自行分析。

表 3.1 break 和 continue 语句的区别

循环体内用 break 语句	循环体内用 continue 语句
<pre>for i in range(20,0,-1):     if i % 6 == 0:         break     print(i,end=" ")</pre>	<pre>for i in range(20,0,-1):     if i % 6 == 0:         continue     print(i,end=" ")</pre>
运行结果： 20 19	运行结果： 20 19 17 16 15 14 13 11 10 9 8 7 5 4 3 2 1
功能：当找到能够被 6 整除的最大的数时不再循环。	功能：显示 20 到 1 之间非 6 的倍数的数

### (3) pass 语句

pass 语句是不执行任何操作，实质是空语句；作用就是占据位置，当某处需要有语句但又不产生任何操作时可使用 pass 语句。

**【例 3.18】**模拟打字机效果。

分析：可利用空循环起到延迟作用，实现模拟打字机效果。程序如下：

```
for c in "welcome":
    print(c,end=" ")
    for j in range(1,10000000): # 延迟作用，起到打字机效果
        pass
```

## 2. else 子句

在其他语言中，else 子句主要用在 if 选择结构中；而在 Python 中，else 子句可以出现在循环结构中，也可以出现在异常处理结构中。

在循环结构中出现 else 子句，表示循环正常结束后执行 else 后的语句块。语句形式：

```
for 循环变量 in 序列:
    循环体
else:
    语句块

while 表达式:
    循环体
else:
    语句块
```

else:  
语句块

【例 3.19】统计列表中数据的均值。程序如下：

```
sum=0
n=0
for i in [66,88,98,76,89,55,67,54]:
    sum+=i
    n+=1
    print(i,end=" ")
else:
    print("\n 共有%d 个数，平均值为%.2f"%(n,sum/n))
```

程序运行：

```
66 88 98 76 89 55 67 54
共有 8 个数，平均值为 74.12
>>>
```

此例看似没有 else 子句也可有相同效果，但当在循环体中有 break 语句时，else 子句的作用就显现，尤其在循环的嵌套的场合使得代码更精炼。

【例 3.20】求 2~100 之间的素数，并且每行显示 8 个素数。

分析：素数就是除 1 和本身以外，不能被其他任何正整数整除的数。

(1) 根据此定义，要判别某数 m 是否为素数最简单的方法就是依次用 j=2~ m-1 去除，只要有一个能整除，m 不是素数，退出循环；若都不能整除，则 m 是素数。

(2) 对 2~100 之间的素数，只要增加一个外循环，m=2,3,..., 100 就可。

程序如下：

方法一、一般程序设计语言处理	方法二、用 else 子句
<pre>countm=0 for m in range(2,101):     tag=True      # 假定m为素数     for j in range(2,m):         if m % j ==0:             tag=False #m能被j 整除不是素数             break     # 退出本次循环，处理下一个m     if tag:        # 若tag为True 则m是素数显示         print(m,end="\t")         countm+=1         if countm % 8 ==0:             print()</pre>	<pre>countm=0 for m in range(2,101):     for j in range(2,m):         if m % j ==0:             break     else:         print(m,end="\t")         countm+=1         if countm % 8 ==0:             print()</pre>

程序运行：

```
2      3      5      7      11     13     17     19
23     29     31     37     41     43     47     53
59     61     67     71     73     79     83     89
97
>>>
```

从上例求素数的两种方法对比可知，方法二在循环结构中使用 `else` 子句可使代码更精炼、算法更利于理解。

思考：将上例方法一中的 `break` 语句改为 `continue` 语句，运行结果变化吗？

## 3.4 综合应用

本章介绍了构成结构化程序设计的三种基本结构：顺序结构、选择语句、循环语句，它们是程序设计的基础，对今后编程非常重要，希望读者熟练掌握。

对初学者来说，从本章开始，编程工作量明显增多，要调试通过一个程序有时要花很多时间。经验告诉我们，学习程序设计没有捷径可走，只有多看多练、通过上机调试，发现问题，解决问题，才能真正理解、掌握所学的知识。

本节介绍相关的应用和常用算法。

### 1 数字之美——数字图

**【例 3.21】** 利用循环结构显示有规律的数字字符图，如图 1.3.16 所示。

```

1×8+1=9
12×8+2=98
123×8+3=987
1234×8+4=9876
12345×8+5=98765
123456×8+6=987654
1234567×8+7=9876543
12345678×8+8=98765432
123456789×8+9=987654321

```

图 1.3.16 有规律数字图

```

1
12
123
1234
12345
123456
1234567
12345678
123456789

```

图 1.3.17 有规律数值

分析：

(1) 如何构建有规律的数值（如图 1.3.17 所示）以及如何控制输出格式。有规律数值生成通项： $t=t*10+i$   $1 \leq i \leq 9$ ；输出格式控制，左边的空格数与行数有关： $(11-i)*"$  "，效果。

(2) 整个数字图显示就根据可变数值和不变字符逐项输出就可。

程序如下：

```

t=0
for i in range(1,10):
    t=t*10+i
    print("{}×8+{}={}".format((11-i)*" ",t,i,t*8+i))

```

思考：若要输出如图 1.3.18 所示的数字图，如何实现？

```

1×1=1
11×11=121
111×111=12321
1111×1111=1234321
11111×11111=123454321
111111×111111=12345654321
1111111×1111111=1234567654321
11111111×11111111=123456787654321
111111111×111111111=12345678987654321

9×9=81
98×98=8881
987×987=88881
9876×9876=888881
98765×98765=8888881
987654×987654=88888881
9876543×9876543=888888881
98765432×98765432=8888888881
987654321×987654321=88888888881

```

图 1.3.18 有规律数字图例

进一步思考：从上例可以理解如何将各位数组成一个有规律的数值，这需要利用循环和



循环体的通项表达式： $t=t*10+i$ 。当  $i$  为循环变量时就构成如图 1.3.17 或图 1.3.18 右边的有规律数值；当  $i$  为常量时，就构成图 1.3.18 左边的有规律数值。同样如何将一个数值逐一分离，例如：1234 分离输出为 4,3,2,1 或者构成反序的数值，算法如下：

```
x=int(input("输入 x: "))
fx=0
while x>0:
    d=x % 10          # 从右往左开始分离每一位
    x=x//10
    print(d,end=" ")
    fx=fx*10+d        # 将每一位连接成一个反序数
print("\n 反序数为: {}".format(fx))
```

程序运行显示：

```
输入 x: 1234
4 3 2 1
反序数为: 4321
>>>
```

## 2. 部分级数和——求圆周率 $\pi$

求部分级数和是用计算机求解初等数学问题近似解的常用方法，如求  $\pi$ 、 $e$ 、 $\sin x$  等。解决的方法关键是根据部分级数和展开式找规律写通项。

**【例 3.22】** 利用格里高利公式求圆周率  $\pi$ ，当通项的绝对值小于  $10^{-6}$  时停止计算，公式为：

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

分析：本例涉及到程序设计中重要的运算求累加问题，通项是  $t=1/n$ ，并且有正负号切换；结束条件  $\left|\frac{1}{n}\right| < 10^{-6}$ 。

程序如下：

```
p=0
n=t=sign=1
while abs(t)>1e-6:
    p=p+t
    n=n+2
    sign=-sign
    t=sign/n
p=p*4
print("π={0:8.6f}".format(p))
```

运行运行显示：

```
π =3.141591
```

## 3. 枚举法——计算机排考试

枚举法亦称穷举法或试凑法。它的基本思想是利用计算机具有高速运算的特点将可能出现的各种情况一一罗列测试，直到所有情况验证完。若某个情况符合题目的条件，则为本题的一个答案；若全部情况验证完后均不符合题目的条件，则问题无解。枚举法是一种比较耗

时的算法，利用了计算机快速运算的特点可方便地求解。枚举的思想可解决许多问题。

**【例 3.23】**计算机来判断肇事车辆。交通事故中肇事车辆撞人后逃逸，警方在现场找到 3 位目击证人询问肇事车辆 5 位数的车牌号码。甲说只看清最左两位为 27；乙说只看清最后位是 3；丙说牌号是 67 的倍数。

分析：利用枚举法求解。

- (1) 确定需要枚举的数字有几位，车牌号是：27XX3，枚举范围 X 均为是 0~9；
- (2) 枚举的几位数字间的关系，即：number=27003+i3\*100+i2\*10
- (3) 验证是否是问题的解？，即：if number % 67==0

程序代码如下：

```
for i3 in range(0,10):
    for i2 in range(0,10):
        number =27003+i3*100+i2*10
        if number % 67==0:
            print("肇事车辆号码为：", number)
```

程序运行显示：肇事车辆号码为： 27403

从上例可看到，枚举法能有效解决问题的关键在于三点：

- (1) 确定枚举的范围，尽量避免扩大枚举范围；
- (2) 确定满足的条件，把所有可能的条件一一罗列；
- (3) 枚举解决问题效率不高，因此，为提高效率，根据解决问题的情况，尽量减少嵌套循环层数或每层循环次数。

思考：古代百元买百鸡问题是典型的枚举法来求解的问题。假定小鸡每只 0.5 元，公鸡每只 2 元，母鸡每只 3 元。现在有 100 元钱要求买 100 只鸡，即：

$$0.5x+2y+3z=100$$

$$x+y+z=100$$

三个未知数两个方程，有若干个解，可用三重或两重循环列出所有可能的购鸡方案。

#### 4.迭代法——求高次方程的根

迭代法又称为递推法，其基本思想是把一个复杂的计算过程转化为简单过程的多次重复。每次重复都在旧值的基础上递推出新值，并由新值代替旧值。

**【例 3.24】**用迭代法求  $x = \sqrt[3]{a}$ ， $a=1\sim 10$ 。求立方根的迭代公式为：

$$x_{i+1} = \frac{2}{3}x_i + \frac{a}{3x_i^2}$$

分析： $x_0$  的初值是任意的，迭代到  $|x_{i+1}-x_i| < \varepsilon = 10^{-5}$  为止；算法流程图如图 1.3.19 所示。方法是给定一个初值，利用迭代公式求得新值，比较新值与初值的差，若小于所要求的精度，则新值为求得的根；否则用新值替代初值，再重复利用迭代公式求得新值。

