

第四章 自定义函数

函数 (function) 是所有程序设计语言的核心内容之一, 在前面我们已经多次接触过函数。函数的最大优点是增强了代码的重用性和可读性, 能提高代码的重复利用率。像 `input()`、`print()` 等这些函数是 Python 工程师已经编写好了, 为我们直接使用。但我们也可根据需要自己编写函数, 称为自定义函数。本节介绍怎样自己编写函数的相关知识。(注: 本章有的地方涉及列表、字典、元组等数据类型概念, 理解有困难时可参考第五章数据结构相关内容)

4.1 函数的创建与调用

4.1.1 自定义函数的语法

```
def functionname(parameterlist):  
    ''' 函数说明'''  
    [函数体]
```

`functionname`: 函数名, 任何有效的 Python 标识符;

`parameterlist`: 参数列表, 如果有多个参数, 参数之间用 “,” 隔开, 也可以没有参数;

函数说明: 对函数的描述, 可以不写;

函数体: 函数被调用时执行的功能代码, 由一条或多条个语句组成。如果你只希望定义一个空函数, 可以使用 `pass` 语句占位。

注意: 函数说明与函数体需要有一定的缩进量, 且具有相同的缩进量。否则会引发 “invalid syntax” 异常。

例子: 定义一个对变量加 4 的函数。

```
01 def add(x):  
02     ''' 这个函数的功能是对变量 x 加 4'''  
03     x=x+4  
04     return x
```

在这里, `def` 是定义函数的关键字, 凡是在定义函数的地方都需要书写, `add` 是函数的名称, `x` 是参数, 语句 `return x` 的作用是返回 `x` 的值。但是, 这段代码是不会被计算机执行的, 因为这里只是定义而已, 相当于我们只是制造了一个具有适当功能的工具, 这个工具是否发挥作用, 还需要像前面调用内建函数一样进行调用, 函数才会被执行。

4.1.2 函数的调用

函数的调用很简单, 使用: `functionname(parameterlist)` 即可。`parameterlist` 要与定义函数时一致。如果函数有返回值, 需要使用形如: `value = functionname(parameterlist)`, 接收函数的返回值。比如我们调用上面创建的函数, 就可以使用下面的代码调用:

```
y = add(5)
```

当计算机执行完这条语句后, `y` 的值为 9。

例子: 创建一个函数, 函数的功能为: 判断两个数是否具有倍数关系。

```
01 def isMultiple(x, y):  
                                #定义函数  
02     if x % y ==0 or y % x ==0:  
03         print('yes')  
04     else:  
05         print('no')  
06 isMultiple(3, 6)             #调用函数  
07 isMultiple(2, 5)             #调用函数
```

注意:在函数被调用之前,必须已定义。如果把 06、07 行放到 01 行的前面就会引发“NameError: name 'isMultiple' is not defined r”异常。表示 isMultiple 没有被定义。

4.2 函数参数

4.2.1 形式参数与实际参数

在函数定义时,小括号里面的参数叫做形式参数。在调用函数时,小括号里面的参数叫做实际参数。形式参数只是一个标记,没有具体的值,只有当函数被调用时,通过实际参数赋值给形式参数,这时形式参数才有具体的值。

实际参数与形式参数之间是怎样传递值的呢? Python 中有两种传递值的方式。请先测试下面的例子。

例 1:

```
01 def add(x, y):
02     x += 1
03     y += 2
04     print(x, y)           #改变 x, y 的值后, 显示 x, y 的值
05 x = 2
06 y = 3
07 print(x, y)             #显示 x, y 的值
08 add(x, y)               #调用函数,
09 print(x, y)             #显示 x, y 的值
```

输出:

```
2 3
3 5
2 3
```

可以看出,调用函数后,尽管在函数中对形式参数 x, y 的值进行了修改,但是函数外面的 x, y 的值并没有被改变。这个例子说明了:尽管形式参数的名称与全局变量的名称相同,但他们并不是同一个对象。

例 2:

```
01 def add(a, b):
02     a[0] += 1
03     b[0] += 2
04     print(a[0], b[0])
05 x = [2]                #x 为列表, 元素为 2, 关于列表的知识参见第五章
06 y = [3]
07 print(x[0], y[0])     #显示 x, y 的第一个元素的值
08 add(x, y)             #调用函数
09 print(x[0], y[0])     #显示 x, y 的第一个元素的值
```

输出:

```
2 3
3 5
3 5
```

可以看出,调用函数后,在函数中对形式参数的值进行修改的同时也修改了实际参数的值。像例 1 中那样,修改形式参数的值并不影响实际参数的值,这种传递值的方式称为**值传递**。像例 2 中那样,修改形式参数的值也影响实际参数的值,这种传递值的方式称为**引用传递**。那么,什么情况进行值传递,什么情况进行引用传递呢? 答案是:由 Python 根据实际参数的类型自动确定,我们是不能改变这种传递方式的。Python 规定:把像数字、字符串、元组等**不可变类型数据**作为实际参数时按照**值传递**方式进行,把像列表、字典等**可变类型数据**作为实际参数时按照**引用传递**方式进行。

4.2.2 形式参数的类型

在函数定义时，参数列表中的参数可能有以下几种形式：

`functionname(parameter1, parameter2=' default' ,*args, parameter3,**kwargs)`，为方便后面的说明，把这个式子称为“范式”。

(1) 位置参数

位置参数也叫必选参数，是指在调用函数时必须传值的参数。这种参数在函数定义时只有名字且位置在其它参数类型的前面，如“范式”中的 `parameter1`。

(2) 默认参数

默认参数是指，在函数定义时设置了默认值的参数，在调用函数时可以传值也可以不传值。如“范式”中的 `parameter2`。位置必须在位置参数的后面，关键字参数的前面。建议默认值的类型为不可变类型。

(3) 可变参数

可变参数是用于接收那些除位置参数、默认参数以外的无名字的实际参数的值，个数可以是0个或多个，定义时需参数名前加上*号，如“范式”中的 `*args`。如果有位置参数或默认参数的话，可变参数必须写在他们的后面。

(4) 命名关键字参数

命名关键字参数是指在调用函数时，实际参数必须以“参数名=值”的形式进行调用的参数。在函数定义时有两种方式可以指明参数是命名关键字参数。

①在*后面的非关键字参数；如 `add(a, b, *, c, d)` 中的 `c, d`。（这里的*不是参数，只表示后面的 `c, d` 是命名关键字参数）；

②在可变参数后面的非关键字参数。如“范式”中的 `parameter3`。

(5) 关键字参数

关键字参数与可变参数类似，是用于接收那些除命名关键字参数以外的“键=值”形式的实际参数的值。个数可以是0个或多个，定义时需参数名称前用**号，如“范式”中的 `**kwargs`。关键字参数必须写在其它所有类型参数的后面。

