第3章 数据表示和计算

计算机的程序处理数据生成有用的信息，写程序就是描述数据的处理过程，其中必然涉及数据的表示和数据的计算。编程语言提供符号化的手段表示现实世界中的各种信息。例如：求三门课程成绩的平均分，四舍五入到整数，在 Python 中可以写出下面加粗部分表示的“表达式”。

```
>>> round((98 + 95.5 + 85)/3)
93
```

加粗斜体部分的表达式中包含了一些数据，如整数和实数等，还包含了可以对数值数据的计算操作加法“+”和除法“/”，计算三门课的平均分，数学函数 round 实现四舍五入。

要理解和正确地写出这个表达式，必须要知道 Python 语言对各种基本数据在写法上的规定，还需要了解 Python 语言如何表示各种基本数据的运算，以及计算的结果是什么，这些是本章讨论的主要问题。

在理解了基本数据之后，又如何在基本数据的基础上去组织文本数据和批量数据，以表示和处理更复杂的复合数据呢？

3.1 数据和数据类型的概念

3.1.1 数据的表示

在计算机的世界中，数据是对现实世界的抽象。使用计算机解决现实世界中的问题时，首先要分析有哪些信息是解决问题所需要的，并将它们表示成计算机能够接受的形式。所以使用计算机解决问题的第一步是挖掘具体数据对象与所解决的问题相关的信息，加以描述和表示，而忽略那些与所解决的问题无关的信息，这个过程就是抽象，抽象可以梳理计算问题所关注的主要数据并加以表示。程序中的数据一般会包括数值数据、文本数据和复合类数据。

假设为了预测 PM2.5 浓度的走势，需要记录所测得的 PM2.5 浓度值，一个测得的 PM2.5 浓度值为 60$\mu g/m^3$，在 Python 语言中可以用整数 60 来表示，也可以用浮点数 60.0 来表示。这一类的数据都会以数值数据来表示，以方便对 PM2.5 的浓度值进行数学计算。

又如，在一个英语学习的程序中，表示一个英语句子的方法可以是双引号括起来的字符串"my English words "，也可以用单引号括起来'my English words'。这一类的数据以文本数据来表示，可以对文本数据进行大小写转换，取子串等文本操作。

又如，在一个学籍管理的计算机信息系统中，需要表示现实中的学生，不可能把所有的学生信息都录入到系统中，如学生头发的长短、性格、着衣风格等信息就不是学籍管理所要关注的，学籍管理关注的是学生的学号、姓名、性别、出生日期、入学日期、专业等，学生的数据类型会抽象为(学号，姓名，性别，出生日期，入学日期，专业编码)，对应一个具体的学生王小明，他的数据为(10132110115，'王小明'，'男'，1993-2-18，2013-9-1，21601)，这就完成了从现实世界的王小明到计算机世界的王小明的抽象，如图 3-1-1 所示。

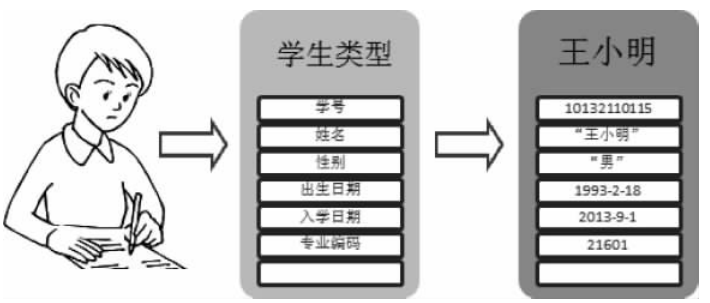

图 3-1-1　数据的抽象

3.1.2　数据类型的概念

1. 数据与数据类型的关系

计算机的世界是二进制的世界，0 和 1 描述着计算机世界中的指令、地址、数据等。数据是未经组织的要素，进入计算机后由程序处理得到相应的信息，计算机再以不同的形式展示这些信息(文字显示、多媒体呈现、打印、存储文件等)。数据进入到计算机并交由程序处理之前，先要存储在内存中，所以程序数据的表示与内存的组织方式密切相关。

内存单元的管理是以字节(8bit，一位 0 或 1 称为一个 bit)为单位进行分配和回收的。在内存中存储和处理数据与现实世界不同，是区分数据类型的，不同的数据类型占用不同的字节数，并有着不同的编码和运算机制。例如一个整数 20 要存入内存，整数在大多数计算环境中占 4 个字节，整数在硬件中的编码直接转化为二进制，所以在内存中存储的形式为 0000 0000 0000 0000 0000 0000 0001 0100，而不是 1 0100。可以这么说，数据类型决定程序数据的表示。

2. 数据类型的定义

数据类型是一组数据及在这组数据上的运算，它包含三方面的含义：

一是存储结构，一种数据类型的数据由几个字节构成，由存储的空间大小确定，它可以表示的数据范围也就确定了。也就是说，计算机数据表示是受限制的，没有数学中“无限”的概念。

二是存储机制，即如何解释比特流，各种数据类型的编码方式是怎样的？

三是运算，每种数据类型可以执行的运算有哪些？每一种数据类型有着不同的运算集，也就是对不同数据类型的数据的操作是不同的。同一运算符号，不同的数据类型也有着不同的解释。例如运算符“＋”，整数类型的解释是加法，3＋5 得到 8。

```
>>> 3 + 5
8
```

但文字类型的解释是连接,"3"+"5"得到"35"。

```
>>> "3" + "5"
'35'
```

综上所述,当数据具有以下相同的特性时就构成一类数据类型:

- 采用相同的书写形式。
- 在具体实现中采用同样的编码形式(内部的二进制编码)。
- 在内存存储中具有相同的二进制编码位数。
- 能做同样的运算操作。

一般来说,学习计算机解决实际问题要从学习数据类型入手,了解某一种编程语言提供了哪些数据类型支持对数据的表示。学习每一种数据类型时,要学习这种数据类型的 4 个特征,了解数据类型能表示怎样的数据,对这些数据能做怎样的操作。

3.1.3 Python 的内置类型

每一种编程语言都会预先设置一些基本的数据类型,称为内置(bulit-in)类型,并允许在内置类型的基础上构造更复杂的数据类型。

Python 的内置类型如图 3-1-2 所示,主要区分为简单类型和容器类型,简单类型主要是数值型数据,包括整型数据、浮点型数据、布尔类型数据和其他语言不多见的复数数据。容器类型可以应用于一次处理多个对象的场合,包括字符串 str、元组 tuple、列表 list、集合类型 set、字典类型 dict。

图 3-1-2 Python 的主要数据类型一览表

其中支持按给定顺序访问的对象称为序列对象,包括字符串 str、元组 tuple、列表 list。字符串,可方便地表示文本数据,元组和列表可以表示数据的序列。除序列对象之外的容器对象包括集合和字典。集合也可以表示数据的组合,但是其中的数据的存储不是连续有序的,与序列类型的区别在于不能按下标索引。字典是 Python 中唯一内置映射数据类型,字典元素由键(key)和值构成,可以通过指定的键从字典访问值。

3.1.4 常量和变量

1. 常量和变量定义

通常,程序中数据有两种表示方式:常量和变量。

例如整数 389,浮点数 23.56,字符串'hello',这些数据是不会改变的,也称为字面常量;在本章的后继小节对数据类型的具体介绍中将讲述每一种数据类型的常量的书写

形式。

变量描述的是存储空间的概念，将数据存储在内存中，用一个名称来访问内存空间，这个名称称为变量名。变量的值在程序运行的过程中是可以变化的。

【例 3-1-1】 将 3.1415926 存储在变量 a 中，显示 a 的值为 3.1415926，当再次将 3.1415 存储到变量 a 中，显示 a 的值为 3.1415。

```
>>> a = 3.1415926
>>> a
3.1415926
>>> a = 3.1415
>>> a
3.1415
>>>
```

2. 标识符和变量名

标识符是指在程序书写中程序员为一些特定对象的命名，包括变量名、函数名、类名、对象名等。Python 中的标识符由大小写英文字母、数字、下画线组成，以英文字母、下画线为首字符，长度任意，大小写敏感。

为了增加程序的可读性，通常使用有一定意义的标识符命名变量，但标识符不能与 Python 关键字同名。

Python 的关键字随版本不同有一些差异，可以按下面方法查阅，下面查阅的示例是 Python 3.3 版本的关键字，如图 3-1-3 所示。

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>help()
help> keywords
Here is a list of the Python keywords.  Enter any keyword to get more help.
False           def         if          raise       None        de
import          return      True        elif        in          try
and             else        is          while       as          except
lambda          with        assert      finally     nonlocal    yield
break           for         not         class       from        or
continue        global      pass
help>quit
```

图 3-1-3　Python 关键字

3. 变量的基本操作

(1) 变量的赋值

例 3-1-1 中出现的“=”不是数学中的等号，在程序语言中称为赋值运算符，赋值语句的语法格式为：

```
<变量> = <表达式>
```

赋值运算符的定义是将右边表达式的值赋给左边的变量,即将数据存储到变量所引用的内存空间中。

【例 3-1-2】 通过下面一组操作来理解变量的操作。

```
>>> x = 10
>>> y = 10 * x
>>> x = x + y
>>> x
110
>>> y
100
```

第一个表达式 x=10:将整数常量 10 赋值给变量 x;第二个表达式 y=10 * x:先从变量 x 中读取整数值 10,再参加乘法运算得到 100,将整数 100 赋值给变量 y;第三个表达式 x=x+y:先从变量 x 中读取整数值 10,再从变量 y 中读取整数值 100,再参加加法运算得到 110,最后将整数 110 赋值给变量 x。

赋值运算不难理解,但很容易与数学中的等号相混淆,例如在数学中下面的等式是正确的:

```
Y + 2 = X
```

读作:Y 加 2 等于 X。但根据赋值运算符的定义,在程序中显然是个错误的表达式,赋值运算符的左边必须是一个变量,用于接收右边表达式的值,不是等值的概念。这也就能够理解下面的式子:

```
X = X + 2
```

在程序中解释为先读取 X 的值,再参加加法运算,加 2 后的值,再重新赋值给 X。这样的一类计算过程中,执行对变量自身增值的操作,该变量称为累加器。

程序中使用“=”为变量赋值,注意与关系运算中的等于“==”区分。等号左边是变量名,右边可以是常量或表达式。例如以下都是有效的赋值语句:

```
X = 10    yϕ = 0.5    _ yes_no = True    number = 52.5 + x    Boy = 'boy'
```

与许多编程语言不同,在 Python 中,还允许同时对多个变量赋值,即所谓的“并行赋值”。

【例 3-1-3】 变量的并行计算。

```
>>> x,y,z = 3,4,5
>>> x,y,z
(3, 4, 5)
```

通过并行赋值能直接通过赋值语句交换两个变量的值,而不需要借助中间变量。

【例 3-1-4】 交换两个变量的值。

```
>>> x,y = y,x
>>> x,y
(4, 3)
```

对于常用的数学运算符(+、-、*、/、** 等),还可以使用增强的赋值运算符形式。例如 x=x+1,也可写成 x+=1 的形式,称为复合赋值运算。这种形式不仅表达更精练,由于在运行时仅需查询一次变量的值,因而执行速度也较快。

【例 3-1-5】 复合赋值运算符示例。

```
>>> x,y,z = 3,4,5
>>> x += 1
>>> y * = 2
>>> z ** = 3
>>> x,y,z
(4, 8, 125)
```

(2) 变量的读取

读取变量的方式就是将变量直接写在表达式中。计算过程中遇到变量,就会读取变量的值参加计算。

在表达式中可以反复读取一个变量的值。

【例 3-1-6】 求解三边为 3cm、4cm、5cm 三角形面积的表达式,其中求平方根的函数为 sqrt。

没有变量参与时可写出:

```
>>> import math
>>> math.sqrt( ((3 + 4 + 5)/2) * ( ((3 + 4 + 5)/2) - 3) * (((3 + 4 + 5)/2) - 4) * (((3 + 4 + 5)/
2) - 5) )
6.0
```

增加中间变量 s,表达式简短清楚多了:

```
>>> s = (3 + 4 + 5)/2
>>> math.sqrt( s * ( s - 3) * (s - 4) * (s - 5) )
6.0
```

s 通过计算获取了三条边之和的一半的值后,在计算面积的公式中可以反复地读取。

由于变量的值是可以变化的,表达式依赖变量的值就可以计算出不同的结果,与只包括常量的表达式有本质的不同。程序要解决的问题应有代表性,能够解决一类问题。

如果仅仅如上面求三边为 3cm、4cm、5cm 的三角形的面积,只需要一个计算器就能解决问题,编写一个程序,通常要解决的是求任意一个三角形的面积,这就存在一个数据建模的过程。

这个程序要解决的问题可描述为已知三角形的三条边,求三角形的面积。问题解决的方法这里采用的是通过三角形的面积公式求解。在求三角形的面积公式中,a、b、c 表示三条边,s 表示三条边之和的一半,area 表示三角形的面积。当 a、b、c 赋予不同的值时,就可以求解不同的三角形的面积。a、b、c、s、area 就是程序中解决问题所需的数据模型。

【例 3-1-7】 对求解三角形的过程再做如下修改,当对 a、b、c 赋予不同的值时,可以计算得到不同三角形的面积。