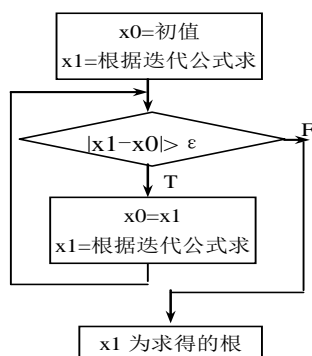


图 1.3.19 算法流程图

程序如下：

```

for a in range(1,11):
    x0=3
    x1=2/3*x0+a/(3*x0*x0)
    while abs(x1-x0)>0.00001:
        x0=x1
        x1=2/3*x0+a/(3*x0*x0)
    print(a,"立方根为",round(x1,4))    # round(x1,4) 保留 4 位小数
  
```

程序运行显示如图 1.3.20 所示。

```

1 立方根为 1.0
2 立方根为 1.2599
3 立方根为 1.4422
4 立方根为 1.5874
5 立方根为 1.71
6 立方根为 1.8171
7 立方根为 1.9129
8 立方根为 2.0
9 立方根为 2.0801
10 立方根为 2.1544
  
```

图 1.3.20 运行结果

## 习题

### 一、选择题

- Python 提供了结构化程序设计的三种基本结构，这三种基本结构是\_\_\_\_\_。
  - 递归结构、选择结构、循环结构
  - 选择结构、过程结构、顺序结构
  - 过程结构、输入输出结构、转向结构
  - 选择结构、循环结构、顺序结构
- 结构化程序设计的基本思想之一是“单入口和单出口”的控制结构，不符合此思想的语句是\_\_\_\_\_。

A.if 语句

B.for 语句

C.while 语句

D.break 语句

3.用 if 语句表示分段函数  $f(x) = \begin{cases} \sqrt{x+1}, & x \geq 1 \\ x^2 + 3, & x < 1 \end{cases}$ , 不正确的是\_\_\_\_\_。

A.f=x\*x+3

B. if x >=1 :

if x >=1:

f=math.sqrt(x+1)

f= math.sqrt (x+1)

if x <1:

f=x\*x+3

C.if x >=1 :

D.if x < 1 :

f= math.sqrt (x+1)

f=x\*x+3

f=x\*x+3

else:

f= math.sqrt(x+1)

4.下面程序段求两个数中的大数, \_\_\_\_\_不正确。

A.if x > y : max = x

B.if x > y : max = x

if y >= x : max=y

else : max = y

C.max = x

D.if y >= x : max = y

if y >= x : max = y

max = x

5.下面 if 语句统计满足性别为男、职称为副教授及以上、年龄小于 40 岁条件的人数, 不正确的语句是\_\_\_\_\_。

A.if sex == "男" and age < 40 and (duty== "教授" or duty== "副教授"): n=n+1

B.if sex == "男" and age < 40 and (duty== "教授" and duty== "副教授"): n=n+1

C.if sex == "男" and age < 40 and "教授" in duty: n=n+1

D.if sex == "男" and age < 40 and duty.index("教授")>=0: n=n+1

6.已知循环语句块:

```
for i in range(1,10):
```

```
    pass
```

```
print(i)
```

循环体的语句块即 pass 语句执行的次数和循环结束后 i 显示的值分别是\_\_\_\_\_。

A.9、 10

B.10、 11

C.9、 9

D.10、 10

7 已知循环语句块:

```
i=1
```

```
while(i<10):
```

```
    pass
```

```
    i=i+1
```



```
else:
```

```
    y=y-x
```

```
print(x,z/x)
```

### 三、程序填空题

1. 当 C 字符串变量中第三个字符是"C"时, 显示"Yes", 否则显示"No".

```
if ____ (1) ____ :
```

```
    print("Yes")
```

```
else:
```

```
    print("No")
```

2. 输入当前年份, 判断它是否为闰年, 并显示相应信息。判断闰年的条件是: 年份能被 4 整除但不能被 100 整除, 或者能被 400 整除。

```
d=int(input("输入年份"))
```

```
if ____ (1) ____ or d % 400 == 0 :
```

```
    print(d,"是闰年")
```

```
else:
```

```
    print(d,"是平年")
```

- 3.利用 if 语句计算分段函数

$$y = \begin{cases} x^2 + 3x + 2 & \text{当 } x > 20 \\ \sqrt{3x} - 2 & \text{当 } 10 \leq x \leq 20 \\ \frac{1}{x} + |x| & \text{当 } x < 10 \end{cases}$$

```
import math
```

```
x = int(input("输入 x 的值: "))
```

```
if x>20:
```

```
    y = x * x + 3 * x + 2
```

```
elif ____ (1) ____:
```

```
    y = 1 / x +abs(x)
```

```
else:
```

```
    ____ (2) ____
```

```
print("y=", y)
```

- 4.下面的程序段是检查输入的算术表达式中圆括号是否配对, 并显示相应的结果。

```
s=input("输入表达式,直到输入=: 表示结束>")
```

```
count=0
```

```
for c in s:
```

```

if c=="(":
    count = count + 1
elif c==")":
    (1)
if (2):
    print("左右括号配对")
elif (3) :
    print("左括号多于右括号",count,"个")
else:
    print("右括号多于左括号",-count,"个")

```

#### 四、问答题

1.有哪几种基本控制结构？

2. 如下 if 语句：

```
if 表达式: 语句块
```

请问其中表达式可以是哪些类型的表达式？

3. 如下 for 语句：

```
for 变量 in 序列:
```

请问序列可以是哪些数据类型？

4. 如下语句：

```
while True: print("Ok")
```

请问输出了多少个“Ok”？

5. 简述 continue 与 break 的区别。

6. 简述 for 循环与 while 循环的相同与不同。

## 第四章 函数

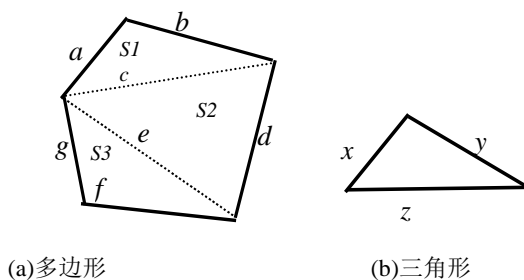
函数是完成某个特定任务的程序语句组合，程序设计中引入函数主要有两个作用：一是实现程序的模块化，便于管理和阅读；二是实现代码复用，提高程序开发的效率。

函数包括两种，一种是开发环境提供的函数，如 Python 提供的 `abs`、`max`、`print`，以及一些类库中提供的函数，如 `math` 库中的 `sqrt`、`sin` 等，这些函数用户直接调用即可。还有一种是用户根据问题而自定义的函数。本章就用户自定义函数的语法及调用展开讨论，希望通过本章的学习，读者可以通过函数的定义提高程序的模块化编写能力。

### 4.1 函数定义与调用

#### 4.1.1 引例-多边形面积的计算

【例 4.1】已知五边形的各条边和对角线的长度，计算其面积，见图 1.4.1(a)。



(a)多边形

(b)三角形

图 1.4.1 多边形面积的计算

计算多边形面积，可将多边形分解成若干个三角形。对图 1.4.1(b)所示的三角形，已知三条边边长为  $x$ 、 $y$ 、 $z$ ，其面积的计算公式如下：

$$\text{area} = \sqrt{c(c-x)(c-y)(c-z)}, \quad \text{其中 } c = \frac{1}{2}(x+y+z)$$

其中： $c$  为三角形周长的一半。

本例中求三个三角形的面积，使用的公式相同，程序的代码相同，仅计算数据不同。如果将这组代码定义为一个函数，调用它三次就可完成计算任务。

```
import math
def triangle(x,y,z):
    c=(x+y+z)/2
    area=math.sqrt(c*(c-x)*(c-y)*(c-z))
    return area

s=input("请输入输入 a、b、c、d、e、f、g 边长,以逗号间隔")
a,b,c,d,e,f,g=eval(s)
```



```

tri1=triangle(a,b,c)
tri2=triangle(c,e,d)
tri3=triangle(e,f,g)
area=tri1+tri2+tri3
print(area)

```

上面程序中，定义了计算三角形面积的函数 `triangle`，它是用户自定义的函数，通过三次调用 `triangle` 完成多边形面积的计算，使得程序的代码更加简洁，容易阅读。

## 4.1.2 函数的定义和调用

### 1. 函数的定义

定义函数由 `def` 开始，其语法格式为：

```

def 函数名([形式参数表]):
    函数体

```

说明：

①函数名必须符合标识符定义

②形式参数表由逗号间隔的一组形式参数组成（可以缺省），形式参数简称形参，用于接受调用函数向函数传递数值或返回结果。

③函数体由一组语句组成，书写时右缩，体现从属关系。函数执行过程中，遇 `return` 语句，执行控制权返回到调用函数；无 `return` 语句时，则执行到函数最后。

④`return` 语句的格式为：

```

return 表达式          # 表达式可以省略

```

`return` 的作用是，如果表达式不省略，则表达式的值被返回给调用函数。表达式省略或函数无 `return` 语句时，函数本身返回 `None`。

**【例 4.2】** 编写求一元二次方程实根的函数 `quadricEquation(a,b,c)`，并调用它求解  $2x^2+7x-3=0$  的根。

分析：一元二次方程由三个参数 `a`、`b`、`c` 确定，可能求得 2 个根，也可能求不出根。考虑列表可以包含多个数值，因此可以将两个根组合成一个列表返回，当无实根时，返回空列表。故此定义函数如下：

```

import math
def quadricEquation(a,b,c):
    delta=b*b-4*a*c
    if delta>=0:
        root1=(-b+math.sqrt(delta))/(2*a)
        root2=(-b-math.sqrt(delta))/(2*a)
        return [root1,root2]
    else:
        return []
print('求方程 2x^2+7x-3=0 的根')
x,y,z=2,7,-3
roots=quadricEquation(x,y,z)          #函数有返回值，函数出现在表达式中

```

```
print('roots=',roots)
```

程序的运行结果为:

求方程  $2x^2+7x-3=0$  的根

roots= [0.386, -3.886]

本例中的函数无论走哪路 if 分支, 都通过 return 语句返回, 语句:

```
roots=quadricEquation(x,y,z)
```

使得变量 roots 保存函数的返回结果。

**【例 4.3】** 编写程序, 打印图 1.4.2 所示图形

```
      *
     ***
    *****
   *********
  ***********
 *
 ***
 *****
 *******
 *****
 *****
 *****
```

图 1.4.2 例 4.3 输出图形

分析: 图形由两个三角形组成, 只是构成三角形的行数不同, 所以可编写函数输出指定行数 n 的图形, 该函数无需返回值, 所以 return 语句可有可无。代码如下:

```
def pict(n):
    i=0
    while i<n:
        print("{} {}".format(" "*(10-i),"*"*((2*i+1))))
        i += 1

    pict(4)                                #制作 4 行的图, 函数无返回值, 调用以语句形式调用
    pict(6)                                #再制作 6 行的图
```