注意：在函数定义时，这5种参数可以部分或全部使用，但顺序必须是：位置参数>默认参数>可变参数>命名关键字参数>关键字参数。

为了更好地理解这些参数类型的意义，下面以实例解释实际参数与形式参数的匹配顺序。

说明：

①调用函数时，实际参数只有两种形式，一是具体的值，二是“变量=值”的形式（如果实际参数中有*和**时，*会首先被“解包”为具体的值，**会首先被“解包”为“变量=值”的形式）。例如：

```
01 tup = (2, 3, 4)
02 dict = {'k2':2, 'k3':3}
03 fun(1,*tup)                #等价于 fun(1, 2, 3, 4)
04 fun(k1=1,**dict)           #等价于 fun(k1=1, k2=2, k3=3)
```

②调用函数时，实际参数必须顺序是：“变量=值”形式的参数在后面。例子：

```
01 def test(arg1, arg2=3, *args, arg3, **kwargs):
02     print('位置参数的值:', arg1)
03     print('默认参数的值:', arg2)
04     print('命名关键参数的值:', arg3)
05     print('可变参数的值:', args)
06     print('关键字参数的值:', kwargs)
07 test(1, 2, 3, 4, 5, 6, 7, k1=1, k2=2, arg3=8)
```

输出：

位置参数的值： 1

默认参数的值： 2

命名关键参数的值： 8

可变参数的值： (3 ,4, 5, 6, 7)

关键字参数的值： {k1 : 1, k2 : 2}

解释：首先按由前向后的顺序把具体值匹配给位置参数和默认参数，1 与 arg1 匹配，2 与 arg2 匹配，3, 4, 5, 6, 7 打包成元组匹配给 args；然后匹配命名关键字参数，8 匹配给 arg3；最后把其余的“键=值” k1=2、k2=2 打包成字典匹配给关键字参数 kwargs。

```
test(1,k1=1,k2=2,arg3=8)
```

输出：

位置参数的值： 1

默认参数的值： 3

命名关键参数的值： 8

可变参数的值： ()

关键字参数的值： {k1 : 1,k2 : 2}

解释：首先按由前向后的顺序，把具体值匹配给位置参数和默认参数，1 与 arg1 匹配，由于具体值只有 1 个，所以 arg2 使用默认值 3，args 为空；然后匹配命名关键字参数，8 匹配给 arg3；最后把其余的“键=值” k1=2、k2=2 打包成字典匹配给关键字参数 kwargs。

请你再编写一些测试用例自行测试。

4.3 return 语句

在函数体中使用 return 语句，可以使函数向调用语句返回值。语法格式如下：

```
return value
```

value: 返回值，可以是 1 个或多个用逗号 (,) 隔开的值。

例子：

```
def func_test(a,b):  
    c=a+b  
    d=a*b  
    e=a/b  
    f=a-b  
    return c, d, e, f
```

说明：

- (1) 在函数体中可以有多多个 return 语句；
- (2) 无论 return 语句在哪个位置，只要当程序执行 return 语句后，将结束函数体中后面代码的执行，释放函数中定义的所有变量，退出函数；
- (3) 当返回值为多个用逗号 (,) 隔开的值时，Python 会把这些值先包装成元组再返回，本质上还是返回一个值；
- (4) 当函数体中没有 return 语句时，默认返回 None。

4.4 递归函数

如果一个函数在内部又调用函数自己，这个函数就叫做递归函数。例子：

```
01 def func(n):
```

```

02 if n==1:
03     return 1
04 else:
05     return n*func(n-1)

```

这个函数的功能是求 $n!$ (n 的阶乘)。我们以计算 $4!$ 为例分析程序的执行逻辑。

```

func(4)                # 第 1 次调用自己
4 * func(3)           # 第 2 次调用自己
4* (3 * unc(2))       # 第 3 次调用自己
4 * (3 * (2 * func(1))) # 第 4 次调用自己, 由于 n 等于 1, 结束递归

```

从规模上看, 每递归一次相比上次递归都应有所减少, 直到满足某个条件时就结束递归调用。在本例中, n 等于 1 就是结束递归的条件。如果一个递归函数中没有结束递归的条件, 递归过程将一直继续下去, 类似于死循环。

递归函数的优点是定义简单, 逻辑清晰, 非常适合解决具有递推关系的问题, 缺点是占用资源较多, 且 Python 默认递归的最大深度为 979, 如果一个问题可能需要不止 979 层才能完成的话, 那就可以考虑使用循环实现。理论上, 所有的递归函数都可以写成循环的方式, 但循环的逻辑不如递归清晰。例子, 把计算 $n!$ 改用循环实现。

```

01 m = 1
02 i = 1
03 while i<=n:
04     m *= i
05     i += 1

```

4.5 匿名函数

匿名函数就是没有名字的函数, 使用关键字 `lambda` 定义, 语法格式如下:

```
result = lambda arg1, arg2, ……: expression
```

`arg1, arg2, ……` 叫做参数, 都可以省略, `expression`, 实现函数具体功能的表达式, 且只有一个表达式, 不能省略, 表达式的值就是匿名函数的返回值。

调用方法: `result(arg1, arg2, ……)`

例子:

```

>>>h=lambda a, b, c:a*b*c          #定义匿名函数
>>>h(2, 3, 4)                      #调用匿名函数

```

尽管匿名函数具有代码简捷易读的优点, 但是毕竟只有一个表达式, 只能实现有限的功能。其实, 匿名函数的主要用途是作为回调函数使用, 已超出本书范围, 不举例说明。

4.6 变量的作用域

Python 中, 所有的变量名都会按照定义时的位置被保存在不同的区域中, 这些区域叫做命名空间或者作用域。作用域分为四种类型: 局部作用域(Local)、嵌套作用域(Enclosing)、全局作用域(Global)、内置作用域(Built-in)。这四种作用域简称 LEGB。本节我们重点介绍局部作用域和全局作用域。

在运行函数时, 函数体中定义的所有变量将构成一个局部作用域, 模块(一个 `py` 文件)中已经执行的赋值语句, 将构成一个全局作用域。

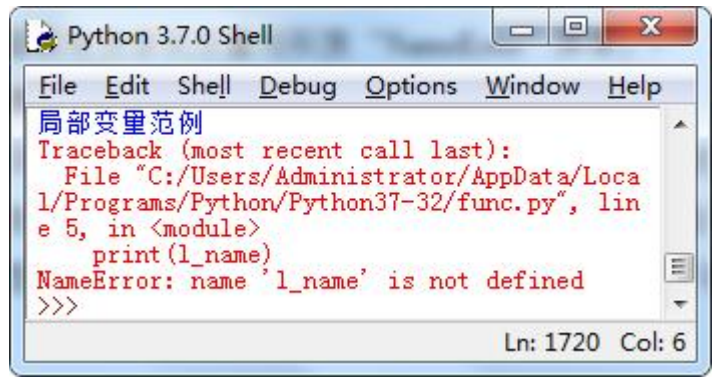
在访问变量时, 搜索路径遵循 LEGB 顺序。这里的 LEGB 顺序是指从当前所处区域开始依次搜索当前区域、上一级区域、……, 级别的优先级从高到低依次为 L、E、G、B。比如当前级别为 L, 则搜索顺序为 L、E、G、B; 如果当前级别为 E, 则搜索顺序为 E、G、B。一旦在某个作用域搜索到了变量则停止搜索, 如果一直没有搜索到变量则引发 “NameError” 异常。