```
>>> import math
>>> a,b,c = 3,4,5
>>> s = (a + b + c)/2
```

```
>>> area = math.sqrt( s * ( s - a) * (s - b) * (s - c) )
>>> area
6.0
>>> a,b,c = 12,33,25
>>> s = (a + b + c)/2
>>> area = math.sqrt( s * ( s - a) * (s - b) * (s - c) )
>>> area
126.8857754044952
```

试想如果把这段代码写到一个程序,a、b、c 的取值可以在程序运行时从键盘输入,那么这个程序可以处理计算任意一个三角形的面积。

3.1.5 Python 的动态类型

从计算机硬件的角度考虑,数据是存储在内存的存储单元中,CPU 获取存储地址,访问内存的存储单元。

从程序的角度考虑,变量所引用的数据空间可视为一个容器,对应内存的存储单元,程序运行时,可以把数据存入到变量所引用的数据空间,也可以读取变量所引用的数据空间的值,每一个变量都有一个名字,程序中按名字访问变量,进行存取工作。

编译器在将高级语言翻译为机器语言时,将变量名对应到内存地址。

许多程序设计语言需要预先指定变量的数据类型,变量是区分不同数据类型的,不同的类型的变量中存储特定的数据类型的数据,数据类型确定了,一个变量对应内存的字节数,对应的编码形式和可参加的运算也就确定了。一旦存入的数据与预先声明的数据类型不一致时,程序就会出错,这种程序设计语言称为静态类型化的语言。

Python 语言使用“动态类型”技术,变量使用前不需要声明数据类型即可使用,然后根据变量中存放的数据不同,决定其数据类型。通过 type(<变量>)函数可以检测变量的数据类型。Python 程序在创建每一个数据对象时,给每一个数据对象设置一个对象 ID。使用函数 id(<变量>),可以得到对象的 ID。

【例 3-1-8】 动态类型示例。

当对 x 赋值整数 354 时,Python 在内存中创建整数对象 354,并使变量 x 指向这个数据对象,x 的类型为整型 int,此时 x 所指向的对象的 ID 为 34539888。

```
>>> x = 354
>>> type(x)
<class 'int'>
>>> id(x)
34539888
```

如再次对 x 赋值“word”时,Python 在内存中创建字符串对象“word”,并使变量 x 指向这个字符串数据对象,x 的类型变为字符串 str,x 所指向的对象的 ID 为 33407296,参见图 3-1-4。

```
>>> x = "word"
>>> type(x)
<class 'str'>
>>> id(x)
33407296
```

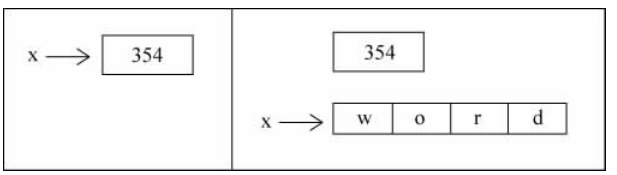

图 3-1-4　Python 的动态类型技术

也就是说，并不是 x 所代表的内存空间的内容发生了改变，而是 x 去指向了存储在其他内存空间的另一个对象。当 x 从整数对象 354 转向字符串对象“word”后，整数对象 354 没有变量引用它，它就成了某种意义上的“垃圾”，Python 会启动“垃圾回收”机制回收垃圾数据的内存单元，供其他数据使用。

读者可以自行思考例 3-1-1 中为变量 a 赋值的实质，请查看两次赋值变量 a 的 id 值。

3.2　数值数据的表示与计算

3.2.1　数值数据的常量表示

Python 的数值数据包括整型数据、浮点型数据、布尔类型数据和复数类型数据。

1. 整型数据 int

Python 的整数的大小只受机器的内存大小限制，默认情况下采用十进制，但也可采用其他进制形式。

例如：0o137(八进制的 95，o 表示八进制数)、0b1111 (二进制的 7，b 表示二进制数)、0xff(十六进制数的 255，x 表示十六进制数)，使用 type 函数可以获取数据的类型。

【例 3-2-1】 int 数据示例。

```
>>> 0o137
95
>>> 0b111
7
>>> 0xff
255
>>> type(28346283742874)
<class 'int'>
>>> type(0o137)
<class 'int'>
```

2. 浮点型数据 float

浮点类型数据支持小数形式表示和指数形式表示。

例如 12 是整数，12.0 就是浮点数，8.9e-4 表示 8.9×10^{-4}即 0.00089。计算机中的浮点数都是以近似值存储数据，Python 的 float 类型数通常可提供至多 17 个数字的精度，例如：print(23/1.05)显示的值为 21.904761904761905。

【例 3-2-2】 float 数据示例。

```
>>> type(12)
```

```
<class 'int'>
>>> type(12.0)
<class 'float'>
>>> 8.9e-4
0.00089
>>> type(1.2e1)
<class 'float'>
>>> 23/1.05
21.904761904761905
```

3. 布尔类型数据 bool

Python 的布尔类型数据只有两个：True 和 False，表示真和假。注意书写是要区分大小写。以真和假为值的表达式称为布尔表达式，用于表示某种条件是否成立，以支持选择控制和循环控制中必不可少的条件判断。

【例 3-2-3】 布尔数据示例。

```
>>> type(True)
<class 'bool'>
>>> x,y = 10,20
>>> x > y
False
>>> x + 10 <= y
True
```

4. 复数类型数据 complex

Python 提供复数类型数据也是它的一大特色。复数由实数部分和虚数部分构成，表示为：

```
real + imag(J/j 后缀)
```

实数部分和虚数部分都是浮点数。

【例 3-2-4】 复数的常用操作示例。

```
>>> aComplex = 4.23 + 8.5j
>>> aComplex
(4.2300000000000004 + 8.5j)
>>> aComplex.real              # num.real 返回复数的实数部分
4.2300000000000004
>>> aComplex.imag              # num.imag 返回复数的虚数部分
8.5
>>> aComplex.conjugate()       # num..conjugate() 返回复数的共轭复数
(4.2300000000000004 - 8.5j)
```

3.2.2 数值数据的计算

1. 表达式

表达式是数据对象和运算符按照一定的规则写出的式子，描述计算过程。例如算术表达式由计算对象、算术运算符及圆括号构成。最简单的表达式可以是一个常量或一个变量。

【例 3-2-5】 请列出计算半径为 4.5 的球的体积的表达式，math.pi 表示 π。

```
>>> 4 * (math.pi * 4.5 * 4.5 * 4.5)/3
381.7035074111598
```

数值数据可参与的运算包括算术运算、关系运算、逻辑运算，赋值运算，如表 3-2-1 所示。

表 3-2-1 数值对象的运算符

运 算 符	描 述
x+y，x-y	加、减
x * y，x/y，x//y，x%y，x ** y	相乘、相除、整除、求余、求乘方
<，<=，>，>=，==，!=	比较运算符
or，and，not	逻辑运算符
=，+=，-=，* =，/=，%=，** =	赋值运算，复合赋值运算符

2. 数值数据的运算

(1) 算术运算

Python 提供的算术运算包括加、减、乘、除和求余运算，与数学中的算术运算的定义基本相同，不同的地方是 Python 支持的除法区分为普通的除法和整除。

【例 3-2-6】 整数的除法和整除运算示例。

```
>>> x = 8
>>> y = 3
>>> x/y
2.6666666666666665
>>> x//y
2
```

【例 3-2-7】 浮点数的除法和整除运算示例。

```
>>> x = 3.8
>>> y = 0.7
>>> x/y
5.428571428571429
>>> x//y
5.0
```

% 为求余数的运算，可以通过求余运算来判断一个数是否能被另一个数整除。

【例 3-2-8】 判断一个数是否是偶数。

```
>>> x = 834
>>> x % 2 == 0
True
```

(2) 关系运算。

数值运算的关系表达式由数值数据和关系运算构成，得到的结果为布尔类型数据：True 或 False，一般形式为：

```
<数值 1><关系运算符><数值 2>
```

关系运算符包括<、<=、>、>=、==、!=,分别表示小于、小于等于、大于、大于等于、等于和不等于。其中要注意等于运算符“==”和赋值运算符“=”的区别,初学者常犯的错误就是以“等于”来表示“相等”的关系。

【例 3-2-9】 区别运算赋值“=”与关系运算相等“==”。

```
>>> 20 == 20
True
>>> 20 = 20
SyntaxError: can't assign to literal
>>> x,y = 10,20
>>> x == y
False
>>> x = y
>>> x
20
```

与其他高级语言不同,在 Python 中还允许使用级联比较形式,可用如下形式比较 a、b、c 三数的大小:a<=b<=c。

【例 3-2-10】 级联比较形式示例。

```
>>> a,b,c = 10,20,30
>>> a <= b <= c
True
```

对浮点数据进行相等的关系运算时,不能直接用等于“==”操作。浮点数类型能够表示巨大的数值,能够进行高精度的计算,但是由于浮点数在计算机内是用固定长度的二进制表示,有些数可能没有办法精确地表示,计算会引起误差。

【例 3-2-11】 浮点数的误差示例。

```
>>> x = 3.141592627
>>> x - 3.14
0.0015926269999999576
```

上例 x-3.14 的值并没有得到 0.001592627,结果略小一些,又如:

```
>>> 2.1 - 2.0
0.10000000000000009
```

这个例子中得到的结果又比正确的结果略大了一些。从这个例子可以得到一条经验:不能用==来判断是否相等,而是要检查两个浮点数的差值是否足够小,是则认为是相等的。

```
>>> 2.1 - 2.0 == 0.1
False
>>> esp = 0.000000001
>>> abs((2.1 - 2.0) - 0.1)< esp
True
```

(3) 逻辑运算

关系运算只能表示简单的布尔判断,复杂的布尔表达式还需要逻辑表达式来构成。逻

辑表达式通过逻辑运算与(and)、或(or)、非(not),可以将简单的布尔表达式连接起来,构成更为复杂的逻辑判断。

• 逻辑与 and

当计算 a and b 时,Python 会计算 a,如果 a 为假,则取 a 值,如果 a 为真,则 Python 会计算 b 且整个表达式会取 b 值。

• 逻辑或 or

当计算 a or b 时,Python 会计算 a,如果 a 为真,则整个表达式取 a 值,如果 a 为假,表达式将取 b 值。

• 逻辑非 not

如果表达式为真,not 为返回假,如果表达式为假,not 则返回真。

逻辑运算的真值表如表 3-2-2 所示。

表 3-2-2　逻辑运算的真值表

a	b	a and b	a or b	not a
False	True	False	True	True
False	False	False	False	True
True	True	True	True	False
True	False	False	True	False

【例 3-2-12】 判断某一年是否是闰年。

判断为闰年应满足下面两个条件之一:

• 该年能被 4 整除但不能被 100 整数。

• 该年能被 400 整除。

```
>>> y = 2010
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
False
>>> y = 2012                    # 符合第一个条件
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
True
>>> y = 2000                    # 符合第二个条件
>>> (y % 4 == 0 and y % 100!= 0) or(y % 400 == 0)
True
```

3. 表达式的求值

表达式的计算过程又称为表达式的求值。表达式可能很简单,也可能很复杂,其中包含了多个不同类型的运算符,那不同类型的运算符按照什么顺序运算呢?在数学表达式中的数据是不分类型的,都是数值,而计算机表达式中的数据区分不同的类型,同类型数据运算得到同类型的数据,那不同类型的数据出现在同一表达式中,如何运算?得到的是何种数据类型呢?毕竟不同数据类型数据的存储编码形式是不同的。这就涉及表达式的计算顺序和混合运算的类型转换问题。

(1) 计算顺序

影响表达式计算顺序的因素包括:运算符的优先级、运算符的结合方式和括号。

• 优先级

四则运算中先乘除后加减,也就是说乘除的优先级比加减要高,在计算中先做。程序设计语言会给每一个运算符确定一个优先级,具有较高优先级的运算符要比较低运算符优先计算。算术运算符的优先级设定与数学中基本相符。例如:下面表达式先计算 2 的平方,再求负数,而不是求－2 的平方,因为符号运算符的优先级比幂运算要低。