再如编写函数实现对任意给定的实数列表, 将其偶数项元素加 1, 奇数项元素减 1。由于列表是一种带分量的复合数据类型, 函数中只要不对列表本身赋值, 则列表可以将数值的改变带回实参, 所以函数无需返回值, 故定义函数如下:

```
def handleList (x):          # x 是一个列表
    for i in range(len(x)):
        if i% 2==0:
            x[i] +=1
        else:
            x[i] -=1
    li=[1,2,3,4,5]
```

```
handleList(li)          #函数无返回值，直接写函数名
print(li)
```

程序的执行结果为：

```
[2, 1, 4, 3, 6]
```

上述代码中，因为 python 参数采用引用方式，函数体内只要不改变列表的首地址，列表有能力将其含有的值返回给调用函数，所以不需要 return 语句。

对无 return 语句或 return 语句后无表达式的函数，因为函数返回 None，所以调用这种函数时，直接写函数名即可。例如，对函数 handleList 的调用语句改写成：

```
print(handleList(li))
```

则程序的执行结果为：

```
None
```

```
[2, 1, 4, 3, 6]
```

None 是函数的返回值，由于函数无返回值，所以其输出是无意义的。

## 2. 函数调用

函数调用与内部函数相同，调用函数的形式为：

函数名(实在参数表)

其中：

- ①实在参数简称实参，可以是常量、表达式、变量。
- ②一般情况下，实参与形参顺序一一对应，类型兼容，其作用是用实参的值初始化形参。
- ③为保证语义完整，有返回值的函数，函数调用应出现在表达式中（例 4.2），对于无返回值的函数，则应以语句形式出现（例 4.3）。

函数调用时的执行过程如下：

- （1）计算实参的值
- （2）实在参数的值传递给相应的形式参数
- （3）控制转移到被调用函数，执行函数体中的语句
- （4）遇到 return 语句，将 return 后表达式的值作为函数值返回到调用函数，或执行到函数体结束，将控制转回调用函数。
- （5）继续执行调用函数中的后继语句。

以例 4.2 为例，其调用过程示意图如图 1.4.3 所示：

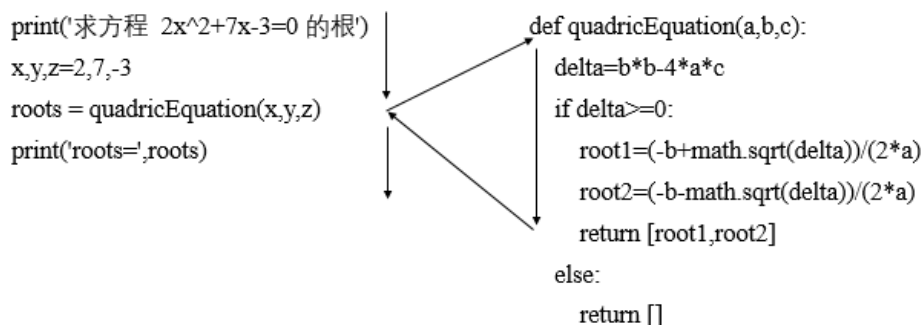


图 1.4.3 函数 quadricEquation 的调用过程

## 4.2 参数及传递

调用函数时，主调函数和被调用函数之间通常存在数据传递，即将实参传递给形参，然后执行函数体。Python 的形参分为三种类型：位置参数、默认值参数、可变长度参数。不同类型同时出现时，会给实参与形参的结合带来一些差异。

### 4.2.1 位置参数及默认值参数

#### 1. 位置参数

形参定义时标识符前后无任何修饰的参数就是位置参数，位置参数也称必选参数。这类参数在函数调用时，必须有对应的实参。

如例 4.2 中的一元二次方程函数，函数头的定义为：

```
def quadricEquation(a,b,c):
```

参数 a,b,c 都是必选参数，调用该函数必须有三个实参。例如下面代码少给一个参数：

```
x,y,z=1,8,3
roots=quadricEquation(x,y)
```

运行时，系统的信息如下：

```
TypeError: quadricEquation() missing 1 required positional argument: 'c'
```

即系统认为对应于“c”的位置参数没有提供。

再如多给一个参数：

```
x,y,z=1,8,3
roots=quadricEquation(x,y,z,10)
```

系统提示如下信息：

```
TypeError: quadricEquation() takes 3 positional arguments but 4 were given
```

即函数 quadricEquation 带有 3 个位置参数，但调用时给了 4 个，也是不合法的。

#### 2. 默认值参数

在函数定义时，直接被赋值的参数称为默认值参数，默认值参数又称可选参数。调用函数时，对应默认值形参的实参，可有可无。无对应实参时，取其默认值；有对应实参时，则取实参值。语法规则，默认值参数必须放在位置参数的后面。格式如下：

```
def 函数名(必选参数 1,...,必选参数 n,可选参数 1=值 1,...,可选参数 n=值 n):
```

```
    函数体
```

**【例 4.4】**设计函数计算下面级数的部分和

$$s = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$$

要求最后一项的绝对值小于  $\text{eps}$ ，即： $\left|\frac{x^n}{n!}\right| < \text{eps}$ 。若用户不指定精度，则  $\text{eps}$  取  $10^{-6}$ 。

分析：根据题意， $\text{eps}$  可采用默认值参数。所以可定义函数如下：

```
def s(x, eps=1e-6):
    w=0
    t=1
    n=1
    while abs(t)>=eps:
        w +=t
        t *= x/n
        n +=1
    return w
print(s(1))           #省略默认值参数调用
print(s(1,1e-8))      #指定默认值参数的值
```

程序运行结果为：

2.7182815255731922

2.718281826198493

可以发现两次运行结果，在百万分之一位后开始有差异。

### 3. 形参名称对应

函数参数较多时，按位置一一对应方式易导致程序可阅读性差，调用容易出错。此时可以直接指定形参的值，形式为：形式参数名=表达式，这种调用方式叫做形参名称对应调用，它不必考虑形参的先后次序。

一元二次方程求解函数三个参数的名字分别为  $a, b, c$ ，采用指定形参名字传递的代码如下：

```
x,y,z=1,10,3
roots=quadraticEquation(a=x, c=z, b=y)    # 指定形参关键字
print('指定形参名字: ',roots)
```

请特别注意，这种调用方式中，一旦一个参数开始使用了形参名字调用，后续的所有参数都必须使用该方式。例如下面的语句：

```
x,y,z=1,10,3
roots=quadraticEquation(a=x, z, y)        # 指定形参 a，但 b 和 c 不指定。
```

SyntaxError: positional argument follows keyword argument

系统认为， $a=x$  关键字调用方式，其后面也必须采用这样的方式，但现在跟随了位置参数。

而下面的调用则是合法的，因为  $c=z$  指定形参名字后， $b$  也被指定。

```
x,y,z=1,10,3
roots=quadraticEquation(x,c=z,b=y)       # c=z 开始形参关键字，后续 b=z 也指定
```

## 4.2.2 可变长参数

某些情形下，需要处理的数据的个数是不定的，如下面的示例代码求两组数的最大值。

```
ma1=max(10,123.5,-34,-1000,12.34)
print(ma1)
ma2=max(89.1,45.1,-89)
print(ma2)
```

可以看到，两次传递给 `max` 函数的数据个数是不同的，但程序都实现了正确的计算。此时显然不能再用位置参数和默认值参数的方式定义函数，python 的解决方案是可变长度参数。

## 1. 定义及使用规则

可变长参数意指参数对应的实参个数可变。这类参数的定义采用两种方式：

①参数的前面带\*，它将对的一组实参转化为一个元组。

如下面代码，定义了可变长参数 `arg`，它搜集主函数传递的任意多个参数，函数的输出证明，实参被组成了一个元组。

```
def test(*arg):
    print(arg)
test(1,10.5,4)
```

程序运行结果为：(1, 10.5, 4)

②参数的前面带\*\*，对应的实参以“标识符=值”的方式传递。该类可变长参数将对应的实参，以“标识符”为键，以“值”为值，转化为字典，\*\*参数也被称为关键字参数。如下面的代码，函数的输出证明搜集的参数形成字典。

```
def test(**arg):
    print(arg)
test(a=1,b=10.5,c=4)
```

代码输出为：{'a': 1, 'b': 10.5, 'c': 4}

可变长参数必须在形参表的最后出现。所以 python 函数形参表中各种类型参数出现的次序依次为：位置参数、默认值参数、可变长度参数。

**【例 4.5】** 定义函数，给定一个字符串指令（求和、求平均、累乘），用可变长参数对后续的任意个实数进行相应的处理。代码如下：

```
def calcu(cmd='求平均',*data): # data 收集多个实参形成是一个元组
    print(data)
    if cmd=='求平均':
        s=sum(data)/len(data)
        return s
    elif cmd=='求和':
        return sum(data)
    else:
        s=1
        for i in data:
            s*=i
        return s
res=calcu('求和',3,3,100)
print(res)
```

程序的运行结果为：

(3, 3, 100)

106

calcu 函数的定义中，data 是可变长参数，它搜集实参中第二个参数开始的所有参数，形成一个元组。

**【例 4.6】** 定义函数，收集实参形成字典，并输出实参的名字和值。

```
def keyArgFunc(**keyarg):      # keyarg 是一个字典
    for x in keyarg:            # 取字典中的每个键
        print( x + ": " + str(keyarg[x]))    # keyarg[x]取键对应的值
keyArgFunc(no=10,name='李彤',spec='学生',age='20')
```

程序的运行结果如下：

no: 10

name: 李彤

spec: 学生

age: 20

上述函数定义了一个可变长关键字参数 keyarg，调用时，no=10,name='李彤'等被关键字参数收集形成字典，所以就得到了所见的运行结果。

最后要注意的一点是，在位置形参、默认值形参与可变长关键字都存在的函数中，其调用规则不符合形参名字方式。因为一旦一个参数使用形参名字调用，后续参数也必须用这种模式，但可变长度形参，对应着一组实参。如例 4.5，如果将调用语句改写为 res=calcu(cmd='求和',3,3,100)，则语法要求后续的三个实参（3，3，100）也必须按形参名字传递，这与函数形参\*data 的定义是不符合的。

## 2. 实参的序列解包

对序列类型的实参，在其前加\*，称为实参解包，可以配合可变长度形参使用。

例如将一元二次方程的三个系数保存在一个序列中，通过实参解包调用函数求解，程序代码如下：

```
def quadricEquation(a,b,c):      #参见例 4.2
    .....
    L=[2,7,-3]                  # L 是列表
    print(quadricEquation (*L))  # L 前加*，传递前解包
```