4.6.1 局部变量

局部作用域中的变量叫做局部变量，在函数内部定义的变量都是局部变量，只在函数内部有效，在函数运行之前或在函数运行之后，这些变量都是不存在的。每个函数具有自己的作用域，因此，即使两个函数中存在名字相同的变量，也不是同一个变量。

例子：访问函数内部的变量。

```
01 def fun_test():
02     l_name = '局部变量范例'
03     print(l_name)
04 fun_test()
05 print(l_name)
```



4.6.2 全局变量

全局作用域中的变量叫做全局变量。在函数外定义的变量是全局变量，在函数内也可以访问全局变量。

例子：访问全局变量。

```
01 g_name = '全局变量范例'
02 def fun_test():
03     print(g_name)
04 fun_test()
05 print(g_name)
```

输出：

全局变量范例
全局变量范例

下面列举几个需要注意的例子，并总结经验。

(1) 局部变量与全局变量同名

```
01 book_name = 'Python 入门与实战'
02 def fun_test():
03     book_name = 'Python 教学系列'
04     print(book_name)
05 fun_test()
06 print(book_name)
```

输出：

Python 教学系列
Python 入门与实战

结论：当局部变量与全局变量同名时，按照 LEGB 规则，在函数内首先搜索到局部作用域里的变量，在函数外部首先搜索到全局作用域里的变量。

(2) 在函数中修改全局变量

```
01 book_name = 'Python 入门与实战'
02 def fun_test():
03     global book_name           #声明 book_name 为全局变量
04     book_name = 'Python 教学系列'
05     print(book_name)
06 fun_test()
07 print(book_name)
```

输出：

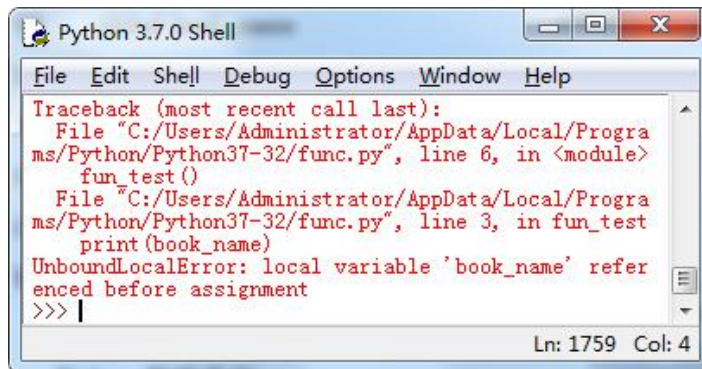
```
Python 教学系列
Python 教学系列
```

结论： 如果需要在函数中修改全局变量，须使用 global 关键字声明。

(3) 先访问局部变量，后定义局部变量

```
01 book_name = 'Python 入门与实战'
02 def fun_test():
03     print(book_name)
04     book_name = 'Python 教学系列'
05 fun_test()
```

运行错误：



为什么不是输出“Python 入门与实战”或“NameError”异常，而是“UnboundLocalError”异常？这是因为模块中的代码在执行之前，并不会经过预编译，但是函数体中的代码在运行前会经过预编译。换句话说，就是在函数被执行之前，Python 解释器就已经知道了函数中有哪些局部变量。所以在执行 03 语句时，Python 解释器知道 book_name 是局部变量，但并未赋值，就引发了该异常（在引用前需要先赋值）。

本章小结

本章全面介绍了 Python 自定义函数的基础知识，包括函数的创建、调用、参数的类型、返回值、变量的作用域等。这些技术涉及到许多细节，如值传递和引用传递、传递参数的顺序，不注意这些细节，尽管可以编写出没有语法错误的程序，但是会出现得不到预期结果的逻辑错误，建议在编写函数时自己多设计一些测试用例，测试函数的健壮性，逐步养成良好的编程习惯。对于递归函数，由于 Python 对递归函数的一些限制，重点是理解它的思想，在具体使用时一定要对规模进行预测评估。匿名函数在入门阶段了解即可。

练习题

1. 自定义一个无参函数，打印 'I am a student'。
2. 自定义一个有参函数，计算 x^2+y^2 。
3. 编写递归函数，计算 $1+2+3+4+\dots+n$ 。
4. 编写一个函数，计算 n 个数的最大值。
5. 自定义一个函数，已知圆的半径求圆的面积。
6. 水仙花数是指一个三位数，其各位数字立方和等于该数本身的数，请编写程序打印出 [100, 999] 之间的所有水仙花数。如 153 是一个水仙花数， $153=1^3+5^3+3^3$

第五章 数据结构

在实际工作中，我们往往需要处理具有一种或多种关系的一组数据，比如一个班级的学生成绩、某一时间段内天网摄像头拍摄的图像数据。在计算机科学中，把这些具有一种或多种关系的数据元素集合称为数据结构。可以把数据结构简单理解为存储数据的容器。Python 内置多种数据结构，在本章我们介绍 4 种基本数据结构：列表、元组、字典、集合。

5.1 索引与切片

5.1.1 索引

在操场上列队时，所有同学组成了一个有顺序的队列，经过报数，每个同学都有一个不同的数字，通过这个数字可以找到相应同学。在 Python 中，把这个队列称为序列，编号称为索引或下标。Python 提供两种索引方式：

0	1	2	n
---	---	---	-------	---

正数索引，从 0 开始递增

-n	-3	-2	-1
----	-------	----	----	----

负数索引，从 -1 开始递减

注意：当采用负数索引时，是从 -1 开始，而不是从 0 开始。
当访问一个元素时，既可以用正数索引，也可以用负数索引。

5.1.2 切片（分片）

使用索引可以访问序列中单个元素，但有时需要访问序列中指定范围内的元素，即序列的子序列，Python 定义了一个称为切片的操作符 [: :] 来得到子序列。

语法格式：[start_index: end_index: step]

表示从 start_index 索引对应的元素开始，每隔 step 个元素取出来一个，直到取到 end_index 对应的元素结束，但不包括 end_index 索引位置上的元素。

start_index 表示起始位置，end_index 表示终止位置，step 表示步长。

实践：在 IDLE 的交互模式下输入下列语句，观察结果并总结。

```
>>>li = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>li[:]
>>>li[0:5]
>>>li[0:8:]
>>>li[3::2]
>>>li[-1:-5:-1]
>>>li[:3:-1]
```

我们可以总结出：

- (1) start_index、end_index、step 任何一个都可以省略，step 的默认值为 1；
- (2) step 可以取正整数、负整数，但不能等于 0，step 为正整数时表示从左向右截取，step 为负整数时表示从右向左截取；
- (3) 当 step 为正整数时，start_index 的默认值为 0，end_index 的默认值为最后 1 个元素的索引；当 step 为负整数时，start_index 的默认值为 -1，end_index 的默认值为第 1 个元素的索引；
- (4) 每个元素都有两个索引，一个是从左向右的依次编号（从 0 开始），一个是从右向左的依次编号（从 -1 开始）。