```
>>> -2**2
-4
```

数值数据常用运算符的优先级由高到低如表 3-2-3 所示。

表 3-2-3　数值数据常用运算符的优先级

序号	运　算　符	描　　述	序号	运　算　符	描　　述
1	＋x,－x	正,负	5	x<y,x<=y,x==y,x!=y,x>=y,x>y	比较
2	x**y	幂	6	not x	逻辑否
3	x*y,x/y,x%y	乘,除,取模	7	x and y	逻辑与
4	x＋y,x－y	加,减	8	x or y	逻辑或

• 结合方式

仅靠优先级,上面例子中 5*3/2 的计算方式还没有确定,因为相邻的乘运算和除运算的优先级是一样高的,结合方式或称结合律会规定具有相同优先级的运算符相邻出现时,运算符是从左向右结合,还是从右向左结合。例如,一目运算符是从右向左结合,－2＋3 中的负号是一目运算符,从右向左,操作数 2 为负数,二目运算符是从左向右结合,上式中的加号,从左向右结合。5*3/2 的计算方式可确定先乘后除。这一规定也符合数学中的习惯。

• 括号

括号可以突破计算的优先级,强制地规定计算顺序,括号括起部分的表达式会先行计算,计算的结果再参与括号外的表达式的计算。例如改写上例:12/(4＋5)*3/2,就可强制性先计算加法。

此外,运算对象还存在求值顺序问题。在一个较长的表达式中,不相邻的同级运算先算左边的对象还是右边的对象,例如:(34＋22)*(3＋5)式子中,两个括号应先行计算,但先计算(34＋22)还是先计算(3＋5),不同的程序语言,甚至不同的系统中有不同的规定。所以对运算对象的求值顺序可以这样理解:表达式的求值应不依赖于运算对象的求值顺序,因为无法保证它在各种系统中可能得到不同的顺序。运算对象求值顺序问题是程序语言中的特殊问题,在数学中不存在这种问题,这也是计算机与数学的不同。

(2) 类型转换

计算机中的运算与数据类型有密切关系,由于数据类型的限制,程序中一般同类型数据运算得到同类型的数据,例如 3＋4 得到 5,操作数和结果都是整数,但这一规则在下面式子的理解中会带来迷惑:

```
6/4*4
```

作为数学式子,结果很明显是 6。但在不同的程序设计语言中结果就不同了。在 C 语

言中结果为 4，在 Python 语言中结果为 6.0。如何解释呢？

C 语言严格遵循两个 int 类型数据计算，得到的还是 int 类型的原则，6/4 等于 1，1 乘以 4 等于 4。Python 语言将参加除法运算的操作数自动转化为 float 类型，再进行 float 类型的除法运算，6.0/4.0 等于 1.5，1.5 乘以 4.0 等于 6.0。

```
>>> 6/4 * 4
6.0
```

如果表达式中进行的是整除运算，得到的结果就与 C 语言中的相同了。

```
>>> 6//4 * 4
4
```

如果表达式中包含不同数据类型的数据对象，就出现了混合运算，任何运算都是定义于相同数据类型的，不同类型的数据运算要进行类型转换，只有同类型的数据对象才能进行运算。

表达式计算中碰到不同的数据类型例如 3.0+2，先通过转换将 2 转换为 2.0，然后才会进行实际的浮点数运算，这种转换是系统自动完成的，不需要在程序中写出，所以称为自动转换或隐式转换。

【例 3-2-13】 自动转换示例。

```
>>> 3.0 + 2
5.0
>>> type(3.0 + 2)
<class 'float'>
```

自动转换的基本原则是将表示数值范围小的数据类型的值转换到表示数值范围大的数据类型的值，这样能避免由于类型转换造成的误差损失。

如果自动转化不符合需求，程序语言还引入了**强制转换机制**或**显式转换**，在程序语句中明确类型转换的描述，要求执行类型转换。强制转换的出现形式有强制转换的运算，例如要将实数 3.14 转化为整数，Python 语言提供各种类型的转换函数，上式写作：int(3.14)。常用类型转换函数如表 3-2-4 所示。

表 3-2-4 常用类型转换函数

函　数	描　述	函　数	描　述
int(x[,base])	将 x 转换为一个整数	ord(x)	将一个字符转换为它的 ASCII 编码的整数值
float(x)	将 x 转换为一个浮点数	chr(x)	将一个整数转换为一个字符，整数为字符的 ASCII 编码
complex(real [,imag])	创建一个复数	hex(x)	将一个整数转换为一个十六进制字符串
str(x)	将对象 x 转换为字符串	oct(x)	将一个整数转换为一个八进制字符串
repr(x)	将对象 x 转换为字符串	eval(str)	将字符串 str 当成有效表达式来求值，并返回计算结果

【例 3-2-14】 显式转换示例。

```
>>> x,y = 23,12                    # 变量 x,y 的值为整数 23 和 12
>>> y = float(y) + 0.5             # 强制转换 y 的值为 12.0 参加浮点运算
>>> y
12.5
>>> complex(x,y)                   # 创建复数,x,y 为实部和虚部的值
(23 + 12.5j)
>>> str(x)                         # 读取 x 的值转化为字符串,存储在 x 中的值不变
'23'
>>> hex(x)                         # 读取 x 的值转化为十六进制字符串,存储在 x 中的值不变
'0x17'
>>> oct(x)                         # 读取 x 的值转化为八进制字符串,存储在 x 中的值不变
'0o27'
>>> repr(x + 20)                   # 读取 x 的值加 20 后转化为字符串,存储在 x 中的值不变
'43'
>>> chr(13)                        # 得到 13 所表示的字符: 回车
'\r'
>>> ord('\n')                      # 得到换行符的 ASCII 值
10
>>> eval('x - y')                  # 计算字符串表示的表达式的值
10.5
```

3.2.3 系统函数

除了使用运算符进行运算,一般的高级语言程序系统中都提供系统函数丰富语言功能。Python 提供模块的方式,可方便地扩充语言的功能。Python 的系统函数由标准库中的很多模块提供。标准库中的模块,又分成内置模块和非内置模块,内置模块__builtin__中的函数和变量可以直接使用,非内置模块要先导入模块,再使用。

1. 内置模块

Python 中的内置函数是通过__builtin__模块提供的,该模块不需手动导入,启动 Python 时系统会自动导入,任何程序都可以直接使用它们。该模块定义了一些软件开发中常用的函数,这些函数实现了数据类型的转换、数据的计算、序列的处理、常用字符串处理等。

函数的调用方式与数学函数类似,函数名加上相应的参数值,多个参数值之间以逗号分隔。

```
<函数名>(参数序列)
```

【例 3-2-15】 内置模块函数示例。

```
>>>### help(obj) 在线帮助, obj 可是任何类型,例如查看 math 模块的内容
>>> help(math)


>>> ## int("123")可将字符串"123"转换为整数 123
>>> int("123")
```

```
123
>>> ## int(78.9)得到整数 78(去掉尾部小数)
>>> int(78.9)
78
>>>## reper(obj),将任意值转为字符串,常用于构造输出字符串
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print( s )
The value of x is 32.5, and y is 40000...
>>> ##使用 round(x,n)可按"四舍五入"法对 x 保留 n 位小数
>>> round(78.3456,2)
78.35
>>>##使用 len(s)计算字符串的长度
>>> len("Good morning")
12
```

2. 非内置模块

(1) 非内置模块的导入

非内置模块在使用前要先导入模块,Python 中使用如下语句来导入模块:

```
import <模块名>
```

其中模块名也可以有多个,多个模块之间用逗号分隔。该语句通常放在程序的开始部分。模块导入后,可以在程序中使用模块中定义的函数或常量值:

```
<模块名>.<函数>(<参数>)
<模块名>.<字面常量>
```

【例 3-2-16】 导入数学库示例。

```
>>> import math                ##导入数学库
>>> math.pi                    ##查看圆周率 π 常数
3.141592653589793
>>> math.pow(math.pi,2)        ##函数 pow(x,y):求 x 的 y 次方
9.869604401089358
>>>                    ##计算边长为 8.3 和 10.58,两边夹角为 37 度的三角形的面积的表达式为:
>>> 8.3 * 10.58 * math.sin(37.0/180 * math.pi)/2
26.423892221536985
```

数学库中函数引入和使用的另外一种方式如例 3-2-17。

【例 3-2-17】 数学库中函数引入和使用的另外一种方式。

```
>>> from math import sqrt      #引入数学库中的 sqrt 函数
>>> sqrt(16)
4.0
```

如果希望导入 math 模块中所有的函数定义,而非仅仅是 sqrt 函数可以使用以下形式:

```
>>> from math import *          #引入数学库中所有的函数
>>> sqrt(16)
4.0
```


注意：引入方式不同，对应的函数的使用方式也不同，还要注意所引入模块中的函数名等与现有系统中不产生冲突。

(2) 常用的标准数学函数

常用的标准数学函数包括三角函数、反三角函数、指数函数、对数函数，平方根函数、绝对值函数和乘幂函数，如表 3-2-5 所示。

表 3-2-5　math 库中的常用函数和字面常量

常 用 函 数	描　　述	常 用 函 数	描　　述
pi	常数 π(近似值)	sin(x)	正弦函数
e	常数 e(近似值)	cos(x)	余弦函数
fabs	求绝对值	tan(x)	正切函数
trunc(x)	舍去一个浮点数的小数部分	ceil(x)	大于等于 x 的最小整数
factorial(x)	求 x 的阶乘	floor(x)	小于等于 x 的最大整数
pow(x,y)	求 x 的 y 次方	sqrt(x)	求 x 的平方根

3.3　文本数据的表示和操作

计算机早期是应科学计算的需求而产生的，程序的处理对象是数值型的数据。随着计算机的应用在各行各业的普及，程序的处理对象也日益丰富，大量应用于文本数据的处理，例如各类信息管理系统、文本编辑器、电子出版物、搜索引擎等。文本数据在程序中通常是以字符串类型表示的，字符串由字符构成。

Python 语言表示字符串的数据类型是 str 类，str 类定义了字符串类型的常量表示、基本运算和操作方法。

```
>>> type("shanghai")
<class 'str'>
```

3.3.1　文本的表示

1. 字符

计算机中表示文本的最基本的单位是字符，包括可打印字符和不可打印的控制字符：可打印字符包括：

(1) 英文的大小写字母 a～z，A～Z；

(2) 数字字符 0～9；

(3) 标点符号和一些键盘上的常见符号。

控制字符包括回车、制表符、退格等，在程序中需要以转义字符表示这些控制字符。Python 中的转义字符以“\”为前缀，如表 3-3-1 所示。

表 3-3-1　Python 的转义字符

转义字符	描　述	转义字符	描　述
\\	反斜杠符号	\t	横向制表符
\'	单引号	\r	回车
\"	双引号	\n	换行
\a	响铃	\(在行尾时)	续行符
\b	退格(Backspace)	\f	换页
\e	转义	\oyy	八进制数 yy 代表的字符，例如：\o12 代表换行
\000	空	\xyy	十六进制数 yy 代表的字符，例如：\x0a 代表换行

2. 字符串常量

字符串可以使用单引号、双引号、三引号封装，但前后必须一致。

【例 3-3-1】 字符串常量表示。

"hello"和'hello'表示的都是字符串 hello。

```
>>> print( "hello")
hello
>>> print( 'hello')
```

如果字符串本身要带单引号或双引号，可以用不同的引号嵌套表示。

```
>>> print('"hello"')
"hello"
>>> print("'hello'")
'hello'
```

【例 3-3-2】 多行字符串常量示例。

```
>>> print('''
hello'
world"
''')

hello'
world"
```

Python 同样支持以“\”为前缀的转义字符，例如使用转义字符“\n”可以在输出时使字符串换行。

```
>>> print ("hello everyone\ntoday is a great day!")
hello everyone
today is a great day!
```

等价于

```
>>> print ('''hello everyone
today is a great day! ''')
```

Python 还支持只有引号的空字符串。

```
>>> ""
''
```


3. 字符串变量

字符串同样也可以使用字符串变量来操作。

【例 3-3-3】 字符串变量示例。

```
>>> s = "hello"
>>> print(s)
hello
```

3.3.2 字符串类型数据的基本计算

1. 连接和复制操作

字符串类型支持的运算有＋和＊,可以使用＋连接两个字符串。

【例 3-3-4】 连接运算示例。

```
>>> 'shang' + 'hai'
'shanghai'
```

【例 3-3-5】 复制运算示例。

运算＊可以生成重复字符串,用法是[字符串]＊[整数],非常方便,例如:

```
>>> "hi " * 5
'hi hi hi hi hi '
>>> s = "hi"
>>> t = s * 3
>>> print(t)
hihihi
```

2. 索引操作

使用下标值来获取字符串中指定的某个字符,称为索引操作,下标是一个整数值,可以是整数常量,整数变量,也可以是一个整数表达式,用法是:

```
<字符串>[下标]
```

【例 3-3-6】 字符串下标示例。

```
>>> "Student"[5]
'n'
>>> s = "hello Python!"
>>> s[0]
'h'
>>> i = 10
>>> s[i + 1]
'n'
>>> s[ - 1]
'!'
```

注意:Python 中下标位置是从 0 开始计数的,数值表达式可以为负数,则表示从右向左计数,字符串中的字符引用,s 的合法下标标识如图 3-3-1 所示。

Python 不支持以任何方式改变字符串类型对象的值,不能通过下标的方式来改变字符

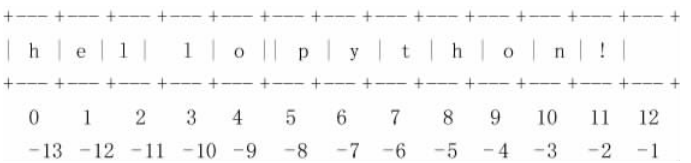

图 3-3-1　字符串的下标示例

串中的某一个字符。

【例 3-3-7】 字符串修改错误示例。

```
>>> s[5] = 'i'
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
s[5] = 'i'
TypeError: 'str' object does not support item assignment
```

出错提示给出类型错误：str 对象不支持对其成员赋值。

Python 还支持通过索引操作获取字符串的子串，也称为切片操作，用法是指定子串的区间，start 表示开始位置，end 表示结束位置(下标+1)。

```
<字符串>[start: end]
```

【例 3-3-8】 子串索引示例。

```
>>> s[0:2]
'he'
>>> s[2:4]
'll'
>>> s[:2]                    # 前面的两个字符
'He'
>>> s[2:]                    # 除了开始两个字符的所有字符
'llo Python'
```

3. 子串测试操作

子串测试操作 in 可以测试一个子串是否存在于一个字符串中，计算返回布尔值，用法为：

```
<子串> in <字符串>
```

【例 3-3-9】 子串测试操作示例。

```
>>> 'py' in s
True
>>> t = 'the'
>>> t in s
False
```

3.3.3　str 对象的方法

str 类提供了丰富的字符串操作的方法，如表 3-3-2 所示，s 表示一个 str 对象。读者同样可以通过 help(str)查询更多的字符串操作的方法。

表 3-3-2　str 对象的常用方法

str 的常用方法	描　　述
s. capitalize()	返回首字符大写后的字符串,s 对象不变
s. lower()	返回所有字符改小写后的字符串,s 对象不变
s. upper()	返回所有字符改大写后的字符串,s 对象不变
s. strip()	返回删去前后空格后的字符串,s 对象不变
s. replace(old,new)	将 s 对象中所有的 old 子串用 new 子串代替
s. count(sub[,start[,end]])	计算子串 sub 在 s 对象中出现的次数,start 和 end 定义起始位置
s. find(sub[,start[,end]])	计算子串 sub 在 s 对象中首次出现的位置
s. join(iterable)	将序列对象中所有字符串合并成一个字符串,s 对象为连接分隔符
s. split(sep=None)	将 s 对象按分隔符 sep 拆分为字符串列表,默认为空格

str 对象方法的调用形式为:

<字符串>. 方法名(<参数>)

如图 3-3-2 所示,在命令行提示符后输入对象名,稍作停留,会显示该对象的所有方法的列表,使用上下光标键可以选择所需的方法。

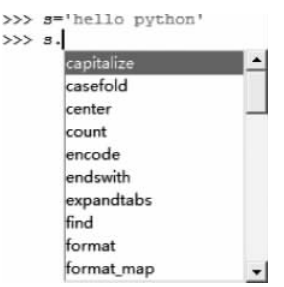

图 3-3-2　弹出的对象的方法列表示例

【例 3-3-10】 str 对象方法示例。

```
>>> s.find('he')              #求子串'he'第一次出现的位置
0
>>> s.count('h')              #求'h'出现的次数
2
```

同样,使用 str 类的方法不能改变字符串对象的值,例如调用 strip 函数去除字符串的前后空格,它的作用是返回一个去除了原字符串的前后空格的新串。

```
>>> s = '    hello Python '
>>> t = s.strip()
>>> s
'    hello Python '
>>> t
'hello Python'
```

同样 lower、upper、replace 等会产生字符修改的函数,都是返回一个新的字符串对象,而字符串对象本身的内容是不变的。

【例 3-3-11】 字符串对象的连接和分裂操作示例。

```
>>> a = ["hello","Python" ]     #a 为一个包含两个字符串的列表
>>> b = " ".join(a)             #将 a 中的两个字符串连接,并以空格分隔,赋值给 b
>>> b
'hello Python'
>>> b.split()
['hello', 'Python']
>>> c = ",".join(a)
>>> c
```

```
'hello,Python'
>>> c.split(',')
['hello', 'Python']
```

3.4 批量数据表示与操作

3.4.1 批量数据的构造

1. 批量数据的概念

在实际的计算机处理问题中，程序要处理的对象都是大批量的相同数据类型的数据集合，例如一次科学实验中获得的大量实验数据，关键字搜索时大量的网页中所包含的单词，一幅 BMP 图像中包含的像素点。这几个例子中分别包含大量数值的集合、大量文本的集合和大量的点对象的集合。程序语言支持批量数据的存储，用统一的名称管理一批数据，在内存的存储上表现为存储的空间是连续的。

对批量数据中元素的访问可通过下标。例如：a[1]，a[i]。下标的含义：与第一个元素的偏移量，通常从 0 开始。

例如有：

```
Color = ("red","green","blue")
```

则：Color[0]的值是"red"，Color[1]的值是"green"，Color[2]的值是"blue"，内存示意如图 3-4-1 所示。

批量数据的存储与单变量数据存储相比的优势在于：

(1) 一批批量数据只需定义一个名称，程序的通用性更强。一个单变量只可以控制一个数据，使用单变量，程序可控制的数据的个数是固定的。

(2) 使用方便，可以组织循环控制结构，通过控制下标的值控制一批数据。

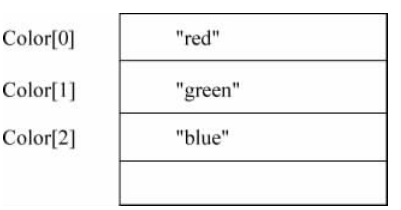

图 3-4-1 Color 值的内存示意图

大多数程序设计语言提供了数组来组织批量数据的存储与操作，一个数组中存储着具有相同数据类型的数据，通过下标引用其中的每一个数组元素，数组元素可以实现该数据类型所定义的运算。面向对象的程序语言还提供了功能更为强大的列表类、向量类等，在定义批量对象的存储之外，同时提供对批量数据的常规操作。

2. Python 对批量数据的表示和操作

Python 可支持批量数据存储和操作的组合数据类型有列表(list)、元组(tuple)、字典(dict)、集合(set)等。字符串也可以看作字符的组合类型。

这些数据类型按数据是否是在内存中连续排列的集合体可区分为两种方式：有序的数据集合体和无序的数据集合体。

有序的数据集合体，也称为序列，包括字符串、元组和列表。序列的数据成员之间存在排列次序，因此可以通过各数据成员在序列中所处的位置，即索引或下标来访问数据成员。Python 提供序列的一些通用操作，如表 3-4-1 所示，以实现对序列的索引、连接、复制、检测

成员等。

表 3-4-1　序列的基本操作

操　　作	描　　述
x1＋x2	连接序列 x1 和 x2,生成新序列
x＊n	将序列 x 复制 n 次,生成新序列
x[i]	引用序列 x 中下标为 i 的序列成员,i 从 0 开始计数
x[i:j]	引用序列 x 中下标从 i 到 j－1 的子序列
x[i:j:k]	引用序列 x 中下标从 i 到 j－1,间隔 k 的子序列
len(x)	计算序列 x 中成员的个数
max(x)	序列 x 中最大数据项
min(x)	序列 x 中最大数据项
v in x	检测 v 是否存在序列 x 中,返回布尔值
v not in x	检测 v 是否不存在序列 x 中,返回布尔值

无序的数据集合体包括集合、字典等,无序的数据集合体中的数据成员之间不存在存储的先后关系,故也不支持索引操作。

3. 类型构造器

在 Python 语言中,所有事物都是程序可以访问的对象,所有表示数据的类型都是类(class),包括简单数据类型如 int、float、bool、complex,以及 Python 的组合数据类型,列表(list)、元组(tuple)、字典(dict)、集合(set)。这一点可以从 type 函数示例中看到。

每一个类都提供了类型构造器,类型构造器是一个与类同名的函数。例如表 3-2-4 中列出的 int()、float()、str()都是类型构造器,它们通常可以生成一个空的对象,将一个对象转换为本类对象。同样本节涉及的容器类型对象都有各自的类型构造器函数,完成创建对象、转换对象的功能。类型构造器的工作原理将在第 8 章介绍。

【例 3-4-1】 类型构造器示例。

生成空字符串对象:

```
>>> s1 = str()
>>> s1
''
```

将一个整数学号转化为字符串:

```
>>> s2 = str(101030311245)
>>> s2
'101030311245'
```

3.4.2　元组和列表

Python 中的元组和列表是可以存储任意数量的一组相关数据,形成一个整体。其中的每一项可以是任意数据类型的数据项。各数据项之间按索引号排列并允许按索引号访问。

元组和列表的区别为:元组的数据项是不可变的,创建之后就不能改变其数据项,这点与字符串是相同的;而列表是可变的,创建后允许修改、插入或删除其中的数据项。

1. 元组的基本操作

(1) 元组的字面表示

元组一般使用圆括号来表示,数组项之间用逗号分隔。

【例 3-4-2】 字面表示方式创建元组。

```
>>> t = 1,2,3
>>> t
(1, 2, 3)
>>> t1 = (1,2,3)
>>> t1
(1, 2, 3)
```

数据项可以是相同数据类型的,也可以是不同数据类型的:

```
>>> t2 = "east","south","west","north"
>>> t2
('east', 'south', 'west', 'north')
>>> t3 = "0010110","张山","men",18
>>> t3
('0010110', '张山', 'men', 18)
```

可以定义空的元组,也可以定义嵌套的元组:

```
>>> t4 = ()
>>> t4
()
>>> t5 = 23,(5,8,6),18,6
>>> t5
(23, (5, 8, 6), 18, 6)
>>> t6 = t1,t2,t3,t4,t5
>>> t6
((1, 2, 3), ('east', 'south', 'west', 'north'), ('0010110', '张山', 'men', 18), (), (23, (5, 8,
6), 18, 6))
```

(2) 元组的类型构造器。

元组的类型构造器 tuple()可以生成一个空的元组,也可以将字符串、列表、集合等转化为元组。可以想象,容器类型对象可以通过它们的类型构造器相互转化。

【例 3-4-3】 元组的类型构造器示例。

生成一个空的元组:

```
>>> t7 = tuple()
>>> t7
()
```

将一个字符串转化为元组:

```
>>> t7 = tuple('Python')
>>> t7
('P', 'y', 't', 'h', 'o', 'n')
```

(3) 元组元素的访问。

可以通过下标访问元组中的某一项,称为元组元素。同样下标从 0 开始,但不能修改元组中的元素。

【例 3-4-4】 元组元素的访问示例。

```
>>> t6[2]
('0010110', '张山', 'men', 18)
>>> t6[2][0]
'0010110'
>>> t6[2] = t2
Traceback (most recent call last):
  File "< pyshell # 60 >", line 1, in < module >
    t6[2] = t2
TypeError: 'tuple' object does not support item assignment
```

2. 列表的基本操作

(1) 列表的字面表示。

列表的创建与元组的区别在于需要使用方括号。其他与元组类似,数据项之间以逗号分隔,可以嵌套定义,可以是不同的数据类型,可以是空列表,可以用下标访问其中的数据项。

【例 3-4-5】 字面表示方式创建元组。

```
>>> L1 = ["one","two","three","four","five"]   # 由 5 个字符串构成的列表
>>> L2 = [[1,2],[3,4],[5,6]]                   # 由 3 个列表构成的嵌套列表
>>> L3 = ["zhangsan",True,185,"lisi",False,165,"wanger",True,176]          # 混合类型的列表
>>> L4 = [[],[],[]]                            # 嵌套空列表的列表
>>> L5 = []                                    # 生成空的列表
```

(2) 列表的类型构造器。

列表的类型构造器 list()可以生成一个空的列表,也可以将字符串、元组、集合等转化为元组。

【例 3-4-6】 列表的类型构造器示例。

```
>>> L5 = list()                                # 生成空的列表,与 L5 = []功能相同
>>> L6 = list('Pyhton')                        # 将一个字符串对象转化为列表
>>> L6
['P', 'y', 'h', 't', 'o', 'n']
>>> L7 = list(('he','her','here'))             # 将一个元组转化为列表
>>> L7
['he', 'her', 'here']
```

(3) 列表元素的访问。

列表同样支持索引访问,访问特定列表元素或是子列表,修改列表元素。

【例 3-4-7】 列表元素的访问。

```
>>> L1[1]
'two'
>>> L2[2][1]
6
```

列表和元组根本区别在于可以改变列表中的元素。

【例 3-4-8】 修改指定位置的元素。

```
>>> L2[2] = 5
>>> L2
[[1, 2], [3, 4], 5]
>>> L2[0][0] = L2[0][1] * 10
>>> L2
[[20, 2], [3, 4], 5]
```

【例 3-4-9】 连接列表元素。