运行机制为，实参\*L 解包后去掉列表的方括号，形成对应于形参 a,b,c 的三个实参。

**【例 4.7】** 利用实参解包实现矩阵转置。

对二维列表而言，如 X=[[1,2,3],[4,5,6],[7,8,9]]，其解包的结果是三个列表：[1,2,3]、[4,5,6]、[7,8,9]，利用 zip 函数，将三个列表的对应元素组合在一起形成元组，这就实现了将矩阵的列转为对应的行，再将元组转换为列表即可实现函数功能。实现矩阵转置的程序代码为：

```
X=[[1,2,3],[4,5,6],[7,8,9]]
Xt=list(map(list,zip(*X)))    # list、map、zip 函数请参见第五章
```

```
print(Xt)
```

程序的执行结果如下：

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

### 4.2.3 形参对实参的影响

Python 中实参和形参间传递，全部采用引用的方式，即给对象挂标签的方式。按数据类型，可以分成简单类型和复杂类型形参进行讨论。

#### 1. 简单类型形参

简单类型的形式参数，要对其值进行变更，必须对其进行赋值，导致形参重新绑定到其它数值上，所以不会影响实参的值。

**【例 4.8】** 寻找三个数 x，y，z 中最大者。代码如下：

```
def getMax(x,y,z,Max):
    if x<y:
        x,y=y,x
    if x<z:
        x,z=z,x
    Max=x
    print('函数求得最大=',Max)
x,y,z=10,47,-1
Max=x
getMax(x,y,z,Max)
print('主函数中 Max=',Max)
print('x={ },y={ },z={ }'.format(x,y,z))
```

程序的运行结果如下：

```
函数求得最大= 47
```

```
主函数中 Max= 10
```

```
x=10,y=47,z=-1
```

从程序运行的结果看，函数中实现了寻找最大值，但最大值没有通过形参 Max 返回给调用函数，同时，函数中对 x, y, z 的值进行了变更，但主函数中它们的值并没有改变。

#### 2. 复杂类型形参

复杂类型是指变量本身含有分量，如序列及后续课程内容中的面向对象编程中的对象。这类形参值的改变又可分为两种情况，形参本身值和形参分量值的改变。

##### ①形参本身值改变

形参本身值的改变，使得形参变量标签被重新绑定，所以不会影响实参的值。

**【例 4.9】** 编写函数检验形参本身值改变不影响实参，编写函数求得斐波那契数列的前 n 项

```
def fibs(ans,n):
    for i in range(n-2):
        ans =ans+[ ans[-1]+ans[-2] ]
```



```
print(ans)
start=[0,1]
fibs(start,5)
print(start)
```

程序的运行结果如下：

```
[0, 1, 1, 2, 3]
```

```
[0, 1]
```

可以发现，函数中实现了求取斐波那契数列，但主函数的中的 `start` 值并没有改变。其原因在于函数中对形参 `ans` 进行了重新赋值。

## ②形参分量值改变

形参分量值改变，不会导致形参本身的重新绑定，所以会导致实参的变更。

**【例 4.10】** 编写函数验证形参分量的改变对实参的影响，求取数值列表每项的平方

```
def power2(li):
    for i in range(len(li)):
        li[i]=li[i]**2
    print(li)                                # 验证函数内部形参值的变化

start=[1,2,3,4,5]
power2(start)
print(start)
```

程序的执行结果如下：

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

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

可以发现，由于函数中只是对形参的分量进行赋值，不响应形参本身的绑定，所以形参导致了实参值的变更。

总结而言，形参对实参值的改变遵循以下原则：

①函数体的实现语句中，如果形参出现在“=”的左侧，导致其地址变更（标签再挂），则后续操作中，形参值的改变不影响实参的值。

②形参是列表、对象等复杂类型，形参对列表元素或对象属性值的变更，将导致对应实参的变更。因为这种操作不会影响形式参数的地址。

## 4.2.4 变量的作用域

变量的作用域，是指程序代码对变量使用的有效范围。在函数外定义的变量，被称为全局变量，函数内部也可以定义变量，称为局部变量。

### 1. 全局变量

全局变量定义后可以被整个程序后续代码使用。可以分为两种情形：隐式使用和显式声明使用。

### ①隐式使用

如果在函数中引用了一个变量的值，而该变量在函数中并未定义，但调用函数前，有同名的全局变量存在，则函数中使用的是全局变量。

#### 【例 4.11】隐式使用全局变量

```
def fx(n):  
    z=100*x          # 函数内未定义 x，但调用函数前，定义了全局的 x  
    print("z=",z,'x=',x)  
  
x=1                  #全局变量 x  
fx(3)  
print ("全局的 x=", x)
```

看程序的运行结果：

z= 100 x= 1

全局的 x= 1

可以发现，函数中使用的 x，是全局变量 x。

请特别注意，在函数内，变量处于赋值符的左边，则是函数内定义的局部变量，出现在赋值符=的右侧，则是引用全局变量。

### ②显式声明指定全局变量

为避免全局变量在函数中未定义就使用而引起的不确定性，可通过 global 关键字强调使用全局变量。

#### 【例 4.12】显式使用全局变量

```
def fx(n):  
    global x          # 明确使用全局变量 x  
    z=100*x  
    print("z=",z,'x=',x)  
    x=1000            # x 是全局的  
    print('x=',x)  
x=1  
fx(3)  
print ("全局的 x=", x)
```

程序的运行结果：

z= 100 x= 1

x= 1000

全局的 x= 1000

此时，即使函数中将变量 x 处于赋值符的左边，x 仍是全局变量，会影响其在主程序中的取值。

## 2. 局部变量

函数中定义的变量，包括形式参数，都是函数的局部变量，只可在其定义的函数内访问。

#### 【例 4.13】局部变量覆盖全局变量及局部变量的全局访问测试

程序中定义了全局变量 `x`，在函数 `fx` 内定义了局部变量 `x`，函数体内它将覆盖全局变量 `x`，主函数中尝试访问局部变量 `z`。代码如下：

```
def fx(n):
    x=1000*n          # 函数内，局部的 x
    print("局部的 x=",x)
    z=10

x=1                  # 全局 x
fx(3)
x+=1
print("全局的 x=", x)    # 主程序，使用全局的 x
print("访问局部变量 z=",z)
```

程序的运行结果如下：

局部的 `x= 3000`

全局的 `x= 1`

`print("访问局部变量 z=",z)`

`NameError: name 'z' is not defined`

可以发现两个变量 `x` 互不影响。但主程序中对 `z` 的访问导致程序出错。

## 4.3 lambda 函数

`lambda` 函数指通过 `lambda` 表达式定义的函数，是对逻辑特别简单的函数的快速实现。`lambda` 表达式是一个匿名函数，即没有函数名的函数，可以在某些场合直接使用，如对象列表的排序等。`lambda` 表达式的格式为：

**lambda 形参表:表达式**

如果给 `lambda` 表达式赋予名称，它就是 `lambda` 函数，其格式为：

**函数名=lambda 形参表:表达式**

其中，表达式的值是函数的返回值，函数的调用方式仍然是：函数名(形参表)。

**【例 4.14】** 用 `lambda` 函数计算给定列表的平均值和标准偏差。

```
aver=lambda L: sum(L)/len(L)
sigma=lambda L,aver:math.sqrt(sum([(i-aver)**2 for i in L])/(len(L)-1))
Li=[1,2,100,45]
average=aver(L)
print(sigma(L,average))
```

程序的执行结果如下：

37.0

46.74041791283714

上面例子中定义了两个 `lambda` 函数，`aver` 函数实现求给定列表 `L` 的平均值，`sigma` 函数实现求列表 `L` 的标准偏差。

## 4.4 递归函数

递归是程序设计的一种重要方法，递归函数具有结构清晰、可读性强的特点。

### 4.4.1 什么是递归

在日常生活中，递归是一种常见的现象。例如，两面相对的镜子所产生的被称为德罗斯特效应的“像中像”，还有“从前有座山，山上有座庙，庙里有个老和尚……”的故事，都是递归现象。

在数学中，许多问题都是通过递归定义的。例如，斐波那契数列的递归定义为：

$$\text{fibs}(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ \text{fibs}(n-1) + \text{fibs}(n-2) & n \geq 3 \end{cases}$$

即  $\text{fibs}(n)$  可以分解为  $\text{fibs}(n-1)$  和  $\text{fibs}(n-2)$  两个字问题。在程序设计中，递归是一种与迭代相对应的设计思想。当一个问题可以转化为规模较小的同类子问题时，就可以使用递归。递归函数是指在定义时直接或间接调用自身的函数。递归分成两种类型：

- ① 直接递归：一个函数直接调用自身。例如，函数  $f$  调用自身，记作  $f \rightarrow f$
- ② 间接递归：一个函数调用其他函数，而其他函数又调用该函数。例如，函数  $f$  调用函数  $g$ ，而函数  $g$  又调用函数  $f$ ，记作  $f \rightarrow g \rightarrow f$ 。

在实际应用，常见的是直接递归，间接递归较少应用。

【例 4.15】编写递归函数  $\text{fact}(n)$ ，求阶乘  $n!$  程序。

假定求  $n!$  的函数为  $\text{fac}(n)$ ，根据  $n!$  的定义可知递推公式为：

$$\text{fact}(n)! = \begin{cases} 1 & n=1 \\ n * \text{fact}(n-1) & n>1 \end{cases}$$

因此，可以编写如下  $\text{fac}(n)$  函数并且加以调用：

```
def fact(n):  
    if n==1:                                # 递归结束条件  
        return 1  
    else:  
        return n*fact(n-1)                  # fact 函数调用自己计算(n-1)!  
n=int(input())  
result=fact(n)                               # 调用 fact 函数  
print(result)
```

在函数  $\text{fact}$  定义中， $n==1$  是递归结束条件（又称递归边界条件）。若没有递归结束条件，递归将无限进行下去；当  $n>1$  时，调用  $\text{fact}(n-1)$  求  $(n-1)!$ ，然后再求  $n * \text{fact}(n-1)$ 。

分析程序执行过程，如图 1.4.4 所示，递归的过程可以分成两个子过程：递推和回归，

如图 1.4.5 所示。

① 递推 即 `fact(4)`调用 `fact(3)`，`fact(3)`调用 `fact(2)`，`fact(2)`调用 `fact(1)`，`fact(1)`直接计算。

② 回归 当 `fact(1)`计算完毕后，先由 `fact(1)`计算得到 `fact(2)`，再由 `fact(2)`计算 `fact(3)`，最后由 `fact(3)`得到 `fact(4)`。