5.2 列表（list）

列表与我们在操场上列队时类似，是由一系列按一定顺序排列的元素组成。只要用逗号把各个数据项使用方括号“[]”括起来就创建了一个列表。如：

```
ages = [13, 14, 16, 13, 15]
names = ['李晓', '张慧', '王姗姗']
students = ['lixiao', 14, 'zhanghui', 15]
```

Python 中列表与其它编程语言的数组非常类似，使用非常灵活。

5.2.1 列表的创建

Python 中可以有多种方法创建列表，下面分别介绍。

1. 使用赋值语句直接创建

语法格式如下：

```
listname = [element1, element2, ……]
```

其中，listname 表示列表名，element1, element2 表示列表的元素。listname 可以是任何符合 Python 命名规则的变量名。列表中的元素可以是不同的数据类型，也可以是列表等 Python 支持的其它数据类型，元素的个数没有限制。

例子：

```
school_name = ['贵阳一中', '铜仁一中', '凯里一中', '思南中学', '德江一中']
student_score = ['语文', 90, '数学', 80, '英语', 97]
family_member = ['父亲', ['王进', 45], '母亲', ['张霞', 43], '哥哥', ['王小宝', 16]]
```

2. 创建空列表

创建空列表非常简单，直接使用下面的代码：

```
listname = [] 或 listname = list()
```

3. 使用 list() 函数创建

语法格式如下：

```
listname = list(data)
```

data 表示可以转换为列表的对象，如 range 对象、字符串等其它任何可迭代对象。

例子：

```
age = list(range(1, 5))
```

将创建 [1, 2, 3, 4] 列表。

```
name = "xiaofang"
```

```
names = list(name)
```

将创建 ['x', 'i', 'a', 'o', 'f', 'a', 'n', 'g'] 列表

4. 列表推导式

使用推导式可以快速生成一个列表，是一种运用较多而又非常重要的功能，同时也是最受欢迎的 Python 特性之一。

基本语法格式如下：

```
[表达式 for 变量 in 迭代对象] 或者 [表达式 for 变量 in 迭代对象 if 条件]
```

例子：

```
age = [i for i in range(1, 120)]
```

将创建一个列表，里面的元素分别为 1, 2, 3, ... 119，注意不包括 120。

```
age = [i for i in range(1, 120) if i%2==0]
```

将创建一个 1 到 120 之间的偶数列表，注意不包括 120。

说明：在使用列表推导式创建列表时，执行完推导式语句后，将在内存中立即生成列表，如果元素个数很多，会占用大量内存空间，建议使用生成器表达式。

5.2.2 列表的访问

1. 访问单个元素

Python 中列表是一种序列，元素是有顺序的，都拥有自己的编号，通常情况我们是通过索引对元素进行访问。

如 `li = [1, 1, 2, 3, 5, 8]`，`li[0]`将访问列表中第 1 个元素，`li[3]`将访问列表中第 4 个元素，`li[-2]`将访问列表中倒数第 2 个元素。

2. 遍历列表

遍历是计算机科学中的一种重要运算，是指依照某种顺序对所有元素做一次且仅做一次访问。比如需要找出班级里身高最高的同学，就需要测量每个同学的身高。测量身高的过程就相当于对列表进行遍历。下面介绍三种遍历列表的方法。

(1) 使用 for 循环实现

语法格式：

```
for item in listname:
```

```
    #输出 item
```

其中，`item`用于保存依次从列表中获取到的元素的值，`listname`列表名。

例子：定义编程语言列表 `program_language = ['java', 'c', 'c++', 'Python', 'c#']`，通过 for 循环遍历，输出各种编程语言的名称，代码如下：

```
program_language = [ 'java', 'c', 'c++', 'Python', 'c#' ]
```

```
for pname in program_language:
```

```
    print(pname)
```

程序运行后，将得到如图 5.1 所示的结果。

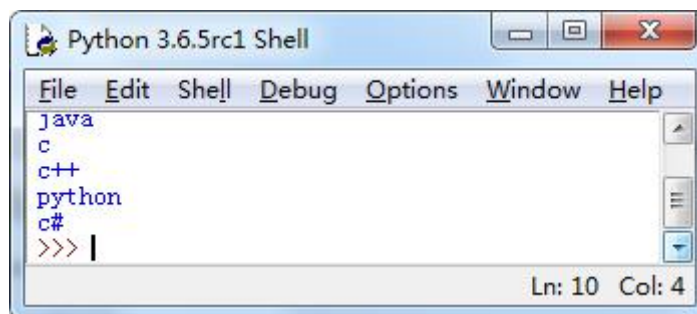


图 5.1

(2) 使用 for 循环和索引实现

把上面的代码修改为：

```
program_language = [ 'java', 'c', 'c++', 'Python', 'c#' ]
```

```
for i in range(len(program_language)):
```

```
    print(program_language[i])
```

程序运行结果如图 5.1。

(3) 使用 for 循环和 enumerate() 函数实现

把上面的代码修改为：

```
program_language = [ 'java', 'c', 'c++', 'Python', 'c#' ]
```

```
for index, item in enumerate(program_language):
```

```
    print(index, item)
```

程序运行后，将得到如图 5.2 所示的结果。

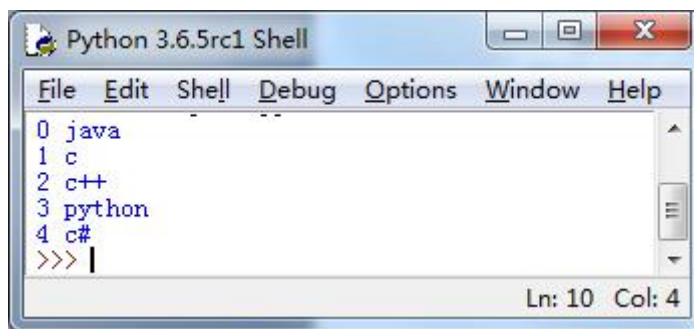


图 5.2

可以看出，使用 `enumerate()` 函数可以同时获得列表元素的索引和值。

3. 获得子列表

获得子列表的简单方法就是使用切片，在 IDLE 的交互模式下运行如下代码：

```
>>>li = [1, 2, 3, 4, 5, 6]
>>>li[0]
>>>1
>>>li[0:1]
>>>[1]
```

你能总结出 `li[0]` 与 `li[0:1]` 的区别吗？

5.2.3 对列表元素增加、删除、修改操作

对数据对象进行增、删、改、查是计算机科学中的基本操作。在实际开发过程中，很大一部分都是基于增、删、改、查操作。

1. 增加元素

列表对象提供了三种增加元素的方法，分别是 `append()` 方法、`insert()` 方法、`extend()` 方法。注意，这三种方法是列表对象提供的方法，不是内置函数。