```
>>> L2 = L2 + [7,8]                                    #将两个列表的表项连接为一个列表
>>> L2
[[20, 2], [3, 4], 5, 7, 8]
>>> L2 = L2 + [[9,10]]
>>> L2
[[20, 2], [3, 4], 5, 7, 8, [9, 10]]
```

【例 3-4-10】 在指定位置插入数据项。

```
>>> L2[3:3] = [6]                          #3:3 表示下标为 3 的位置,在此位置插入列表[6]的表项 6
>>> L2
[[20, 2], [3, 4], 5, 6, 7, 8, [9, 10]]
```

【例 3-4-11】 删除指定位置的数据项。

```
>>> L2[2:6] = []                       #引用 L2 中 2 到 5 的表项,将子序列通过赋值操作更改为空序列
>>> L2
[[20, 2], [3, 4], [9, 10]]
```

【例 3-4-12】 在指定位置插入嵌套数据项。

```
>>> L2[2:2] = [[5,6],[7,8]]
>>> L2
[[20, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
```

【例 3-4-13】 可以使用 Python 内置函数 len()来计算元组或列表的长度。

```
>>> len(L2)
5
```

【例 3-4-14】 使用 in 和 not in 来测试是否是元组或列表成员,测试结果返回布尔值(True 或 False)。

```
>>> 'wanger' in L3
True
>>> t2 in t6
True
>>> 7 in L2
False
>>> [7,8]in L2
True
```

3. 元组对象方法和列表对象的方法

由于元组对象创建后不能改变自身的值,是只读属性的对象,它的方法只有两个,如表 3-4-2 所示,T 表示一个元组对象。

表 3-4-2 元组对象的常用方法

方 法	描 述
T. count(value)	计算 value 值在元组中出现的次数
T. index(value)	计算 value 值在元组中出现的下标

相对于元组对象的方法,列表的方法就丰富得多,如表 3-4-3 所示,L 表示一个列表对象。

表 3-4-3 列表对象的常用方法

方 法	描 述
L. append(object)	在列表 L 尾部追加对象
L. clear()	移除列表 L 中的所有对象
L. count(value)	计算 value 在列表 L 中出现的次数
L. copy()	返回 L 的备份的新对象
L. extend(Lb)	将 Lb 的表项扩充到 L 中
L. index(value, [start, [stop]])	计算 value 在列表 L 指定区间第一次出现的下标值
L. insert(index, object)	在列表 L 的下标为 index 的表项前插入对象
L. pop([index])	返回并移除下标为 index 的表项,默认最后一个
L. remove(value)	移除第一个值为 value 的表项
L. reverse()	倒置列表 L
L. sort()	对列表中的数值按从低到高的顺序排序

使用列表对象的方法,可以很方便地存储、维护、分析批量数据。

【例 3-4-15】 对某居民家庭一年的用电情况进行维护和分析。

(1) 初始化列表:

```
>>> t = []
```

(2) 增加 1 月份的用电量:

```
>>> t.append(271)
```

(3) 批量增加 2 月份到 12 月份的用电量:

```
>>> t.extend ([151,78,92,83,134,357,421,210,88,92,135])
>>> t
[271, 151, 78, 92, 83, 134, 357, 421, 210, 88, 92, 135]
```

注意:append 方法是将一个对象追加到列表中,append 方法的参数是一个任意对象,作为一个表项加入到列表中;extend 方法是将一个列表中的表项扩充到列表中去,所以 extend 方法的参数是一个列表,参数列表的表项加入到列表中。

(4) 修改 8 月份的用电量 421 为 425：

```
>>> t.remove(421)
>>> t.insert(7,425)
>>> t
[271, 151, 78, 92, 83, 134, 357, 425, 210, 88, 92, 135]
```

注意：修改列表表项的方法还可以直接通过索引访问：t[8]＝425，更为方便。在此只为演示 move 和 insert 方法的使用。

(5) 求用电量最高的月份 maxm。

解题思路：先计算 t 中的最大值，再寻找最大值在 t 中出现的位置，下标从 0 开始，加 1 就是对应的月份。求最大值使用 max 函数实现，寻找一个数值在列表中出现的位置，使用 index 方法实现：

```
>>> maxm = t.index(max(t)) + 1
>>> maxm
8
```

(6) 按用电量从低到高排序：

```
>>> s = t.copy()
>>> s.sort()
>>> s
[78, 83, 88, 92, 92, 134, 135, 151, 210, 271, 357, 425]
```

注意：这里不能直接 s＝t，而是要使用 copy 函数得到一个副本对象。s＝t 的含义是 s 指向 t 对象，那么对 s 排序等同于对 t 排序了。如果想得到从高到低的排序结果，可以增加一个参数值设定：s.sort(reverse＝True)。

(7) 找出用电量最高的三个月。

解题思路：s 序列是 t 序列的从低到高的有序序列，倒序后，序列值从高到低排列，序列的前三项就是用电量最高的三个值。再寻找三个值在源序列 t 中出现的位置，加 1 后就可以计算出月份：

```
>>> s.reverse()
>>> m1,m2,m3 = t.index(s[0]) + 1,t.index(s[1]) + 1,t.index(s[2]) + 1
>>> print(m1,m2,m3)
8 7 1
```

结合下一章所讲的控制结构，可以对居民家庭的用电情况做更为复杂的分析统计。

3.4.3 集合和字典

1. 集合

Python 的集合也是一个内置的数据类型，与列表和元组不同的是集合是无序的，而且集合的元素不能重复出现，不能通过数字进行索引。正是因为它具有的这些特性，所以通常可用来进行一些数据中转处理，例如去除列表中的重复元素（集体元素是唯一的），两个列表的相同元素（交集）等。

(1) 创建集合。

Python 的集合可分为可变集合(set)和不可变集合(frozenset)。对可变集合(set),可以添加和删除元素,对不可变集合(frozenset)则不允许这样做。可变集合可以通过集合标识符{}直接创建,也可以通过类型构造器 set()创建,不可变集合需要通过类型构造器 frozenset()创建。

使用{}创建的是可变集合 set,{}中用逗号分隔的数据项作为集合的一个元素。

【例 3-4-16】 集合的字面表示示例。

```
>>> s1 = {2,4,6,8,10}
>>> type(s1)
<class 'set'>
>>> s1
{8, 10, 4, 2, 6}
>>> s2 = {'hello'}
>>> s2
{'hello'}
```

set 函数的参数是容器对象,可以是字符串、列表和元组,它可以将序列的数据元素作为集合 set 的元素。

【例 3-4-17】 集合的类型构造器示例。

```
>>> s3 = set('hello')
>>> s3
{'l', 'e', 'o', 'h'}
```

字符串“hello”由 5 个字符构成,其中'l'出现了两次,转换到集合中,重复项只能保留一个,且字符次序与原字符串的次序不同。集合的这种特性,可以很方便地对列表对象执行去重复的功能。同样还可以根据列表对象来创建集合:

```
>>> s5 = set(['he','hello','her','here'])
>>> s5
{'here', 'hello', 'he', 'her'}
```

【例 3-4-18】 列表去重复操作示例。

通过 set 函数建立列表的去重复集合元素,再通过 list 方法根据集合创建列表:

```
>>> L1 = [1,2,3,4,1,2,3,4]
>>> s4 = set(L1)
>>> s4
{1, 2, 3, 4}
>>> L2 = list(s4)
>>> L2
[1, 2, 3, 4]
```

L2 是去重复后的列表,上面的过程也可简单地写为:

```
>>> L2 = list(set(L1))
>>> print L2
[1, 2, 3, 4]
```

set 是可以改变的集合类型，如果创建后的集合元素不需要改变，可创建不可变集合。

【例 3-4-19】 创建一个星期的英文缩写的不可变集合。

```
>>> s6 = frozenset(('MON','TUE','WED','THU','FRI','SAT','SUN'))
>>> s6
frozenset({'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'})
```

（2）访问集合。

由于集合本身是无序的，所以不能为集合创建索引或切片操作，只能循环遍历或使用 in、not in 来访问或判断集合元素。

【例 3-4-20】 集合访问示例。

```
>>> 'SUN' in s6
True
>>> 'MON' in s6
False
>>> for i in s6:
        print(i,end = " ")

SUN WED TUE SAT FRI MON THU
```

（3）集合运算。

集合支持的运算有：交集、并集、差集、对称差集，与中学数学中学习的集合的运算的概念相同，常用的集合运算如表 3-4-4 所示。

表 3-4-4 集合的常见运算

运　算	描　述	运　算	描　述
x in s1	检测 x 是否在集合 s1 中	s1==s2	判断集合是否相等
s1|s2	并集	s1<=s2	判断 s1 是否是 s2 的子集
s1&s2	交集	s1<s2	判断 s1 是否是 s2 的真子集
s1-s2	差集	s1>=s2	判断 s1 是否是 s2 的超集
s1^s2	异或集，求 s1 与 s2 中相异元素	s1>s2	判断 s1 是否是 s2 的真超集
s1|=s2	将 s2 的元素并入 s1		

【例 3-4-21】 集合运算示例。

对前面已建立的集合 s2、s5 做以下操作：

```
>>> s2
{'hello'}
>>> s5
{'here', 'hello', 'he', 'her'}
>>> s2 <= s5                                    #判断 s2 是否是 s5 的子集
True
```

创建新的集合 s7，作以下操作：

```
>>> s7 = {'hen','height','her'}
>>> s7| = s2                                    #将 s2 并入 s7
>>> s7
```

```
{'hello', 'her', 'height', 'hen'}
>>> s7&s5                                    #求 s7 和 s5 的交集
{'hello', 'her'}
>>> s7|s5                                    #求 s7 和 s5 的并集
{'here', 'her', 'hello', 'hen', 'he', 'height'}
>>> s7 - s5                                  #求 s7 中去除 s5 中有的元素
{'height', 'hen'}
>>> s7 ^ s5                                  #求 s7 和 s5 中各不相同的元素
{'he', 'hen', 'height', 'here'}
```

(4) 集合对象的方法。

Python 以对象方式实现集合类型,假设 s1 是集合对象,s2 可以是一个可迭代对象,集合对象的常用方法如表 3-4-5 所示,可以支持可变集合完成集合元素的增加、删除和集合的复制。

表 3-4-5　集合对象的常用方法

描　　述	方　　法
s1.union(s2)	s1|s2,返回一个新的集合对象
s1.difference(s2)	s1－s2,返回一个新的集合对象
s1.intersection(s2)	s1&s2,返回一个新的集合对象
s1.issubset(s2)	s1<＝s2
s1.issuperset(s2)	s1>＝s2
s1.update(s2)*	将 s2 的元素并入 s1
s1.add (x)*	增加元素 x 到 s1
s1.remove(x)*	从 s1 移除 x,x 不存在报错
s1.clear ()*	清空 s1
s1.copy()	复制 s1,返回一个新的集合对象

其中加星号 * 的方法是 set 集合独有的方法,不加星号的方法是 set 和 frozenset 两种集合都有的方法。

【例 3-4-22】 集合对象的方法示例。

set 和 frozenset 对象都有 union 方法,返回一个合并后的新对象(调用该方法的原对象内容不变),要注意的是 set 对象调用 union 方法,返回的是 set 对象,frozenset 对象调用 union 方法,返回的是 frozenset 对象。

```
>>> s5                                  #set 对象
{'here', 'hello', 'he', 'her'}
>>> s6                                  # frozenset 对象
frozenset({'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'})
>>> s9 = s6.union(s5)                   # frozenset 对象调用 union 得到 frozenset 对象
>>> s9
frozenset({'her', 'SUN', 'hello', 'WED', 'here', 'TUE', 'SAT', 'FRI', 'MON', 'he', 'THU'})
>>> s10 = s5.union(s6)                  # set 对象调用 union 得到 set 对象
>>> s10
{'her', 'SUN', 'MON', 'WED', 'here', 'TUE', 'SAT', 'FRI', 'hello', 'he', 'THU'}
>>> s5                                  #调用 union 产生新对象,原对象不变
```

```
{'here', 'hello', 'he', 'her'}
>>> s10 = s10.difference(s5)        #返回 s9 和 s5 的差集,产生一个新对象,再赋值给 s10
>>> s10
{'SUN', 'WED', 'TUE', 'SAT', 'FRI', 'MON', 'THU'}
```

注意：union 方法是返回一个新的对象，调用对象本身不发生变化。update 方法是对调用方法的对象直接修改，所以只适合可修改的 set 对象，表 3-4-5 中加星号 * 的方法都会修改调用方法的对象本身。

```
>>> s5.update(s6)                   #将 s6 的元素并入 s5 中
>>> s5
{'MON', 'FRI', 'he', 'THU', 'here', 'WED', 'her', 'hello', 'SUN', 'SAT', 'TUE'}
```

表中的 s2 并不要求是相同类型的对象，只要是一个可迭代(iterable)的对象，包括字符串、列表、元组、集合。例如：