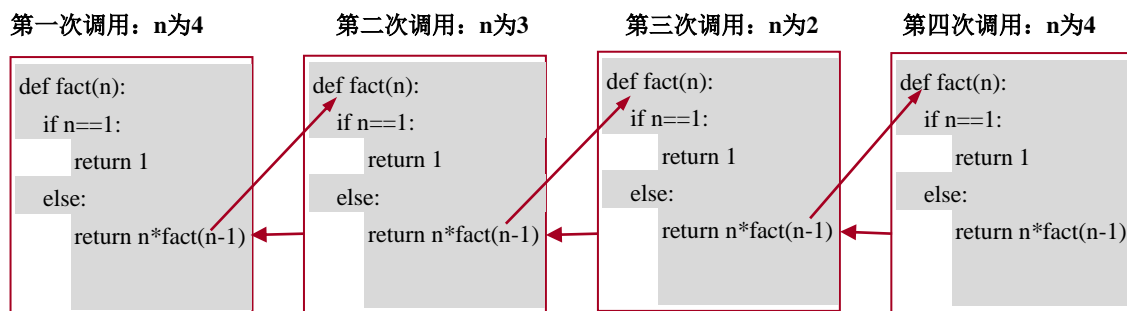


图 1.4.4 求 4! 的执行过程

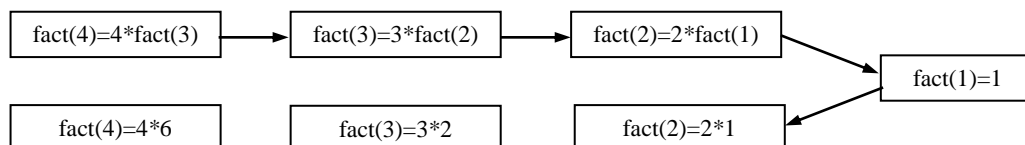


图 1.4.5 求 4! 的递推和回归过程

从问题求解的角度来说，计算  $n!$  问题可以分解成更小的问题（子问题）：计算  $(n-1)!$  问题，当计算  $(n-1)!$  问题解决后，计算  $n!$  问题也就解决。由于计算  $(n-1)!$  是计算  $n!$  的子问题，并且算法是相同的，仅仅是参数值不同，因此可以用递归解决。

由此可见，编写递归程序的关键有两个：

#### ① 问题分解

将一个问题分解成若干个子问题，若其中的子问题与整个问题的算法是相同的，仅仅参数值不同，则可以用递归方法解决。

#### ② 构造递归

当问题最小化时，递归结束。若没有递归结束条件，则会无限递归，程序出错。

## 4.4.2 递归函数的设计

递归函数的设计过程一般分成三个步骤：问题分解、归纳出递归模式、编写代码。

### 1. 问题分解

不是所有问题都可以或适合用递归求解。一个问题是否适合用递归求解，可以从两个方面思考。

① 一个问题很小时，是否有一个直接的结果，即递推的结束条件。

② 当一个问题不小时，是否可以将其分解为一个规模较小的问题，且可以一直分解下

去。

以字符串逆转为例，就符合上述 2 个条件。

当字符串  $s$  中只有一个字符时（最小化时），其逆转就是本身。

当字符串  $s$  中有多个字符时，其每次操作是字符串的分片  $s[1:]$  逆转的结果和第一个字符  $s[0]$  的+运算，而  $s[1:]$  比  $s$  本身少了一个字符，规模缩小了。所以字符串的逆转问题可以用递归来实现。

## 2. 归纳出递归模式

递归模式是根据问题规模的逐步缩小而总结出来的逻辑关系，是递归程序算法实现的关键点。如字符串的逆转问题。设字符串  $s="ABC"$ ，将其转化为" $CBA$ "的过程如下：

①  $s="ABC"$ ，转换为" $BCA$ "，是  $s[1:]$  的逆转和  $s[0]$  的+运算。

②  $s="BC"$ ，转换为" $CB$ "，也是  $s[1:]$  的逆转和  $s[0]$  的+运算。

③  $s="C"$ ，长度为 1，转换过程结束。

总结上述规律，即可归纳出字符串的递归模式如下：

子字符串  $s[1:]$  的逆转与  $s[0]$  的+运算

## 3. 编写代码

算法实现是根据递推公式编写递归函数的过程，遵循一般函数的定义语法，但强调递归的终止条件及问题分解规则。

**【例 4.16】** 编写函数实现字符串的逆转

```
def rev(s):
    if len(s)==1:
        return s
    else:
        return rev(s[1:])+s[0]
print(rev("ABCDEF"))
```

**【例 4.17】** 用递归的方法编程求两个正整数的最大公约数。

求最大公约数的递推公式为：

$$\text{gcd}(m,n) = \begin{cases} n & m \% n = 0 \\ \text{gcd}(n, m \% n) & m \% n \neq 0 \end{cases}$$

分析：求  $m$  和  $n$  的最大公约数  $\text{gcd}(m,n)$  可以分解成子问题：求  $n$  和  $\text{gcd}(n, m \% n)$  的最大公约数  $\text{gcd}(n, m \% n)$ 。由于求解子问题和整个问题的算法是相同，仅仅是参数不同，如图 1.4.6 所示，因此可以用递归方法解决。递归的结束条件为  $m \% n=0$ ，此时最大公约数为  $n$ 。

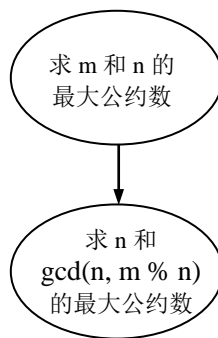


图 1.4.6 求最大公约数问题

程序如下：

```
def gcd(m,n):
    if (m % n==0):                # 递归结束条件
        return n
    else:
        return gcd(n,m % n)      # 递归调用
m,n=eval(input())
result=gcd(m,n)
print(result)
```

**【例 4.18】**汉诺塔问题，又称为“世界末日问题”。该问题是：有 A、B、C 三根柱，A 柱上从下到上放了半径由大到小的六十四片圆金片，要求借助 B 柱，把这些金片全部由 A 柱移到 C 柱，一次只能移一片。移动时，金片的大小次序不能颠倒，小片永远只能放在大片上面。设金片数为  $n$ ，则移动次数= $2^n-1$ 。

盘子的搬迁的过程如图 1.4.7 所示：

汉诺塔移动的过程，是借助 C 柱移动到 B 柱，再借助 A 柱移动到 C 柱，其过程循环往复，因而可以总结出如下的递归步骤：

- ①当只剩下一个盘片时， $n=1$ ，递归结束
- ②将上面的  $n-1$  块盘子，借助 C 柱，移动到 B 柱（子问题 1）
- ③将最下面的第  $n$  块盘子，从 A 柱移动到 C 柱
- ④借助 A 柱，将  $n-1$  块盘子，从 B 柱子移动到 C（子问题 2）

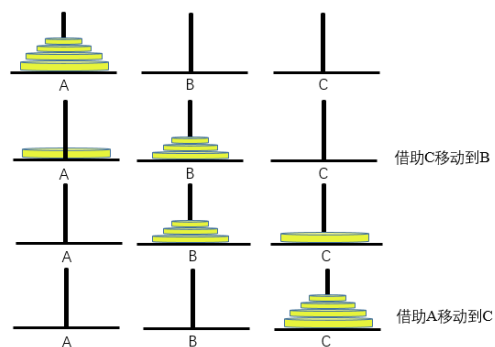


图 1.4.7 汉诺塔盘片的移动逻辑

经过问题分解， $n$  个盘子的移动被分解移动 2 个  $n-1$  块盘子和移动第  $n$  块盘子的问题，因而可以用递归方法实现，如图 1.4.8。

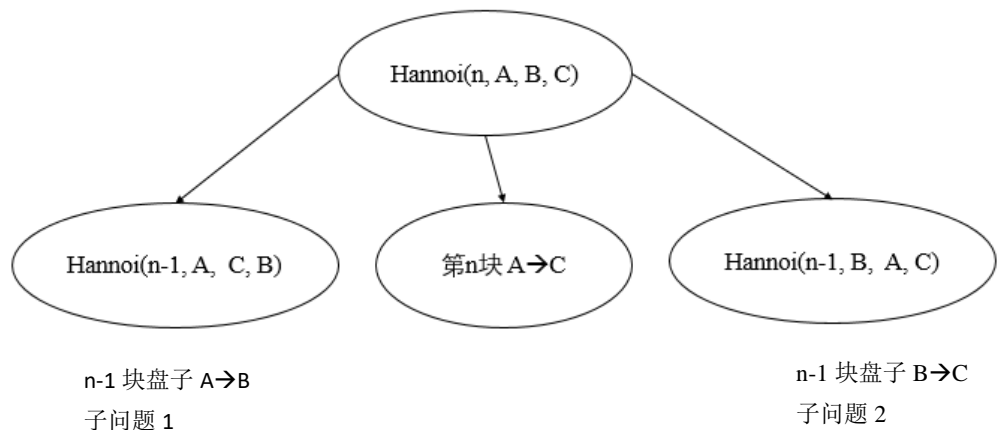


图 1.4.8 汉诺塔问题分解

设移动总盘数为  $n$ ， $A$ 、 $B$ 、 $C$  分别用字符“A”、“B”、“C”表达。函数返回移动的次数，汉诺塔移动的递归函数可定义如下。

```

def Hanoi(n, A, B, C):
    if n == 1:
        print(A+"→"+C)
        return 1
    else:
        t1=Hanoi(n - 1, A, C, B)          #n-1 盘子，借助于 C，从 A 移到 B
        print(A + "→" + C)                # 最后一个盘子，从 A 移动到 C
        t2=Hanoi(n - 1, B, A, C)          # n-1 个盘子，借助于 A，从 B 移到 C，
        return t1+t2+1

times=Hanoi(3,"A","B","C")               # 借助列表的值，函数中改变可传递回来
print('移动次数=',times)
  
```

主函数的调用中，设盘子总数为 3，程序的运行结果如下：

A→C

A→B

C→B

A→C

B→A

B→C

A→C

移动次数= 7

**【例 4.19】** 根据 4.4.1 描述的斐波那契数列每项的计算公式，编写递归函数求解数列的任意指定项。

分析：公式描述中已经完整描述递归的终止调节及递归的问题分解，按递归公式可以编



写函数代码如下：

```
def fibs(n):
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return fibs(n-1)+fibs(n-2)

print(fibs(2))
```

代码运行结果为 1.