方法名称	语法	描述
<code>append</code>	<code>listname.append(obj)</code>	向 <code>listname</code> 的末尾增加 <code>obj</code> 元素
<code>insert</code>	<code>listname.insert(index, obj)</code>	向 <code>listname</code> 的 <code>index</code> 位置插入 <code>obj</code> 元素
<code>extend</code>	<code>listname.extend(obj)</code>	将 <code>obj</code> 列表中的元素增加到 <code>listname</code> 中

实践：在 IDLE 的交互模式下执行下列代码，体会三种方法的异同。

```
>>>num = [1, 2, 3, 4, 5]           #创建列表 num
>>>num.append(6)                  #使用 append() 方法增加元素 6
>>>num                            #输出 [1, 2, 3, 4, 5, 6]
>>>num.append([7, 8])            #使用 append() 方法增加元素 [7, 8]
>>>num                            #输出 [1, 2, 3, 4, 5, 6, [7, 8]]
>>>num.insert(1, 9)              #使用 insert() 方法在索引 1 处增加元素 9
>>>num                            #输出 [1, 9, 2, 3, 4, 5, 6, [7, 8]]
>>>num.extend([10, 11])          #使用 extend() 方法增加元素 10, 11
>>>num                            #输出 [1, 9, 2, 3, 4, 5, 6, [7, 8], 10, 11]
```

我们可以总结出：

- (1) `append(obj)` 方法总是把 `obj` 对象作为一个元素，追加到列表的末尾；
- (2) 使用 `insert()` 方法可以在不同位置插入元素；
- (3) `extend()` 方法是先把 `obj` 当作序列，枚举出每个元素，然后再追加到列表的末尾。

2. 删除元素

需要删除列表中的元素时，可以使用内置语句 `del` 或列表对象的 `remove()` 方法、`pop()` 方法。

`del` 语句、`pop()` 方法是根据元素的索引进行删除，`remove()` 方法是根据元素值进行删除，且只删除第一个满足条件的元素。

例子：

```
>>>tea = ['都匀毛尖', '湄潭翠芽', '正安白茶', '石阡苔茶', '梵净山翠峰茶']
>>>del tea[1]                #根据索引删除'湄潭翠芽'
>>>tea.remove('都匀毛尖')   #根据值删除'都匀毛尖'
>>>tea.pop(0)                #根据索引删除'正安白茶'
```

说明：列表对象的 `clear()` 方法将清除所有元素。

3. 修改元素

修改元素只需要通过索引获取相应元素，然后重新赋值即可。如：在上例中需要将‘梵净山翠峰茶’改为‘雷公山银球茶’，则修改的代码为：`tea[4] = '雷公山银球茶'`。

5.2.4 列表对象的常用方法

方法名称	语法	描述
<code>count</code>	<code>listname.count(obj)</code>	返回元素 <code>obj</code> 在列表中出现的次数
<code>index</code>	<code>listname.index(obj)</code>	返回元素 <code>obj</code> 在列表中首次出现的索引
<code>sort</code>	<code>listname.sort(Key=None, reverse=False)</code>	对列表排序
<code>reverse</code>	<code>listname.reverse()</code>	将列表翻转

知识拓展：Python 中一切皆为对象，几乎所有对象都有成员属性、方法属性，访问对象的成员和方法统一使用“对象.成员”或“对象.方法”格式，关于类和对象的详尽知识将在第六章阐述。

5.2.5 排序与查找算法介绍

有一个猜数游戏：甲先想好一个小于 1000 的自然数，乙的任务是猜出这个数。乙每猜一个数，甲会说“大了”、“小了”或“正确”。如果你是乙，你能保证在十次之内猜中吗？

将一组数据按从小到大（或从大到小）的顺序排列就称为排序，实现排序的方法就称为排序算法。虽然 Python 已经为我们提供了 `sort()` 方法用于排序，但是排序方法是如何工作的呢？下面介绍几种经典的排序算法和二分查找法。

假设列表 `li = [33, 11, 23, 67, 46, 2]`，现要求将列表变为 `li = [2, 11, 23, 33, 46, 67]`

1. 冒泡排序

算法分析：依次比较相邻的两个数，将小数放在前面，大数放在后面。

第一趟：首先比较第 1 个和第 2 个数，将小数放前，大数放后。然后比较第 2 个数和第 3 个数，将小数放前，大数放后，以此类推，直到比较最后两个数，将小数放前，大数放后。最后一个数为最大数。

第二趟：首先比较第 1 个和第 2 个数，将小数放前，大数放后。然后比较第 2 个数和第 3 个数，将小数放前，大数放后，以此类推，直到比较除最后一个数外的两个数，将小数放前，大数放后。

重复以上过程，直到最后一趟只比较第 1 个和第 2 个数时为止。

实现代码：

```

01 def bubble_sort(li):
02     """冒泡排序算法"""
03     n=len(li)-1
04     for i in range(n):
05         for j in range(n-i):
06             if li[j] > li[j+1]:
07                 li[j] , li[j+1] = li[j+1] , li[j]           #交换两个数
08 li = [33, 11, 23, 67, 46, 2]
09 bubble_sort(li)
10 print(li)

```

第一趟结果: li = [11, 23, 33, 46, 2, 67]

第二趟结果: li = [11, 23, 33, 2, 46, , 67]

第三趟结果: li = [11, 23, 2, 33, 46, , 67]

第四趟结果: li = [11, 2, 23, 33, 46, , 67]

第五趟结果: li = [2, 11, 23, 33, 46, , 67]

2. 选择排序

算法分析: 首先比较第 1 个和第 2 个数, 将小数放前, 大数放后, 然后比较第 1 个数和第 3 个数, 将小数放前, 大数放后, 如此继续, 直至比较第 1 个数和最后 1 个数为止。得到最小的一个元素, 存放在第一个位置。然后用第 2 个数与后面的所有数比较, 得到第二小的元素, 存放在第二个位置, 重复这个步骤, 直到全部待排序的数据元素排完。即第一趟找到最小(最大)元素放到第 1 个位置, 第二趟找到第二小(大)的元素放到第 2 个位置, ……

实现代码:

```

01 def select_sort(li):
02     """选择排序算法"""
03     n=len(li)
04     for i in range(n):
05         for j in range(i+1, n):
06             if li [i] > li[j]:
07                 li[i] , li[j] = li[j] , li[i]
08 li = [33, 11, 23, 67, 46, 2]
09 select _sort(li)
10 print(li)

```

第一趟结果: li = [2, 33, 23, 46, , 67, 11]

第二趟结果: li = [2, 11, 23, 46, , 67, 33]

第三趟结果: li = [2, 11, 23, 46, , 67, 33]

第四趟结果: li = [2, 11, 23, 33, , 67, 46]

第五趟结果: li = [2, 11, 23, 33, , 46, 67]

在每次比较时，如果前面的数大于后面的数，则交换，我们可以修改为不交换，只记录位置，待一趟比较完成后，再交换，这样每一趟只交换 1 次，可以缩短运行时间，请你试试怎样修改代码？