```
>>>                                 ##将一个列表与集合 s6 联合
>>> L1 = [1,2,3]
>>> s11 = s6.union(L1)
>>> s11
frozenset({1, 2, 3, 'TUE', 'SAT', 'FRI', 'SUN', 'MON', 'WED', 'THU'})
>>>                          ##判断 1 是否在 s11 集合中不能用 int 类型,要用列表类型或集合类型
>>> s11.issuperset(1)
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    s11.issuperset(1)
TypeError: 'int' object is not iterable
>>> s11.issuperset({1})
True
>>> s11.issuperset([1])
True
```

(5) 应用。

可以利用集合运算很方便地比较两个集合的相同元素和不同元素。

【例 3-4-23】 利用集合分析活动投票情况。

两个小队举行活动评测投票，按队员序号投票，第一小队队员序号为 1、2、3、4、5，第二小队队员的序号为 6、7、8、9、10，可以对投票数据进行分析，投票数据为 1,5,9,3,9,1,1,7,5,7,7,3,3,1,5,7,4,4,5,4,9,5,5,9。

建立集合 s2 表示第一小队队员序号，s3 表示第二小队队员序号：

```
>>> s2 = {1,2,3,4,5}
>>> s3 = {6,7,8,9,10}
```

使用投票数据建立集合 s3，集合去重复后表示获得了选票的队员序号：

```
>>> s1 = {1,5,9,3,9,1,1,7,5,7,7,3,3,3,1,5,7,4,4,5,4,9,5,5,9}
>>> s1
{1, 3, 4, 5, 7, 9}
```

第一小队获得选票的队员有：

```
>>> s1 - s3
{1, 3, 4, 5}
```

第一小队没有获得选票的队员有：

```
>>> s2 - (s1 - s3)
{2}
```

第二小队获得选票的队员有：

```
>>> s1 - s2
{9, 7}
```

第二小队没有获得选票的队员有：

```
>>> s3 - (s1 - s2)
{8, 10, 6}
```

2. 字典

序列采用查找信息的方式是通过序列元素的位置下标引用指定的序列元素，字典采用了另一种通过键值来查找信息的方式，键值和索引值反映了一种数据之间的关联，例如在表示星期时，通常用 1 表示星期一(MON)，6 表示星期六(SAT)，0 表示星期日(SUN)。字典是一个由键和值组成的键值对构成的集合，每一个字典元素分为两部分：键(key)和值(value)。

字典是 Python 中唯一内置映射数据类型，可以通过指定的键从字典访问值。字典类型 dict 与集合类型 set 一样是无序的集合体，键值对没有特定的排列顺序，所以不能通过位置下标访问字典元素。

(1) 字典的创建。

字典的创建同样可以通过字面值和类型构造器的方式。

• 字面值

字典的字面值是由一对大括号括起的，以逗号分隔的键值对构成的，键值对的书写形式为<键>：<值>。

【例 3-4-24】 字典的字面表示示例。

```
>>> d1 = {1:'MON',2:'TUE',3:'WED',4:'THU',5:'FRI',6:'SAT',0:'SUN'}
>>> d1
{0: 'SUN', 1: 'MON', 2: 'TUE', 3: 'WED', 4: 'THU', 5: 'FRI', 6: 'SAT'}
```

与集合类似，字典中键值对的顺序与定义时的顺序是不一样的。

字典可嵌套，可以在一个字典里包含另一个字典。

【例 3-4-25】 嵌套字典示例。

```
>>> test = {"test":{"mytest":10} }
>>> test
{'test': {'mytest': 10}}
```

• 类型构造器 dict()

使用类型构造器构造字典，参数为键值对，键值对之间以“,”分隔，键值对的书写形式为<键>=<值>。

【例 3-4-26】 字典的类型构造器构造示例。

```
>>> monthdays = dict( Jan = 31, Feb = 28, Mar = 31, Apr = 30, May = 31, Jun = 30, Jul = 31, Aug =
31, Sep = 30, Oct = 31, Nov = 30,Dec = 31 )
>>> monthdays
{'May': 31, 'Aug': 31, 'Feb': 28, 'Mar': 31, 'Jan': 31, 'Jul': 31, 'Jun': 30, 'Sep': 30, 'Nov':
30, 'Dec': 31, 'Oct': 31, 'Apr': 30}
```

类型构造器对键值对的要求比字面值的键值对的要求更严格，键名 key 必须是一个标识符，而不能是表达式，例如：类似 d1 的字典不能使用类型构造器生成，因为整数不能作为 key。

【例 3-4-27】 使用类型构造器构造字典示例。

```
>>> weekday = dict(1 = 'MON',2 = 'TUE',3 = 'WED',4 = 'THU',5 = 'FRI',6 = 'SAT',0 = 'SUN')
SyntaxError: keyword can't be an expression
>>> weekday = dict(a1 = 'MON',a2 = 'TUE',a3 = 'WED',a4 = 'THU',a5 = 'FRI',a6 = 'SAT',a0 = 'SUN')
>>> weekday
{'a3': 'WED', 'a2': 'TUE', 'a1': 'MON', 'a0': 'SUN', 'a6': 'SAT', 'a5': 'FRI', 'a4': 'THU'}
```

(2) 字典元素的访问。

字典元素的访问方式是通过键访问相关联的值，访问形式为：<字典对象>[<键>]。

例如 monthdays["Jan"]，可访问值 31。如果没有找到指定的键，则解释器会引起异常。

【例 3-4-28】 字典元素的访问。

```
>>> monthdays["Jan"]
31
>>> monthdays["Jau"]
Traceback (most recent call last):
  File "< pyshell # 3 >", line 1, in < module >
    monthdays["Jau"]
KeyError: 'Jau'
```

(3) 字典的基本运算。

• 字典是可修改的。

【例 3-4-29】 monthdays["Jan"]=30，可把 Jan 的值由 31 改为 30。

```
>>> monthdays["Jan"] = 30
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul':
31, 'Mar': 31, 'Sep': 30, 'Nov': 30}
```

• 字典是可添加元素的。

【例 3-4-30】 monthdays["test"]=30 可添加一个新键值对。

```
>>> monthdays["test"] = 30
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul':
31, 'test': 30, 'Mar': 31, 'Sep': 30, 'Nov': 30}
```

• 字典是可删除元素的。

【例 3-4-31】 del monthdays["test"]可删除字典条目。

```
>>> del monthdays["test"]
>>> monthdays
{'Apr': 30, 'Dec': 31, 'May': 31, 'Feb': 28, 'Aug': 31, 'Oct': 31, 'Jan': 30, 'Jun': 30, 'Jul':
31, 'Mar': 31, 'Sep': 30, 'Nov': 30}
```

字典不属于序列对象,所以不能进行连接和相乘操作。字典是没有顺序的。

(4) 字典对象的方法

与列表一样,字典也提供了对象方法来对字典进行操作。假设 d 为字典对象,字典对象的常用方法如表 3-4-6 所示。

表 3-4-6 字典对象的常用方法

方 法	描 述
d. keys()	返回字典 d 中所有键的列表,类型为 dict_keys
d. values()	返回字典 d 中值的列表,类型为 dict_values
d. items()	返回字典 d 中由键和相应值组成的元组的列表,类型为 dict_items
d. clear()	删除字典 d 的所有条目
d. copy()	返回字典 d 的浅拷贝,不复制嵌入结构
d. update(x)	将字典 x 中的键值加入到字典 d
d. pop(k)	删除键值为 k 的键值对,返回 k 所对应的值
d. get(k[,y])	返回键 k 对应的值,若未找到该键返回 none,若提供 y,则未找到 k 时返回 y

【例 3-4-32】 字典方法示例。

```
>>> monthdays.keys()                    # 显示字典 monthdays 的键值序列
dict_keys(['Apr', 'Dec', 'May', 'Feb', 'Aug', 'Oct', 'Jan', 'Jun', 'Jul', 'Mar', 'Sep', 'Nov'])
>>> monthdays.values()                  # 显示字典 monthdays 的值序列
dict_values([30, 31, 31, 28, 31, 31, 30, 30, 31, 31, 30, 30])
```

dict_keys 和 dict_values 也是一个迭代器对象,可以通过迭代方式访问其中的元素,例如:

```
>>> for i in monthdays.keys():
    print(i,end = " ")
Apr Dec May Feb Aug Oct Jan Jun Jul Mar Sep Nov
>>> monthdays.items()                   # 显示字典 monthdays 的键值对序列
dict_items([('Apr', 30), ('Jul', 31), ('Jun', 30), ('Oct', 31), ('Mar', 31), ('Jan', 30), ('May', 31),
('Nov', 30), ('Dec', 31), ('Aug', 31), ('Sep', 30), ('Feb', 28)])

>> x = {'a1':21,'a2':34}                # 创建一个新的字典 x
>>> x
{'a2': 34, 'a1': 21}
>>> monthdays.update(x)                 # 将字典 x 的键值对追加到字典 monthdays 中
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30, 'May': 31, 'Nov': 30, 'Dec':
31, 'a2': 34, 'a1': 21, 'Aug': 31, 'Sep': 30, 'Feb': 28}
>>> monthdays.pop('a1')                 # 删除键为'a1'的键值对
```

```
21
>>> monthdays
{'Apr': 30, 'Jul': 31, 'Jun': 30, 'Oct': 31, 'Mar': 31, 'Jan': 30, 'May': 31, 'Nov': 30, 'Dec':
31, 'a2': 34, 'Aug': 31, 'Sep': 30, 'Feb': 28}
>>> monthdays.get('a2')            #获取键'a2'对应的值
34
>>> monthdays.get('a1','not found')          #获取键'a1'对应的值,没有找到则返回'not found'
'not found'
```

(5) 应用。

【例 3-4-33】 建立一个字典对象,能够通过数字 1～12 表示月份,查阅对应的英文月份的缩写。

创建一个 key 为序号,value 为英文月份的缩写的集合:

```
>>> monthname = {1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May', 6:'Jun', 7:'Jul', 8:'Aug', 9:'Sep',10:
'Oct', 11:'Nov',12:'Dec' }
>>> monthname
{1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct',
11: 'Nov', 12: 'Dec'}
```

按 key 值查询对应英文月份的缩写:

```
>>> monthname [1]
'Jan'
```

输出所有的 12 个月的英文月份的缩写:

```
>>> for i in monthname.keys():
    print(monthname [i],end = ' ')

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

【例 3-4-34】 建立 9×9 乘法表,可以根据两个乘数,查阅字典得到乘积。

以 3 的乘法为例:

```
>>> d1 = {(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}
>>> d1
{(3, 8): 24, (3, 2): 6, (3, 9): 27, (3, 3): 9, (3, 6): 18, (3, 7): 21, (3, 1): 3, (3, 4): 12,
(3, 5): 15}
>>> d1[(3,9)]
27
```

【例 3-4-35】 成绩排序。

已有 5 位同学的姓名和成绩,按成绩从高到低列出同学姓名,假设成绩没有重复值。

方法一:按<成绩:姓名>建立字典,从字典获取由成绩组成的列表,从高到低排序后,根据列表中的成绩,逐个从字典中查找对应的姓名,写出另一个列表。得到的新列表中的姓名就是按成绩排序的。

```
>>> scores = {85:"李鸣",74:"黄辉",92:"张檬",88:"于静颂",63:"钱多多"}
>>> scores
{88: '于静颂', 74: '黄辉', 92: '张檬', 85: '李鸣', 63: '钱多多'}
```

```
>>> L1 = list(scores.keys())
>>> L1
[88, 74, 92, 85, 63]
>>> L1.sort(reverse = True)
>>> L1
[92, 88, 85, 74, 63]

>>> L2 = []
>>> L2.append(scores[L1[0]])
>>> L2.append(scores[L1[1]])
>>> L2.append(scores[L1[2]])
>>> L2.append(scores[L1[3]])
>>> L2.append(scores[L1[4]])
>>> L2
['张檬', '于静颂', '李鸣', '黄辉', '钱多多']
```

得到 L2 列表的过程在学习了循环结构后可改为：

```
>>> L2 = []
>>> for i in range(0,len(L1)):
        L2.append(scores[L1[i]])
>>> L2
```

* 方法二：直接利用嵌套列表的 sort 方法的 key 参数，写一个 lambda 函数，对每一个列表成员返回第二项，即按第二项排序。reverse 为 true，表示倒序从高到底。

```
>>> Ls = [["李鸣",85],["黄辉",74],["张檬",92],["于静颂",88],["钱多多",63]]
>>> Ls.sort(key = lambda x:x[1],reverse = True)
>>> Ls
[['张檬', 92], ['于静颂', 88], ['李鸣', 85], ['黄辉', 74], ['钱多多', 63]]
>>> for i in range(0,len(Ls)):
    print(Ls[i][0],end = ' ')

张檬 于静颂 李鸣 黄辉 钱多多
```