## 4.5 综合应用

本章介绍了函数的实现及函数调用中应该注意的主要问题，函数是实现程序模块化设计的基础，是面向对象程序设计的前导内容，是程序设计中锻炼问题抽象的重要步骤。本节通过一些综合案例向读者介绍函数设计的具体应用场合。

### 1. 二分法求高次方程的根

二分法是求解一元高次方程根的常用方法之一，参见图 1.4.9，设函数  $f(x)$  连续，如果通过测试找到两个点  $a$ ,  $b$ ，使得  $f(a)*f(b)<0$ ，则可以断定，函数在  $[a, b]$  中必定存在一个根。求解过程为：取中间值  $m=(a+b)/2$ ，判断  $f(m)$  的符号与  $f(a)$  是否同号，如果同号则令  $a=m$ ，否则令  $b=m$ ，不断缩小求解区间，并且保持  $f(a)*f(b)<0$ ，当  $a$  和  $b$  无限接近时，例如  $\text{abs}((a-b)/a)$  的绝对值  $<1e-6$ ，则认为找到了解。

化学中弱酸弱碱盐溶液 pH 求解问题是一个一元高次函数，以求解浓度为  $c$  mol/L 的任意一元弱酸溶液 pH 值为例，该类溶液都可以根据弱酸的解离方程、水溶液质子平衡、水的离解常数等化学平衡知识，经过推导，将氢离子浓度  $x$  表达为如下方程：

$$x^3 + K_a x^2 - (K_a c + K_w) x - K_a K_w = 0$$

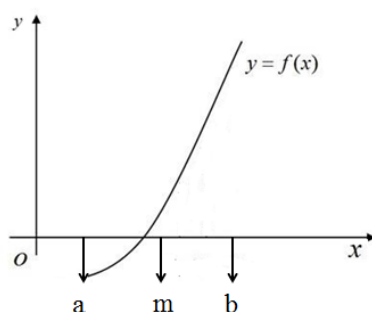


图 1.4.9 二分法求高次方程的根示意图

其中  $k_a$  是一元弱酸的电离常数，对给定的酸是固定值，如醋酸的值为  $10^{-4.56}$ ， $K_w$  是

水的电离常数，为  $10^{-14}$ 。

**【例 4.19】** 求解指定浓度的醋酸溶液的 pH 值，将高次方程和二分法分别用函数实现。该问题可以归纳为两个函数，一个是高次方程本身，它是  $x$  和浓度  $c$  的函数。为调用函数方便起见，规定函数中使用的  $c$  是全局变量，所以可以定义如下：

```
def fx(x):
    global c                # 使用全局变量
    ka=10**(-4.56)          # 醋酸的 ka
    kw=1e-14
    kakw=ka*kw
    f=x**3 + ka * x**2 -(ka * c + kw)*x-kakw
    return f
```

问题中的另一个函数是二分法求解逻辑，它接收给定的区间  $[a, b]$ 、高次方程函数作为二分法函数的参数，根据算法原理求根并返回。所以可以定义函数如下：

```
def biAlgorithm(a,b,func):   # func 对应高次方程函数，作为二分法的参数
    fa=func(a)               # func 对应 fx，fx 中使用了全局的 c，简化处理
    while (abs(a-b)>1e-10):
        m=(a+b)/2
        fm=func(m)
        if (fa*fm>0):
            a=m
        else:
            b=m
    return a
```

在上面两个函数支持下，主程序的任务是获取溶液的浓度，找到合理的区间，并传递给二分法函数。所以主程序定义如下：

```
fa=1
fb=1                        # fa 和 fb 可以随意取值，保证进入下面的 while 循环即可
c=float(input('请输入醋酸浓度:'))
while(fa*fb>0):             # while 循环寻找合理区间
    a=float(input('请输入左区间: '))
    b=float(input('请输入右区间'))
    fa=fx(a)
    fb=fx(b)

    answer=biAlgorithm(a,b,fx)
import math
pH=-math.log10(answer)
print(pH)
```

程序的运行过程及结果如下：

请输入醋酸浓度:0.01

请输入左区间: 0

请输入右区间 1

pH= 3.291394771168757

## 2. 递归制作树形分型图

树形分形图的图元由一个主树干和树干上生长的 3 个子枝形成，其问题分解步骤是：

(1) 从主干底部计算，分别在  $1/3$ 、 $1/2$ 、 $2/3$  的位置上，以左转  $30^\circ$ 、右转  $30^\circ$ 、左转  $30^\circ$  的角度，分别绘制树枝  $A_1B_1$ 、 $A_2B_2$ 、 $A_3B_3$ ，三个树枝的长度分别为主干的  $0.5$ 、 $0.5$ 、 $0.5$ ，如图 1.4.10 所示。

(2) 将每个树枝作为主树干，重复上述步骤 1 的工作。

(3) 终止条件：当树干的长度小于 20 时，递归终止。

(4) turtle 模块

为利用制图展现程序的递归过程，采用小乌龟 turtle 模块实现问题的递归程序。turtle 以一个小海龟的爬行轨迹来完成制图，小海龟的初始状态为头向右，通过旋转、前进、后退、提笔、落笔等函数实现制图。应用 turtle 制图时，先在程序中通过下列语句引用

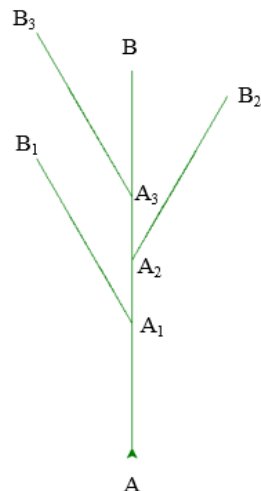


图 1.4.10 树形图元

```
import turtle
```

之后，可以参阅表 1.4.1 完成制图。

表 1.4.1 turtle 制图函数

函数	作用
<code>turtle.color(color)</code>	设置轨迹颜色为 color
<code>turtle.penup()/pu()</code>	提笔
<code>turtle.pendown()/pd()</code>	落笔
<code>turtle.right(x)</code>	向右旋转 x 度
<code>turtle.left(x)</code>	向左旋转 x 度
<code>turtle.forwardward(x)</code>	沿当前方向前进 x 像素
<code>turtle.backward(x)</code>	沿当前方向后退 x 像素
<code>turtle.goto((x,y))</code>	从当前位置走到(x,y)位置

**【例 4.20】** 结合 turtle 制图原理与树形图案的实现原理，设计递归函数，制作树形分型图。

分析：由于制图过程中多次左右旋转角度，使程序的逻辑较难控制，为简单起见，每制作一个子枝时，都采取回到当前主干的底部再制作其他子枝的方案。所以小乌龟爬行的方案制定如下：

画当前 AB 主干，回到主干底部

前进  $1/3$ ，左转  $30^\circ$ ，画  $A_1B_1$  树枝，长度为主干的  $0.5$ ，再右转  $30^\circ$ ，退  $1/3$ ，回树干底部

前进  $1/2$ ，右转  $30^\circ$ ，画  $A_2B_2$ ，长度为主干的  $0.5$ ，再左转  $30^\circ$ ，退  $1/2$ ，回到树干底

部

前进  $\frac{2}{3}$ ，左转  $30^\circ$ ，画  $A_3B_3$ ，长度为主干的 0.5，再右转  $30^\circ$ ，退  $\frac{1}{2}$ ，回到树干底部

部

树干长度小于 20 时，停止递归

设计函数的代码如下：

```
import turtle as tt
import turtle
def draw_brach(brach_length):
    if brach_length>20:
        #画树主干
        turtle.forward(brach_length)
        turtle.backward(brach_length)          #返回底部
        #画低位树枝
        turtle.forward(brach_length/3)
        turtle.left(30)
        draw_brach(brach_length*0.5)
        turtle.right(30)
        turtle.backward(brach_length/3)        #回树枝最底部
        # 绘中枝
        turtle.forward(brach_length*0.5)
        turtle.right(30)

        draw_brach(brach_length*0.5)
        turtle.left(30) # 返回
        turtle.backward(brach_length*0.5)      #回树枝最底部
        # 绘上枝
        turtle.forward(brach_length*2/3)
        turtle.left(30)
        draw_brach(brach_length*0.5)
        turtle.right(30)
        turtle.backward(brach_length*2/3)      #回树枝最底部

    turtle.left(90)                            # 本来向右，改为向上，位置屏幕中央
    turtle.penup()                             #提笔
    turtle.backward(150)                       #向下退 150 像素
    turtle.pendown()                          #落笔
    turtle.color('green')
    draw_brach(400)                           # 主树干 400 像素
    turtle.exitonclick()                      # 点击鼠标退出制图界面
```

设定树枝最短长度为 20，制作的分形图如图 1.4.11 所示。

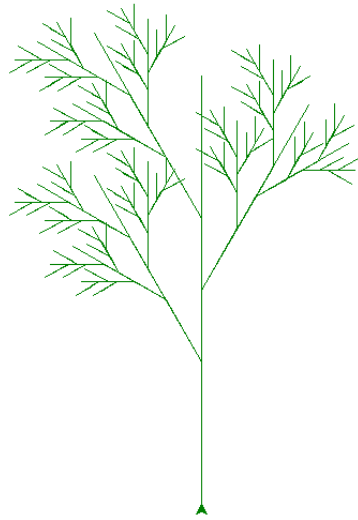


图 1.4.11 树枝最短长度为 20 的树形分型图

### 3. 统计字符串 s 中每个字符出现的次数

分析：函数接受给定的字符串 s，首先通过 set 函数找到不重复的字符列表 L，通过遍历 L，对每个字符 c，用字符串的 count 方法统计 c 出现的次数，也整理成一个列表 freq，返回 L 和 freq 即可实现任务。

**【例 4.21】** 编写函数 charFreq,它接受字符串 s，返回字符串中每个独立字符及其出现的次数。

```
def charFreq(s):  
    chars=list(set(s))          # 整理不重复的字符  
    chars.sort()  
    freq=[]                    #记录字符出现次数  
    for char in chars:  
        v=s.count(char)  
        freq=freq+[v]  
    return chars,freq  
  
str1=input('请输入字符串: ')  
charList,freqList=charFreq(str1)  
print(charList)  
print(freqList)
```

程序运行的过程如下：

请输入字符串：He is a teacher of this university.His name is Yange

[' ', '.', 'H', 'Y', 'a', 'c', 'e', 'f', 'g', 'h', 'i', 'm', 'n', 'o', 'r', 's', 't', 'u', 'v', 'y']