3. 插入排序

算法分析：

第一趟：比较第 2 个元素和第 1 个元素，将小数放前，大数放后；第二趟：比较第 3 个元素和第 2 元素，将小数放前，大数放后，再比较第 2 个元素和第 1 元素，将小数放前，大数放后；第三趟：比较第 4 个元素和第 3 个元素，将小数放前，大数放后，再比较第 3 个元素和第 2 元素，将小数放前，大数放后，再比较第 2 个元素和第 1 元素，将小数放前，大数放后。

在比较时，如果后一元素不小于前一元素，则本趟结束。

如此继续，直到全部待排序的数据元素排完。

实现代码：

```
01 def insert_sort(li):
02     """插入排序算法"""
03     n = len(li)
04     for i in range(1,n):
05         for j in range(i, 0, -1):
06             if li[j] < li[j-1]:           #如果后一个数小于前一个数
07                 li[j],li[j-1] = li[j-1],li[j]   #交换
08             else:                           #否则本趟结束
09                 break
10 li = [33,11,23,67,46,2]
11 insertt _sort(li)
12 print(li)
```

第一趟结果：li = [11, 33, 23, 46, , 67, 2]

第二趟结果：li = [11, 23, 33, 46, , 67, 2]

第三趟结果：li = [11, 23, 33, 46, , 67, 2]

第四趟结果：li = [11, 23, 33, 46, , 67, 2]

第五趟结果：li = [2, 11, 23, 33, 46, , 67]

4. 快速排序

算法分析：以第 1 个元素为比较对象，把后面凡是比第 1 个元素大的排在右边，凡是比第 1 个元素小的排在左边，然后分别对左边和右边进行递归排序。

第一轮：首先从最后 1 个元素开始依次与第 1 个元素比较，如果找到了比第 1 个元素小的时候就停下来（假设此时位置为 j），然后从前面第 1 个元素开始依次与第 1 个元素比较，如果找到了比第 1 个元素大的时候就停下来（假设此时位置为 i），交换位置 i 和位置 j 的 2 个元素，接下来从位置 j 开始依次与第 1 个元素比较，如果找到了比第 1 个元素小的时候又停下来，然后从位置 i 开始依次与第 1 个元素比较，如果找到了比第 1 个元素大的时候又停下来，再次交换，如此继续，直到 i 和 j 相等时，把第 1 个元素与第 i（或 j）个元素交换。

第二轮：按照第一轮的方法，分别把左边和右边进行排序，直到全部排完为止。

实现代码：

```
01 def quick_sort(li, low, hight):
02     """快速排序算法"""
03     if low >= hight:
04         return
05     i = low
06     j = hight
07     while i < j:
08         while i < j and li[j] >= li[low]:
09             j -= 1
10         while i < j and li[i] <= li[low]:
11             i += 1
12         li[i], li[j] = li[j], li[i]
13     li[low], li[i] = li[i], li[low]
14     quick_sort(li, low, i-1)
15     quick_sort(li, i+1, hight)
16 li = [33, 11, 23, 67, 46, 2]
17 quick_sort(li, 0, len(li)-1)
18 print(li)
```

第一轮结果：li = [2, 11, 22, 33, , 46, 67]

第二轮结果：li = [2, 11, 22, 33, 46, 67]

快速排序的思想是将原问题划分成几个小问题，然后递归地解决这些小问题，最后综合它们的解得到问题的解，这就是分治思想。快速排序是最快的通用内部排序算法，有多种实现方法，但基本思想都是通过一趟比较后，这些数据被分成了3个部分，左边、中间（1个数）、右边，中间这个数的位置已经确定。然后再递归对左边、右边进行排序。

我们可以得到这样的启发，如果是要求这些数按从小到大排序的话，那么第一趟结束后，就知道了中间这个数是第几小。

实践：有 n 个不同的正整数，请找出这些数中第 k ($1 \leq k \leq n$) 小的数。换个说法：有 n 个不同的正整数，请找出这些数按从小到大排序后的第 k 个数 ($1 \leq k \leq n$)。

5. 二分查找法

算法分析：二分查找也称折半查找，是一种效率较高的查找方法。但仅适用于已经排好序的序列。其基本思路是：首先将给定值 K 先与序列中间位置元素比较，若相等，则查找结束；若不等，则根据 K 与中间元素的大小，确定是在前半部分还是后半部分中继续查找。这样逐渐缩小范围进行同样的查找。如此反复，直到找到（或查找范围的长度为0）为止。

我们来模拟前面的猜数游戏，实现代码：


```

01 import random
02 total = 0
03 expr = ["甲得意地说：", "甲做了个鬼脸说：", "甲高高蹦了一下说："]
04 def search(li, k):
05     """二分法查找"""
06     global total
07     total +=1
08     mid = len(li) // 2
09     if len(li) > 0:
10         if k == li[mid]:
11             print("乙猜的数：", li[mid], "甲低下头，小声说：猜中了")
12             return True
13         elif k < li[mid]:
14             print("乙猜的数：", li[mid], expr[random.randrange(0, len(expr))]+"大了")
15             return search(li[:mid], k)
16         else:
17             print("乙猜的数：", li[mid], expr[random.randrange(0, len(expr))]+"小了")
18             return search(li[mid+1:], k)
19     else:
20         return False
21 li = [i for i in range(1,1000)]
22 answer = random.randrange(1, 1000)
23 print("甲想的数：", answer)
24 search(li, answer)
25 print("共：", total, "次")

```

5.2.6 动手实践

1. 现有列表 numlist=[3, 2, 14, 18, 88, 34, 76, 9, 10]，请计算列表的最大跨度值(最大跨度值 = 最大值-最小值)。

分析：要求列表的最大跨度值，必须先求出列表元素的最大值和最小值，因此必须遍历列表。对列表进行两次遍历，分别求出最大值和最小值。代码如下：

```

01 numlist = [3, 2, 14, 18, 88, 34, 76, 9, 10]
02 maxvalue = numlist[0]
03 minvalue = numlist[0]
04 for item in numlist:
05     if item > maxvalue:
06         maxvalue = item

```

```

07 for item in numlist:
08     if item < minvalue:
09         minvalue = item
10 span = maxvalue- minvalue
11 print(span)

```

在上面的代码中对列表进行了 2 次遍历，如果列表很长，消耗在遍历列表的时间也会很长，如果实际问题对运行时间有严格要求，我们需要设计出尽可能优雅的代码，下面是改进的代码。

```

01 numlist = [3, 2, 14, 18, 88, 34, 76, 9, 10]
02 maxvalue = numlist[0]
03 minvalue = numlist[0]
04 for item in numlist:
05     if item > maxvalue:
06         maxvalue = item
07     if item < minvalue:
08         minvalue = item
09 span = maxvalue- minvalue
10 print(span)

```

通过改进后，对列表只进行 1 次遍历即可得出最大值和最小值，从而大幅度地缩短了运行时间。仔细分析程序运行过程，我们可以对 07 行代码稍加修改还可以进一步缩短程序运行时间，你知道怎么修改吗？