3.5 本章小结

本章所介绍的主要内容是数值数据对象、文本数据对象和批量数据对象的常量表示和对象创建的方法，以及作用在这些数据对象上的基本操作：如何访问数据对象、数据对象支持的运算以及数据对象提供的方法。

本章的内容是学好 Python 语言的基础，重要的语法知识包括：

(1) 数据都是属于一定类型的，数据类型是一组数据及在这组数据上的运算，它规定了这一类数据：①存储结构；②存储机制，即各种数据类型的编码方式；③运算和操作。

(2) Python 以类的方式管理数据，Python 的内置类型主要区分为简单类型和容器类型，简单类型主要是数值型数据，包括整型数据、浮点型数据、布尔类型数据和其他语言不多见的复数数据。容器类型可以应用于一次处理多个对象的场合，包括字符串 str、元组 tuple、列表 list、集合类型 set、字典类型 dict。

(3) 程序中数据有两种表示方式：常量和变量。常量是数据的文字量，是数据的“书写形式”。变量描述的是存储空间的概念，将数据存储在内存中，内存空间就是可操作的变量，用一个名称来引用内存空间，这个名称称为变量名。变量的值是可以变化的。Python 语言使用“动态类型”技术，变量使用前不需要声明数据类型即可使用，然后根据其中变量存放的数据不同，决定其数据类型。

(4) 标识符用来标识一个对象，Python 中的标识符由大小写英文字母、数字、下画线组成、以英文字母、下画线为首字符，也就是说不能以数字开头，长度任意，大小写敏感。标识符不能与 Python 关键字同名。

(5) Python 的表达式可以由常量、变量、运算符、函数按照一定的规则构造，描述计算过程。最简单的表达式可以是一个常量或一个变量。

(6) 数值运算主要包括算术运算、关系运算和逻辑运算。算术运算符有＋、－、＊、/、//、%；关系运算符有＜、＜＝、＞、＞＝、＝＝、!＝；逻辑运算符有 and、or、not。

(7) 影响表达式计算顺序的因素包括：运算符的优先级、运算符的结合方式和括号。数值运算的优先级是先算术运算，再关系运算，最后是逻辑运算。算术运算同样遵循先乘除后加减的顺序。逻辑运算的优先级是先“非”再“与”最后“或”。括号的优先级最高。

(8) 只有同类型的数据对象才能进行运算，混合类型的数据进行运算时要进行类型转换。系统有自动转换的机制，自动转换的基本原则是将表示数值范围小的数据类型的值转换到表示数值范围大的数据类型的值。强制转换机制是指程序员可以使用 Python 语言提供各种类型的转换函数在表达式中明确混合运算中数据的转换类型。

(9) Python 的系统函数扩充了 Python 的计算能力，系统函数由 Python 标准库中的模块提供。标准库中的模块又分成内置模块和非内置模块，内置模块__builtin__中的函数和变量可以直接使用，非内置模块要通过 import 命令先导入再使用。

(10) 计算机中表示文本的最基本的单位是字符，包括可打印字符和不可打印的控制字符。可打印的字符直接由键盘输入，不可打印的字符以转义字符表示，Python 中的转义字符以“\”为前缀。

(11) 字符串常量以一对双引号或单引号表示，字符串类型支持的运算有＋和＊，实现连接和复制。可以通过下标访问字符串中的一个字符或一个子串，也称为索引方式。但不能通过下标方式去改变字符串的内容。

(12) Python 可支持批量数据存储和操作，其中有序的数据集合体，也称为序列，包括字符串、元组和列表，序列可以通过索引或下标来访问其数据成员，序列的通用操作包括索引、连接、复制、检测等；无序的数据集合体包括集合、字典等，无序的数据集合体不支持索引操作。

(13) 批量数据对象的创建可以通过字面形式，给对象赋常量值，也可以通过类型构造器创建。例如创建一个空的元组对象，可以直接将一个空的元组赋给元组对象：t＝()；也可以使用无参的元组类型构造器创建：t＝tuple()。

(14) 每一种批量数据对象都提供了丰富的方法，以支持对批量数据对象的各种操作，方法的调用形式为：<对象名>.方法名(<参数>)。

3.6 习题与思考

1. 请指出下面合法的 Python 标识符是_____。

A. Day　　B. e10　　C. 2n　　D. a[10]
E. False　　F. aAbB　　G. a+b　　H. _ifdef
I. day_of_year

2. 以下是出现在程序中的数值常量,正确的是_____。

A. 38499L　　B. .314e1　　C. e5　　D. 1e2.5
E. 0o378　　F. 0xabc　　G. 0b1010　　H. true
I. 5-6.5j　　J. 78.90

3. 以下是出现在程序中的文本常量,正确的是_____。

A. """　　B. 'ab'　　C. '*'　　D. '"ab"'
E. " "a" "　　F. '\456'　　G. '\"'　　H. '\xah'
I. "a+b"

4. 不是 Python 的关键字有_____。

A. list　　B. for　　C. from　　D. dict
E. False　　F. print　　G. or　　H. in
I. and

5. 以下表达式中,_____的运算结果是 False。

A. (10 is 11) == 0　　B. 'abc' < 'ABC'
C. 3 < 4 and 7 < 5 or 9 > 10　　D. 24 != 32

6. 已知某函数的参数为 35.8,执行后结果为 35,可能是以下函数中的_____。

A. int　　B. round　　C. floor　　D. abs

7. 如果想要查看 math 库中 pi 的取值是多少,可以利用以下_____方式(假设已经执行了 import math,并且只要包含 pi 取值就可以)。

A. print (math.pi)　　B. dir(math)　　C. help(math)　　D. print(pi)

8. 以下_____语句不可以打印出"hello world"字符串(结果需在同一行)。

A. print('''hello
world''')
B. print("hello world")
C. print('hello world')
D. print('hello \
world')

9. 写出执行完下面数值表达式语句后,变量 a~k 的值。

```
>>> a = 5
>>> b = 2
>>> a * = b
>>> b += a
>>> a,b = b,a
>>> c = 6
>>> d = c % 2 + (c + 1) % 2
```

```
>>> e = 2.5
>>> f = 3.5
>>> g = (a + b) % 3 + int(f)//int(e)
>>> h = float(a + b) % 3 + int(f)//int(e)
>>> i = (a + b)/3 + f % e
>>> j = a < b and c < d
>>> k = not j and True
```

10. 已知 s="Happy Birthday",写出下面输出语句 print 的输出结果。

(1) print(len('\\n\n\456'))

(2) print(('hello ' +'world\n') * 3)

(3) print(s[0]+s[11])

(4) print(s[6:11])

(5) print(s[:5]+s[11:])

11. 已知列表 L1 和 L2,由 L1 和 L2 构造 L3,并回答问题。

```
>>> L1 = [1,2,3,4,5]
>>> L2 = ["one","two","three","four","five"]
>>> L3 = [[L1[1],L2[1]],[L1[2],L2[2]],[L1[3],L2[3]]]
```

(1) L3 的值是什么?

(2) L3[1,1]的值是什么?

(3) 执行 L4=L3.pop(2)后,列表 L3 和 L4 的值是什么?

(4) 再执行 L3.extend(L4),列表 L3 的值是什么?

12. 集合 a、b 中存放着两组文件名的集合,两个集合中有相同的文件也有不同的文件,请写出实现下面功能的表达式。

```
a = {"3 - 1.py","3 - 5.py","3 - 6.py","3 - 8.py","3 - 9.py"}
b = {"3 - 1.py","3 - 2.py","3 - 6.py","3 - 7.py","3 - 8.py"}
```

(1) 求 a 中存在,b 中不存在的文件;

(2) 求 a 中存在与 b 中相同的文件;

(3) 求两个文件夹中互不相同的文件;

(4) 求两个文件夹中总共包括的文件的个数;

13. 下面定义字典 monthdays 的语句都正确吗? 如果不正确,说明为什么?

(1) monthdays = dict(Jan=31, Feb=28, Mar=31, Apr=30, May=31, Jun=30, Jul=31, Aug=31, Sep=30, Oct=31, Nov=30,Dec=31)

(2) monthdays = dict('Jan'=31, 'Feb'=28, 'Mar'=31, 'Apr'=30, 'May'=31, 'Jun'=30, 'Jul'=31, 'Aug'=31, 'Sep'=30, 'Oct'=31, 'Nov'=30,'Dec'=31)

(3) monthdays = {Jan=31, Feb=28, Mar=31, Apr=30, May=31, Jun=30, Jul=31, Aug=31, Sep=30, Oct=31, Nov=30,Dec=31}

(4) monthdays = {Jan:31, Feb:28, Mar:31, Apr:30, May:31, Jun:30, Jul:31, Aug:31, Sep:30, Oct:31, Nov:30,Dec:31}

(5) monthdays = {'Jan':31, 'Feb':28, 'Mar':31, 'Apr':30, 'May':31, 'Jun':30, 'Jul':31, 'Aug':31, 'Sep':30, 'Oct':31, 'Nov':30,'Dec':31}

14. 思考在下面举出的应用环境中适合的数据类型,试在 Python shell 中举实例表示。

(1) 100 以内的素数;

(2) 1～10 的数字的阶乘;

(3) 斐波那契数列;

(4) 班级考试成绩单;

(5) 自定义的英汉字典。

3.7 实训 数据表示和计算

1. 实验目标

(1) 理解数据类型的概念和数据的文字量表示。

(2) 掌握数值类型和文本类型的基本操作。

(3) 理解变量的存储概念。

(4) 掌握列表和元组的概念及基本操作。

2. 实验范例

本章的实验使用 Python 的交互模式,像使用计算器一样使用 Python。交互模式可以在 Python(commands line)环境或 Python shell 环境下使用。

① Python(commands line)

从开始菜单中选择 Python33→Python(commands line),等待提示符>>>。在窗口的第一行显示 Python 的版本号,如图 3-7-1 所示。

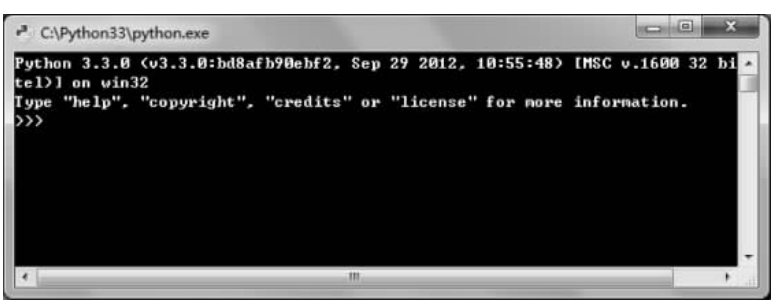

图 3-7-1 Python(commands line)

② Python Shell

从开始菜单中选择 Python33→IDLE(Python GUI),如图 3-7-2 所示。

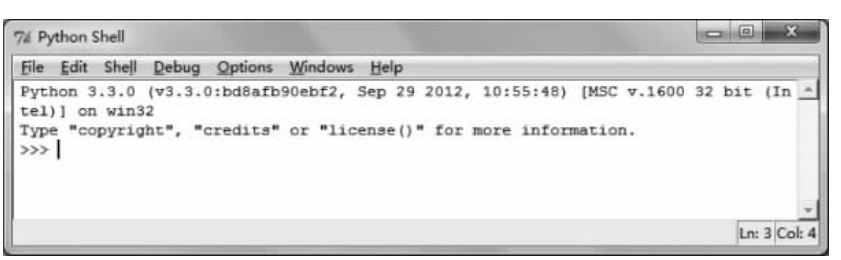

图 3-7-2 Python shell

这两种方式都支持 Python 交互模式，区别在于 Python(commands line)是 DOS 控制台模式，不支持鼠标操作，Python Shell 是窗口模式，支持鼠标操作，也支持剪贴板操作。

例如在交互模式中，要想重复执行前面已执行过的命令，或想获得前面已执行过的命令修改得到新的命令，Python(commands line)中使用光标键“↑”，上翻到所需命令。Python Shell 中可以用复制粘贴的方法，更快捷的操作是在已完成的命令行任意位置单击将光标插入文本后按 Enter 键，该行文本会自动复制到当前等待输入的命令行提示符的后面，可进行修改后或直接按 Enter 键再次执行。

(1) 认识基本数据的类型、表示和运算

① 直接输入以下表达式并查看结果。

```
23 + 3、 23 > 3、 '23' + '3'、  23/3、 23//3、  23 % 3、  23 ** 3
>>> 23 + 3
26
>>> 23 > 3
True
>>> '23' + '3'
'233'
>>> 23/3
7.666666666666667
>>> 23//3
7
>>> 23 % 3
2
>>> 23 ** 3
12167
```

注意：命令行提示符后不要插入空格，否则会引起系统错误。

```
>>> 23 % 3
  SyntaxError: unexpected indent
```

② 直接输入以下表达式并查看结果。

```
23 + 24.5、 23 + '3'、 23 + int('3')、 'hello ' + str("123")、 int(23/3)、 round(23/3,2)、
round(23/3)、 0 < 23 < 100
```