[9, 1, 2, 1, 4, 1, 6, 1, 1, 2, 6, 1, 3, 1, 2, 5, 3, 1, 1, 1]



C 按类型, 参数书写顺序必须是 X,z 和\*paras

D z=1 错误, 不能给参数常数值

## 二 填空题

### 1. 运行并记录函数的结果

```
x=12.6789
```

```
round(x) 结果: _____
```

```
round(x,2) 结果: _____
```

### 2. 记录代码的输出结果

```
a,b=2,4
```

```
s=eval('a*a+10*b')
```

```
print("s=",s) 结果: _____
```

### 3. 记录 math 库中函数的结果

```
x=2
```

```
y=3
```

```
pow(x,y) 结果: _____
```

```
exp(x) 结果: _____
```

```
sqrt(x) 结果: _____
```

```
sin(x) 结果: _____
```

```
log10(x) 结果: _____
```

```
degrees(x) 结果: _____
```

### 4. 记录函数的结果

```
s="Tongji"
```

```
s.upper() 结果: _____
```

```
s.islower() 结果: _____
```

```
s.isnumeric() 结果: _____
```

```
t="When I was young "
```

```
t.split(" ") 结果: _____
```

```
s.count("n") 结果: _____
```

```
s.replace("n", "t") 结果: _____
```

### 5. 下列函数执行后, 程序的执行结果分别为\_\_\_\_\_

```
def foo(t):  
    t=1+1  
    return t  
  
t=1  
print(foo(t),t)
```

6. 下列函数执行后，程序的执行结果分别为\_\_\_\_\_

```
def foo(t):  
    t[0]=t[0]+1  
  
t=[1,2]  
foo(t)  
print(t)
```

7. 下列代码的执行结果为\_\_\_\_\_

```
k,v=3,-1  
s=eval('k**2+10*v')  
print("s=",s)
```

8. 定义函数如下：

```
def average(*Li):  
    return sum(Li)/len(Li)  
y=average(1,2,9,8)  
print(y)
```

程序的执行结果为\_\_\_\_\_

9. 下面程序利用牛顿迭代法求方程  $f(x)=x^3-4x^2+5$  的根，请将程序补充完整，牛顿迭代法请自行查阅。

```
def f(x):                                # 原函数  
    y = x**3-4*x*x+5  
    return y  
def f1(x):                               #函数的一阶导数  
    y=3*x*x-8*x  
    return y  
def nt(x):                               # 牛顿迭代法  
    while True:  
        x0=__(1)____  
        y1=f(x0)  
        y2=__(2)____  
        x= x0-y1/y2  
        if(abs(f(x))<1e-6):  
            __(3)____  
    return(x)  
x0 = 100  
root=__(4)____  
print("Root=%7.5f"%(root))
```

10. 辗转相除法求两个整数  $m,n(m>n)$  的最大公约数，其原理是取  $r$  为  $m$  和  $n$  的余数，如果  $r=0$ ,则  $n$  就是最大公约数，程序终止，否则令  $m=n$ ,  $n=r$ ，然后再继续求新的  $m$  和  $n$  的最大公约数，所以可以用递归法实现，请填写空格中的代码使程序完整。

```
def gcd(m, n):  
    if __(1)____:  
        return __(2)____  
    else:
```



```
    r=__ (3) ____  
    m=n  
    return gcd(__ (4) __)  
print(gcd(12,18))
```

### 三. 简答题

1. 简述为何要编写函数，用户自定义函数给编程带来哪些优势？
2. 请简单描述用户自定义函数的格式，形式参数有哪几种类型？各在何种情况下使用？
3. 函数调用参数传递时，实参与形参有几种匹配传递方式，使用时需要注意哪些规则？
4. `lambda` 函数的定义格式是什么？在什么情况下适合使用 `lambda` 函数。
5. 用递归法求解问题的关键步骤有哪些？如何分析一个问题是否适合用递归方法求解？

# 实验一 Python 程序设计基础

## 一、实验目的

1. 掌握 Python 开发环境的下载和安装；
2. 通过简单的程序设计熟悉 IDLE 和 Anaconda 开发环境；
3. 掌握简单 Python 程序设计的基本组成；
4. 掌握 Python 输出/输出数据的基本方法。

## 二、实验内容

1. 在自己的电脑上，安装 Python 开发环境。
  - ① 安装 IDLE 的开发环境。说明：Python 官网网址为 <https://www.python.org>
  - ② 安装 Anaconda 的开发环境。说明：Anaconda 官网网址为 <https://www.anaconda.com>
2. 下面程序的功能是从键盘输入一个整数，然后计算它的平方根并且输出，保留两位

小数。请在 IDLE 中调试该程序。注：下列程序存在四个错误。

```
import "math"
x=input("请一个整数：")
y=sqrt(x)
print("平方根是：", "y")
```

3. 编写一个程序，要求输入直角三角形的斜边和一条直角边，求另外一条直角边、周长和面积。
4. 编写一个计算一个学生三门课平均成绩的程序。要求学生成绩从键盘输入。
5. 编写一个华氏温度与摄氏温度的换算程序。要求输入摄氏温度，计算并且输出华氏温度。

华氏温度=9/5\*摄氏温度+32

## 实验二 顺序结构

### 一、实验目的

1. 掌握数值、字符串、布尔类型数据的表示和常规处理；
2. 各类表达式的正确书写规则。
3. 掌握常用函数的使用。
3. 掌握 input 函数的使用。
4. 掌握 print 函数和格式的使用。

### 二、实验内容

1. 数学函数的使用。在 IDLE 交互方式运行下，导入数学库 `math`，输入对应变量的值，完成如下表达式的计算并显示计算结果。已知  $x=5$ ， $y=6$ ， $z=7$ 。

$$(1) \frac{\sqrt{(3x+y)/z}}{(xy)^4}$$

$$(2) \sin 45^\circ + \frac{e^{10} + \ln 10}{\sqrt{x+y+1}}$$

2. 字符串函数使用。已知字符串 `s="Python Programming Language"`，在 IDLE 交互方式运行下，利用字符串函数，完成如下功能处理：

- (1) 利用 `lower()` 函数将 `s` 字符串全部转换成小写字母并显示。
- (2) 利用 `upper()` 函数将 `s` 字符串全部转换成大写字母并显示。
- (3) 利用 `count()` 函数统计 `s` 字符串中出现字母“n”的个数。
- (4) 利用 `split()` 函数将 `s` 字符串按单词分离到列表中。
- (5) 利用 `replace()` 函数将 `s` 字符串中出现的空格由“,”替换。

3. 关系和逻辑表达式的书写。编程输入年份，判断该年是否是闰年并显示。

【提示】关键是书写正确的关系表达式。判断闰年应满足以下两个条件之一：

- ① 该年能被 4 整除但不能被 100 整除；
- ② 该年能被 400 整除。

4. 算术运算符的使用。编程利用 `input` 函数输入一个 3 位正整数，然后逆序输出。例如，输入 734，输出是 437。

【提示】利用算术运算符进行取余和整除来实现。

5. 字符串的切片和连接。编程输入 18 位身份证号码，取出身份证中的出生年月日，如图 2.2.1 所示显示。

【提示】

- (1) 利用对字符串的切片获得出生年月日；
- (2) 利用字符串的“+”将切片后的字符串以出生年月日的方式显示。

请输入18位身份证号: 310110199310154011  
身份证号 出生年月日  
310110199310154011 1993年10月15日

图 2.2.1 运行效果

。

## 实验三 选择结构

### 一、实验目的

1. 掌握逻辑表达式的正确书写形式。
2. 掌握单分支与双分支条件语句的使用。
3. 掌握多分支条件语句的使用。
4. 利用选择结构解决实际问题。

### 二、实验内容

1. 求一元二次方程的根。

要求：输入一元二次方程的三个系数 a、b、c，根据系数值，可得出如下三种根：

- (1)  $\Delta > 0$ ，两个实根
- (2)  $\Delta = 0$ ，重根，即相同得根
- (3)  $\Delta < 0$ ，无实数根

其中： $\Delta$ 为判别式，即  $b^2 - 4ac$

输出两个实根  $x_1$  和  $x_2$ ，若没有实根则输出信息：无实根。

2. 在购买某物品时，若所花的钱  $x$  在下述范围内，所付钱  $y$  直接按对应折扣支付：

$$y = \begin{cases} x & x < 1000 \\ 0.9x & 1000 \leq x < 2000 \\ 0.8x & 2000 \leq x < 3000 \\ 0.7x & x \geq 3000 \end{cases}$$

【提示】注意计算公式和条件表达式的正确书写。

3. 编一程序输入上网的时间计算上网费用，计算的方法如下：

$$\text{费用} = \begin{cases} 30 \text{元基数}, & < 10 \text{小时} \\ \text{每小时 } 2.5 \text{元}, & 10 \sim 50 \text{小时} \\ \text{每小时 } 2 \text{元}, & \geq 50 \text{小时} \end{cases}$$

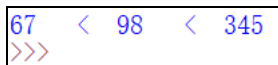
同时为了鼓励多上网，每月收费最多不超过 150 元。

【提示】

(1) 首先利用多分支 if 语句根据三个时间段分级计算费用，例如若  $s$  为使用了 80 小时，则计算费用公式  $m = 30 + 2.5 * 40 + (s - 50) * 2$ ；

(2) 然后再用单分支 If 语句对  $m$  超过 150 元的费用限定为 150 元。

4. 输入  $x$ ,  $y$ ,  $z$  三个数，按从小到大的次序显示，如图 2.3.1 所示：



```
67 < 98 < 345
>>>
```

图 2.3.1 运行效果

5. 利用计算机解决古代数学问题“鸡兔同笼问题”。即已知在同一笼子里有总数为  $M$  只鸡和兔，鸡和兔的总脚数为  $N$  只，求鸡和兔各有多少只？

【提示】鸡、兔的只数通过已知输入的  $M$ ， $N$  列出方程可解，设鸡为  $x$  只，兔为  $y$  只，则计算公式为：

$$x+y=M$$

$$2x+4y=N$$

$$\text{即： } x=M-y$$

$$y=N/2-M$$

但不要求出荒唐的解（例 3.5 只鸡、-4 只兔）。因此，对输入的数据要考虑下面两个条件：

- ① 对输入的总脚数  $N$  必须是偶数，否则提示数据错的原因；
- ② 若求出的头数为负数，提示数据错的原因。

6. 学分绩点计算。为评定学生的学习质量，高校一般采用计算学分绩点的办法，表 2.3.1 是某学校成绩与绩点的折算对照表。编程输入某门课程的成绩，利用 if 多分枝结构显示其成绩和绩点。