实际上，Python 提供了 `max()` 和 `min()` 两个内置函数，可以直接求出列表元素的最大值和最小值，最后我们将代码修改为：

```

01 numlist = [3, 2, 14, 18, 88, 34, 76, 9, 10]
02 maxvalue = max(numlist)
03 minvalue = min(numlist)
04 span = maxvalue- minvalue
05 print(span)

```

Python3.x 提供了一些内置函数，这些函数是由 Python 工程师精心编写的，无论在速度上还是安全性上往往比我们自己设计要高效得多，在实际开发中，优先使用这些内置函数是一个明智的选择。

2. 陶陶摘苹果：陶陶家的院子里有一棵苹果树，每到秋天树上就会结出 10 个苹果。苹果成熟的时候，陶陶就会跑去摘苹果。陶陶有个 30 厘米高的板凳，当她不能直接用手摘到苹果的时候，就会踩到板凳上再试试。

现在已知 10 个苹果到地面的高度（100, 200, 150, 140, 129, 134, 167, 198, 200 111）（以厘米为单位），以及陶陶把手伸直的时候能够达到的最大高度为 110 厘米，请帮陶陶算一下她能够摘到的苹果的数目。假设她碰到苹果，苹果就会掉下来。

分析：淘淘在摘苹果时首先会直接伸手去摘，如果不能直接用手摘到，他会尝试踩到板凳上再尝试，如果还不能摘到就会放弃。根据这个思路，我们把苹果到地面的高度构建为列表，然后遍历列表，依次判断每个苹果的高度是否大于 110，如果是则判断是否大于 140。代码如下：

```

01 apple_highs = [100, 200, 150, 140, 129, 134, 167, 198, 200, 111]
02 total = 0
02 for high in apple_highs:
03     if high<=110:
04         total+=1
05     elif high<=140:
06         total+=1
07 print(total)

```

优化后的代码（请分析优化依据）：

```

01 apple_highs = [100, 200, 150, 140, 129, 134, 167, 198, 200, 111]
02 total = 0
02 for high in apple_highs:
03     if high<=140:
04         total+=1
05 print(total)

```

在解决这个问题过程中，采用的是将每个苹果的高度进行条件验证（判断是否小于或等于 140），符合条件的累加，不符合条件的舍弃，从而求解。我们把这种解决问题的方法称为枚举算法，即将问题的所有可能的答案一一列举，然后根据条件判断此答案是否合适，保留合适的，舍弃不合适的。

3. 删除单词后缀：从键盘不断读入单词，如果该单词以 er 或者 ly 后缀结尾，则删除该后缀，否则不进行任何操作，以“end”结束输入，最后输出所有单词。

分析：为了将所有输入的单词保存，需要创建一个容器，这里用列表实现。然后逐一读入，每读入一个单词，依次判断是否是结束标志“end”，是否以“er”或“ly”结尾，如果是则使用切片截取，动态增加到列表。最后遍历列表输出所有单词。

```

01 word_list = [ ]
02 word = input("请输入单词：")
03 while word!="end":
04     if word.endswith("er") == True or word.endswith("ly") == True:
05         word = word[:-2]
06     word_list.append(word)
07     word = input("请输入单词：")
08 for word in word_list:
09     print(word, end=' ')

```

endswith()方法是字符串对象的方法，用于判断字符串是否以指定后缀结尾，如果以指定后缀结尾返回 True，否则返回 False。

如果把条件修改为以 er 或者 ing 后缀结尾，要如何修改代码？

4. 最长平台：已知一个元素已经从小到大排序的列表，这个列表的一个平台（Plateau）就是连续的一串值相

同的元素，并且这一串元素不能再延伸。例如，在 1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 6 中 1, 2-2, 3-3-3-3, 4, 5-5, 6 都是平台。试编写一个程序，接收一个列表，把这个列表最长的平台找出来。在上面的例子中 3-3-3-3 就是最长的平台。

输入样例：1, 2, 2, 3, 3, 3, 3, 4, 5, 5, 6

输出样例：4

分析：显然，该问题需要遍历列表，由于列表元素已经按从小到大排序，我们只需判断当前元素是否与前一元素相等，若相等，当前平台长度加 1，不相等，表明当前平台已经结束，计算出到目前为止最长平台的长度，然后把下一平台长度初始为 1。实现代码如下：

```
01 maxlen=1 #最大平台长度
02 curlen=1 #当前平台长度
03 pre=None
04 str = input("请输入一个用逗号分隔的字符串：")
05 li = str.split(",") #把字符串转化为列表
06 for num in li:
07     if pre==num:
08         curlen+=1
09     else:
10         maxlen = max(maxlen, curlen)
11         curlen = 1
12     pre = num
13 maxlen=max(maxlen, curlen) #如果只有 1 个平台
14 print(maxlen)
```

我们知道，选择排序的思想是，每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放到序列的起始位置，直到全部排完，请试着比较本题的实现过程与选择排序的思想。

5. 生成列表：给定参数 n，生成顺序随机、值不重复的含有 n 个元素，值为 $1 \sim n$ 的列表。

输入样例：6

输出样例：[5, 3, 1, 4, 6, 2]

分析：先定义一个空列表，产生随机数，判断这个数是否在列表中，如果在列表中，重新产生，否则添加到列表，如此继续，直到列表元素个数等于 n。实现代码如下：

```
01 import random #导入产生随机数模块
02 li = list()
03 count = 0
04 n = int(input())
05 while count < n:
06     tempint = random.randint(1, n) #产生随机数
07     if tempint not in li:
```

```
08         li.append(tempint)
09         count+=1
10 print(li)
```

实际上，使用 `random.sample(range(1, n+1), n)` 可以直接产生满足要求的列表。Python 中由对象提供的方法已完成很多功能，这是 Python 得以流行的一个重要原因之一。

5.3 元组 (tuple)

元组与列表类似，也是由一系列按一定顺序排列的元素组成。只要用逗号把各个数据项使用圆括号“()”括起来就创建了一个元组。如：`tup = (1, 2, 3, 4)`。从形式上可以看出，元组与列表非常相似，元组使用圆括号包裹，列表使用方括号包裹，它们之间的主要区别是：元组为不可变类型，列表为可变类型。

5.3.1 可变类型与不可变类型

当我们用铅笔在纸上写字时，如果写错了还可以擦掉重写，但用钢笔在纸上写字时，如果写错了就不能擦掉重写。这是可变与不可变的简单比喻。Python 中可变是指序列中的元素是可以改变的，不可变是指序列中的元素是不可以改变的。

实践，在 IDLE 的交互模式下执行下列代码：

```
>>>li = [1, 2, 3, 4]
>>>li[1] = 3
>>>li
>>>tup = (1, 2, 3, 4)
>>tup[1] = 3
```