不同的数据类型进行运算时，会进行类型的转换，整型数据和浮点数据相遇，整型数据转化为浮点类型。

```
>>> 23 + 24.5
47.5
```

当自动类型转化不成功或出现系统错误，例如整型与字符串类型相加出错：

```
>>> 23 + '3'
Traceback (most recent call last):
  File "< pyshell # 9 >", line 1, in < module >
    23 + '3'
TypeError: unsupported operand type(s) for + : 'int' and 'str'
```

可以通过类型转化函数，显示完成数据类型转换后计算。

```
>>> 23 + int('3')
26
>>> 'hello ' + str(123)
'hello 123'
```

int 函数还可以完成对浮点数取整的功能：

```
>>> int(23/3)
7
```

round 函数的功能更为灵活，可以按指定位取整，四舍五入。第二个参数指定取整位置，n 表示小数点后 n 位，默认表示没有小数点。

```
>>> round(23/3,2)
7.67
>>> round(23/3)
8
```

Python 支持下面连写的比较方式，但与其他高级语言迥异。

```
>>> 0 < 23 < 100
True
```

与其他高级语言相同的写法应为：

```
>>> (23 > 0) and (23 < 100)
True
```

建议使用逻辑运算的方式表示多个关系表达式之间的关系。

```
>>> 'hello ' + str("123")
'hello 123'
```

③ 直接输入变量赋值语句并接着显示该变量值或类型。

输入：a＝23.5，再输入：a 显示该变量值，最后输入：type(a)显示变量类型；

```
>>> a = 23.5
>>> a
23.5
>>> type(a)
<class 'float'>
```

输入：b＝a＞0，再输入：b 显示该变量值，最后输入：type(b)显示变量类型；

```
>>> b = a > 0
>>> b
True
>>> type(b)
<class 'bool'>
```

输入：c＝None，再输入：c 显示该变量值，最后输入：type(c)显示变量类型；

```
>>> c = None
>>> c
>>> type(c)
<class 'NoneType'>
```

注意：None 表示空类型。

输入：d='23'+'3'，再输入：d 显示该变量值，最后输入：type(d)显示变量类型，输入：len(d)显示变量长度。

```
>>> d = '23' + '3'
>>> d
'233'
>>> type(d)
<class 'str'>
```

(2) 数学模块库函数的使用

使用 math 模块的数学函数。导入数学库 math。然后输入以下表达式理解 math 中函数的使用。

```
math.sqrt(2 * 2 + 3 * 3)、math.log10(100)、 math.exp(2)、math.e、math.pow(2.5,2)、math.floor(2.5)、
math.ceil(2.5)、 math.fmod(4,3)、math.fabs( - 23.56)
```

导入数学库。

```
>>> import math
```

使用 math 前缀访问 sqrt 函数求平方根。

```
>>> math.sqrt(2 * 2 + 3 * 3)
3.605551275463989
>>> math.log10(100)
2.0
```

另一种导入数学库的方式，是如下访问 pow 函数求 x 的 y 次方式。

```
>>> from math import *
>>> pow(2.5,2)
6.25
>>> pi
3.141592653589793
>>> e
2.718281828459045
>>> math.floor(2.5)
2
>>> ceil(2.5)
3
>>> fabs( - 23.56)
23.56
```

(3) 变量的表示和操作

① 输入平面的两个点的坐标(1.0,2.1),(5.2,10.33)，计算两点之间的距离。计算两点之间距离的公式为：$d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$。

```
>>> from math import *
>>> x1,y1 = 1.0,2.1
>>> x2,y2 = 5.2,10.33
>>> d = sqrt(pow(x2 - x1,2) + pow(y2 - y1,2))
```

```
>>> print(d)
9.239745667495399
```

增加 x2 和 y2 的值,再计算:

```
>>> i = 5
>>> x2,y2 = x2 + i,y2 + i
>>> d = sqrt(pow(x2 - x1,2) + pow(y2 - y1,2))
>>> print(d)
16.114369364017943
```

② 求任意三位整数的逆序数。

求一个三位数的逆序数的思路是将构成三位数的三个数字取出,重新按权位组合。取位可以通过求余和整除运算完成。

```
>>> a = 123
>>> d1,d2,d3 = a//100,a % 100//10,a % 10
>>> d1,d2,d3
(1, 2, 3)
>>> b = d3 * 100 + d2 * 10 + d1
>>> b
321
```

(4) 文本对象的表示和操作

假设执行了如下语句:

```
>>> s1 = 'programming'
>>> s2 = 'language'
```

直接输入下面表达式,观察计算结果,理解表达式的含义。

```
s1[1]、s1[:4]、s1[0] + s2[1:3]、s1.capitalize() + ' ' + s2.upper()、s1.count('r') + s1.find('r') +
s1.rfind('r')、s3 = s2.join(' -- ')、s4 = ' - '.join(s2)、L1 = s4.split()、3 * (s2[:2] + ' ')、
"Python" + s2.rjust(10)。
```

① 取下标为 1 的字符串字符。

```
>>> s1[1]
'r'
```

② 取下标从开始到 3 的子串。

```
>>> s1[:4]
'prog'
```

③ 连接 s1 串下标为 0 的字符和 s2 串下标从 1 到 2 的子串。

```
>>> s1[0] + s2[1:3]
'pan'
```

④ 连接首字符大写后的 s1 串和全部大写的 s2 串。

```
>>> s1.capitalize() + ' ' + s2.upper()
'Programming LANGUAGE'
```

⑤ 计算 s1 串中的字符'r'的个数,从字符'r'在 s1 串的左边和右边第一次出现的位置,并累和。

```
>>> s1.count('r') + s1.find('r') + s1.rfind('r')
7
```

⑥ 将字符串 s2 作为分隔符加入到序列参数"--"的每个字符之间。

```
>>> s3 = s2.join('--')
>>> s3
'-language-'
```

⑦ 将字符串'-'作为分隔符加入到参数 s2 串的每个字符之间。

```
>>> s4 = '-'.join(s2)
>>> s4
```

⑧ 以参数字符'-'为分隔符,将字符串 s4 分离到一个列表中。

```
'l-a-n-g-u-a-g-e'
>>> L1 = s4.split('-')
>>> L1
['l', 'a', 'n', 'g', 'u', 'a', 'g', 'e']
```

⑨ 取串 s2 开始两个字符连接一个空格后复制 3 次。

```
>>> 3 * (s2[:2] + ' ')
'la la la '
```

(5) 序列的表示和操作

① 输入:t1=='001001', 'Li Si', 'men', 18,再输入:t1 显示该变量值,输入:t1[0]和 t1[1]显示部分数据,最后输入:type(t1)显示变量类型,输入:len(t1)显示长度。

```
>>> t1 = '001001', 'Li Si', 'men', 18
>>> t1
('001001', 'Li Si', 'men', 18)
>>> t1[0]
'001001'
>>> t1[1]
'Li Si'
>>> type(t1)
<class 'tuple'>
>>> len(t1)
4
```

② 输入:t2=['001001','Li Si','men',18],再输入:t2 显示该变量值,输入:t2[0]和 t2[1]显示部分数据,最后输入:type(t2)显示变量类型,输入:'men' in t2 测试成员。

```
>>> t2 = ['001001','Li Si','men',18]
>>> t2
['001001', 'Li Si', 'men', 18]
>>> t2[0]
'001001'
```

```
>>> t2[1]
'Li Si'
>>> type(t2)
<class 'list'>
>>> 'men' in t2
True
```

③ 输入：t2[3]+=1,再输入：t2 查看该变量值。输入：t1[3]+=1 显示出错信息。

```
>>> t2[3] += 1
>>> t2
['001001', 'Li Si', 'men', 19]
>>> t1[3] += 1
Traceback (most recent call last):
  File "<pyshell#102>", line 1, in <module>
    t1[3] += 1
TypeError: 'tuple' object does not support item assignment
```

这说明列表的元素可以改变,而元组的元素不可以改变。

④ 输入：t2+=['021-65789293'],再输入：t2,查看该变量值。输入：t2[0:1]=[],再输入：t2,查看该变量值。

```
>>> t2 += ['021 - 65789293']
>>> t2
['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t2[0:1] = []
>>> t2
['Li Si', 'men', 19, '021 - 65789293']
```

⑤ 对将 t2 的内容复制一个副本 t3,对 t3 进行增删修改。

直接使用赋值运算得到的 t3,与 t2 是指向同一个列表对象的,对 t2 和 t3 的操作实质是使用不同的名称对一个对象操作。

```
>>> t2 = ['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t2
['001001', 'Li Si', 'men', 19, '021 - 65789293']
>>> t3 = t2
>>> t3[3] = 20
>>> t2
['001001', 'Li Si', 'men', 20, '021 - 65789293']
>>> id(t2),id(t3)
(33775328, 33775328)
```

要得到一个对象副本,可使用列表的方法 copy,复制后,两个列表的内容是相等的,但不是一个对象,注意“==”运算和“is”运算的区别。

```
>>> t3 = t2.copy()
>>> t3
['001001', 'Li Si', 'men', 20, '021 - 65789293']
>>> t3 == t2
True
>>> t3 is t2
```

```
False
>>> id(t2),id(t3)
(33775328, 34958072)
```

也可以从一个空列表开始，将 t2 的内容加入到空列表中。设置一个空列表对象是必须的，因为之前 t3 并没有指定一个确定的数据类型。

```
>>> t3 = []
>>> t3.extend(t2)
>>> t3
['001001', 'Li Si', 'men', 19, '021 - 65789293']
```

通过 append 方法可以在列表的尾部追加一个列表成员，例如增加一个身高的信息。

```
>>> t3.append(1.78)
>>> t3
['001001', 'Li Si', 'men', 19, '021 - 65789293', 1.78]
```

注意 append 与 extend 的区别。如果调用 t3.append(t2)，它是将列表 t2 作为一个列表成员加入的，而不是将列表 t2 的每一个成员分别加入的。

```
>>> t4 = []
>>> t4.append(t2)
>>> t4
[['001001', 'Li Si', 'men', 19, '021 - 65789293']]
>>> len(t4)
1
t4 中只嵌套了一个列表成员
```

insert 方法支持在指定位置增加列表成员，例如在年龄的后面插入身高信息。

```
>>> t3.insert(4,1.78)
>>> t3
['001001', 'Li Si', 'men', 19, 1.78, '021 - 65789293', 1.78]
```

remove 方法可用于删除第一个指定值，pop 可以删除并返回指定位置列表成员，例如将多余的身高成员删除。

```
>>> t3.pop(6)
1.78
>>> t3
['001001', 'Li Si', 'men', 19, 1.78, '021 - 65789293']
```

3. 实验内容

(1) 查阅 Python 3.3.0 自带的帮助文件 Python330.chm，文件存放位置为 Python33\Doc 文件夹，了解 Python 3.3.0 提供的内置函数(Built-in Functions)，写出其中 5 个你学会的函数的使用示例。

(2) 假设执行了如下语句

```
>>> x = 384
>>> a,b = 2.56769, 2.56789
>>> s1 = "She is the best student in her class"
>>> s2 = 'he'
```

写出下面条件判断语句:

① 判断 x 是否是奇数;

② 判断 x 是否能被 3 和 5 整除;

③ 判断 x 是否能被 3 或 5 整除;

④ 判断 b 与 a 的差值不超过 0.0001;

⑤ 判断 s2 是 s1 的子串;

⑥ 判断 s2 在 s1 中出现的次数超过 2 次。

(3) 假设执行了如下语句:

```
>>> s1 = 'programming'
>>> s2 = 'language'
```

利用 s1、s2 和字符串操作,写出能产生下列结果的表达式。

① 'program'

② 'ProLan'

③ 'am am am'

④ 'programming language'

⑤ 'progr@mming l@ngu@ge'

(4) 假设执行了如下语句:

```
>>> s1 = [0,1,2,3,4,5,6]
>>> s2 = ['SUN','MON','TUE','WED','THU','FRI','SAT']
```

利用 s1、s2 和列表操作,创建下列结果的序列对象(可分次完成):

```
s3: 'SUN|MON|TUE|WED|THU|FRI|SAT'
s4: [3, 4, 3, 4, 3, 4]
s5: [[0, 'SUN'], [1, 'MON'], [2, 'TUE'], [3, 'WED'], [4, 'THU'], [5, 'FRI'], [6, 'SAT']]
```

(5) 程序设计:使用圆柱的体积公式计算已知半径和高的圆柱的体积。

(6) 程序设计:输入连续 5 天的气温,求平均气温。

(7) 程序设计:使用查表法完成成绩类别的转换。

① 求任意一个分数(5 分制)对应等级。

1:A、2:B、3:C 、4:D,5:E

② 求任意一个分数(百分制)对应等级。

90～100:A、80～89:B、70～79:C 、60～69:D,0～59:E