表 2.3.1 成绩与绩点折算对照表

成绩	<60	60~69	70~79	80~89	90~100
绩点	0	2	3	4	5

# 实验四 循环结构

## 一、实验目的

1. 掌握 for 语句的使用。
2. 掌握 while 语句的各种形式的使用。
3. 掌握如何控制循环条件，防止死循环或不循环。
4. 利用循环结构解决实际问题。

## 二、实验内容

1. 随机产生一个区间[5,10]内的整数 n，分别打印具有 n 行的有规律字符图，如图 2.4.1(a) 和 (b) 所示。

### 【提示】

- (1) 随机数产生要导入 random 库，调用 randint(a,b)函数；
- (2) 利用序列 range()函数，获得字母编码值，调用 chr()函数将字母编码值转换成字母；
- (3) 对于图 (b) 产生奇数随机数，利用 n=random.randrange(5,10,2)实现；每行字符不同可增加一个计数器变量，每输出一行加 1，再转换成字母的编码值。

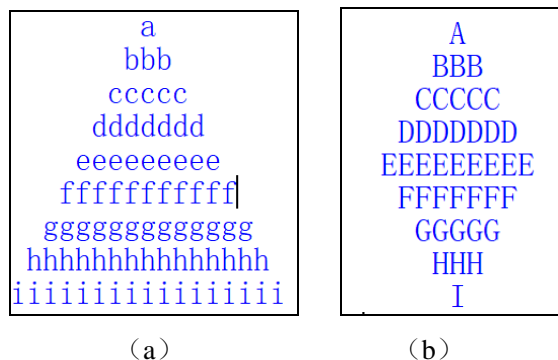


图 2.4.1 运行效果

【提示】序列为 range()函数，则要利用 chr()函数将字母编码值转换成字母。

2. 检查表达式输入中圆括号配对问题。要求对文本框输入的算术表达式，检验其圆括号配对情况，并给出相应信息如图 2.4.2 所示。

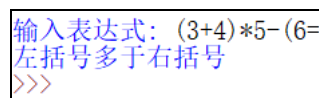


图 2.4.2 运行效果

【提示】利用 for 语句遍历整个表达式，对取出的每个字符判断，对于是左或右括号，进行相应的计数，所有字符都遍历了，根据计数确定表达式配对情况。

3. 输入一字符串，各单词间空格分隔，显示最长的单词和长度。

【提示】利用字符串的 split 方法，将单词分离到列表中；然后对列表遍历利用字符串 len() 函数求出最长单词。

4. 输入任意位的正整数，利用 while 语句将输入的数按逆序显示。即若输入 x=15356，则输出 nx=65351。

【提示】实现的方法是将一个十进制数 x 不断除以十取余、x 整除十，并将余数连接到逆序数中，直到 x 为 0。

5. 计算  $S = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{7} + \frac{1}{11} + \frac{1}{16} + \frac{1}{22} + \frac{1}{29} + \dots$  当第 i 项的值  $< 10^{-4}$  时结束。

【提示】找出规律，第 i 项的分母是前一项的分母加上表示有分母项开始计数，即分母通项为： $t_i = t_{i-1} + i$ 。

6. 计算  $\pi$  的近似值， $\pi$  的计算公式为：

$$\pi = 2 \times \frac{2^2}{1 \times 3} \times \frac{4^2}{3 \times 5} \times \frac{6^2}{5 \times 7} \times \dots \times \frac{(2 \times n)^2}{(2n-1) \times (2n+1)}$$

求 n=1000 时的结果，并与数学库提供的常数 pi 进行验证，为提高精度，如何设置 n 的值？

7. 编一程序，显示出所有的水仙花数。所谓水仙花数，是指一个 3 位数，其各位数字立方和等于该数字本身。例如，153 是水仙花数，因为  $153 = 1^3 + 5^3 + 3^3$ 。

【提示】解该题的方法有两种：

① 利用三重循环，将三个一位数连接成一个 3 位数进行判断。

② 利用单循环将一个 3 位数逐位分离后进行判断。

8. 求  $S_n = a + aa + aaa + \dots + \underbrace{aa \dots aaa}_n$  (n 个 a)，其中 a (1~9)、n (5~10) 的数，通过键盘输入。

例如：当输入 a=2，n=7 时，

显示： $S_n = 2 + 22 + 222 + 2222 + 22222 + 222222 + 2222222$

【提示】

① 为了得到不断重复 a 的 n 位的数 Temp，可用如下程序段实现：

```
temp=0
```

```
for i in range(1,n+1):
```

```
    temp=temp*10+a
```

② 若不光要显示计算的结果，还要显示产生的表达式，即：



$s=2+22+222+2222+22222+222222+2222222=2469134$

则如何实现？也就是在循环体内对字符串 ss 通过：`ss+=str(temp)+"`语句将表达式连接起来，出了循环可以在 `print()` 函数中使用转义字符 `"\b"` 去除最后一个 `"`。

9. 利用枚举法安排期末考试。期末考试在周一到周六的 6 天时间内要考 x、y、z 三门课程，考试课程顺序先考 x，后考 y，最后考 z，规定一天只能考一门，且 z 课程最早安排在周五考。编写程序列出满足条件的所有考试安排方案。

【提示】利用枚举法通过三重循环来解决，关键是每重循环的初值和终值。

10. 利用递推法求解猴子吃桃子问题。小猴在第一天摘了若干个桃子，当天吃掉一半多一个；第二天接着吃了剩下的桃子的一半多一个；以后每天都吃尚存桃子的一半多一个，到第 7 天早上要吃时只剩下一个了，问小猴最初共摘下了多少个桃子？

【提示】这是一个“递推”问题，先从最后一天推出倒数第二天的桃子，再从倒数第二天的桃子推出倒数第三天的桃子...

设第  $n$  天的桃子为  $x_n$ ，那么它是前一天的桃子数  $x_{n-1}$  的二分之一减去 1。

已知：当  $n=7$  第 7 天的桃子数为 1，则第 6 天的桃子由上面递推公式得 4 个，依次类

推，可求得第 1 天摘的桃子数。即： $x_n = \frac{1}{2}x_{n-1} - 1$  也就是： $x_{n-1} = (x_n + 1) \times 2$

11. 猜数游戏。计算机随机产生一个 1~100 之间的整数，由用户去猜，猜中即胜；猜不中，显示提示信息，继续猜，直至猜中，显示相关信息和次数，如图 2.4.3 所示。

```
输入猜测的数:50
50 小了
输入猜测的数:75
75 大了
输入猜测的数:62
62 小了
输入猜测的数:68
68 小了
输入猜测的数:71
71 大了
输入猜测的数:69
69 小了
输入猜测的数:70
70 恭喜你猜对了! 你猜了 7 次
```

图 2.4.3 运行界面

【提示】利用循环来解决，利用随机函数产生一个数，与用户输入的数进行比较，用户输入的数应利用二分法法则提高命中率。若最多猜 5 次，又如何实现？

# 实验五 函数

## 一、实验目的

1. 掌握用户函数的定义与调用
2. 理解位置参数、默认值参数
3. 掌握递归函数的定义和调用

## 二、实验内容

1. 写一个函数，求斐波那契数列的前  $n$  个数据，要求：返回值是由斐波那契数列的前  $n$  个数据的列表，调用该函数，并输出序列。

2. 编写判断  $m$  是否为素数的函数  $\text{fun}(m)$ ，若  $m$  是素数，则返回 `True`，否则返回 `False`，并利用它输出 2~100 之间所有的素数。

3. 编写函数  $\text{fun}(x)$ ，即给定正整数  $x$ ，返回其逆序数，例如 1234 的逆序数是 4321，用这个函数输出 1000~9999 之间所有的回文数。（回文数是指顺读和倒读都相同的数，如 5、151、3553 等）

提示：对给定的整数，可以通过不断与 10 相除取余获得其每个数字位，追加到一个列表中，然后将列表中的数字遍历，和值不断乘 10 加新项的方式实现逆序。

4. 随机产生 30 个成绩（0~100 之间）放入列表  $a$  中，设计一个函数  $\text{MyFun}$ ，将  $a$  传递给它，再为函数设定一个默认值参数  $\text{grade}=5$ 。 $\text{grade}$  传递 5,4,3,2,1，分别统计优、良、中、及格和不及格的人数。不传递值时，统计优秀者人数。请分别使用不指定关键字、指定关键字，不给默认值参数值三种方式调用函数。

5. 编一函数  $s(x)$ ，求级数和

$$s = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

当最后一项的绝对值小于  $10^{-6}$  时结束。

提示：请先推导和式的前后项的递推公式。

6. 编写二分法求高次方程根的函数，求解方程  $2x^3-4x^2+3x-6$  在区间  $[-10,10]$  之间的根。

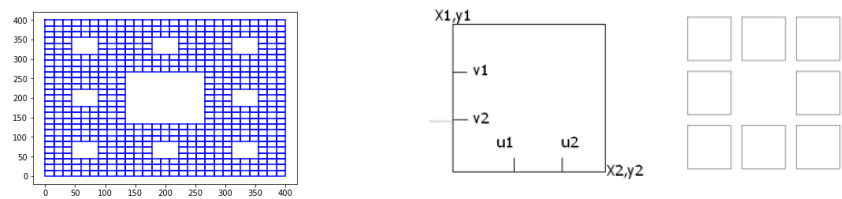
7. 编写函数返回 100-999 之间的所有水仙花数的列表。所谓水仙花数是指一个 3 位数，它的每个位上的数字的 3 次幂之和等于它本身。

8. 定义递归函数，给定整数  $n$ ，返回斐波那契数列的第  $n$  项，已知数列的前两项为 0,1

9. 将十进制的正整数  $d$  转换为  $r$  进制整数( $r=2-9$ )，可以分别取商  $Q=d/r$  和余数  $L=d \% r$ ，将  $Q$  再次与  $r$  求商取余，直到  $Q$  等于 0 结束。将每次求得的余数  $L$ ，按先后次序从右向左排列，就是转换得到的  $r$  进制的数。请用递归函数，实现将  $d$  转化为  $r$  进制的整数，以字符

串方式返回结果。

- 10. 编写递归函数，实现整数列表的逆转，并以 L[1,2,3]对其进行调用。
- 11. 请参照 4.5 节树形分型图，用递归制作如下的四边形分型图。



其原理为从 1 个大的四边形开始，每条边三等分，分成相同的 9 个四边形，然后除中间外的 8 个四边形重复上述过程，直到满足给定的条件底层为止（5 层）。  
调用设计的递归函数，给定最大四边形边长 500，递归 5 层结束