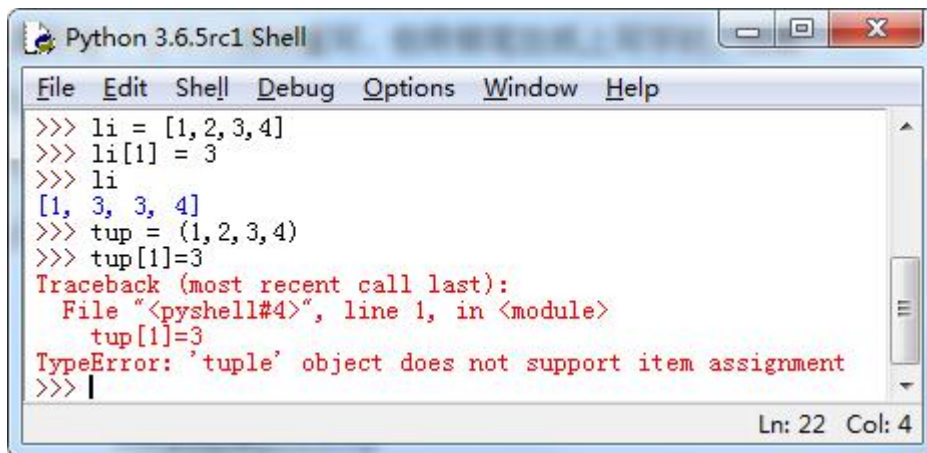


图 5.3 修改不可变类型

图 5.3 的错误提示信息为：元组对象不支持对元素赋值。在 Python 的基础数据类型中，数值、字符串、元组和不可变集合都是不可变类型，列表、字典和可变集合是可变类型。

再如：字符串 `str = "abcdefg"`，由于字符串是不可变类型，所以不能通过 `str[0] = 'h'` 来修改字符串的内容。

5.3.2 元组的创建

Python 中可以有多种方法创建元组，下面分别介绍。

1. 使用赋值语句直接创建

语法格式如下：

```
tuplename = (element1, element2, ……)
```

其中，tuplename 表示元组名，element1, element2 表示元组的元素。tuplename 可以是任何符合 Python 命名规则的变量名。元组中的元素可以是不同的数据类型，也可以是元组等 Python 支持的其它数据类型，元素的个数没有限制。

例如：

```
age = (1, 15, 23, 90, 45)
```

```
street = ("陕西路", 150, "延安路", 85, "中华北路", 30)
```

```
region = ('东北', ('黑龙江', '吉林', '辽宁'), '华东', ['山东', '江苏', '安徽', '浙江', '福建', '上海'])
```

与列表不同的是，元组中的小括号并不是必须的。只要把一组数据用逗号“,” 隔开，Python 会自动识别为元组。如：fruits = '苹果', '香蕉', '橘子', '西瓜', '葡萄' 是合法的。

如果元组只有 1 个元素，在创建时需要在后面加上 1 个“,”，例如：book = ('语文',)。

这是因为 () 具有表示改变运算顺序的含义，如果没有“,” Python 会将 ('语文') 解释为字符串。

说明：在变量声明时不需要指定数据类型，同一个变量在不同时刻可以指向不同的数据类型，可以使用内置函数 type() 判断变量的类型。如下图所示：

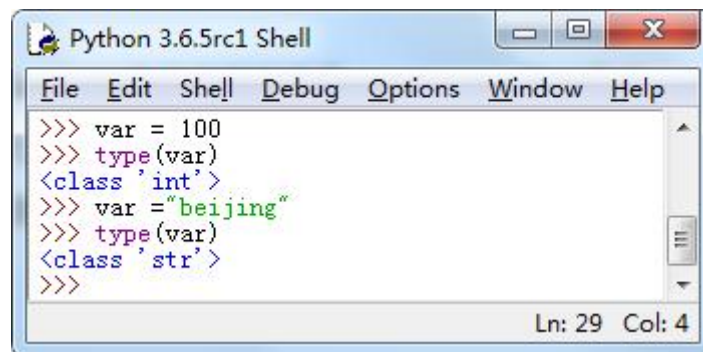


图 5.4 测试变量的类型

2. 创建空元组

创建空元组，直接使用代码：tuplename = () 或 tuplename = tuple()。

3. 使用 tuple() 函数创建

语法格式如下：

```
tuplename = tuple(data)
```

data 表示可以转换为元组的对象，如 range 对象、字符串等其它任何可迭代对象。

例子：

```
age = tuple(range(2, 10))
```

将创建 (2, 3, 4, 4, 5, 6, 7, 8, 9) 元组。

```
name = "xiaofang"
```

```
names = tuple(name)
```

将创建 ('x', 'i', 'a', 'o', 'f', 'a', 'n', 'g') 元组

```
li = [6, 7, 8, 9]
```

```
tup = tuple(li)
```

将创建(6, 7, 8, 9) 元组

5.3.3 元组的访问

访问元组与列表类似，下面简单说明：

1. 访问单个元素

使用索引访问，如 `tup = (1, 1, 2, 3, 5, 8)`，`tup[0]`将访问元组中第 1 个元素，`tup(3)`将访问元组中第 4 个元素，`tup(-2)`将访问元组中倒数第 2 个元素。

2. 遍历元组

(1) 使用 for 循环实现

```
words = ('I', 'am', 'a', 'student')
for word in words:
    print(word, end=' ')
```

输出为： I am a student

(2) 使用 for 循环和 enumerate() 函数实现

```
words = ('I', 'am', 'a', 'student')
for item in enumerate(words):
    print(item, end=' ')
```

输出为： (0, ' I') (1, ' am') (2, ' a') (3, ' student')

```
words = ('I', 'am', 'a', 'student')
for index, word in enumerate(words): #这里有解包过程
    print(index, word, end=' ')
```

输出为： 0 I 1 am 2 a 3 student

说明：在赋值语句中，如果左边有多个变量，Python 需要对右边的表达式先进行解包（每个元素单独提取出来），然后再逐一赋值。如果变量的个数与解包后值的个数不相等则会报错。

5.3.4 对元组的操作

元组属于不可变类型，不能对其元素进行增加、删除、修改操作，但可以进行连接组合，即把两个元组进行相加操作，也可以进行相乘（重复）操作。例子：

```
>>>zoo1 = ('老虎', '狮子')
>>>zoo2 = ('梅花鹿', '大象')
>>>zoo1 += zoo2
>>>print(zoo1)
输出为： ('老虎', '狮子', '梅花鹿', '大象')
```

```
>>>zoo2 = zoo2*3
>>>print(zoo2)
输出为： ('梅花鹿', '大象', '梅花鹿', '大象', '梅花鹿', '大象')
```

5.3.5 元组对象的常用方法

元组对象只有极少的方法，count()、index()等，这两个方法的意义同列表。

5.3.6 列表与元组的比较

项目	列表	元组
可变类型	可变	不可变
元素增、删、改操作	能	不能
支持索引访问	支持	支持
支持切片	支持	支持
访问和处理速度	相对较慢	相对较快
推导式	有	没有
作为字典的键	不能	能
支持 in 和 not in	支持	支持

元组与列表有很多相似的地方，为什么有了列表还要元组呢？由于元组的不可变性提供了数据的完整性，这样可以确保元组在程序中不会被另一个引用修改，而列表就没有这样的保证。元组也可以用在列表无法使用的地方。例如，作为字典的键，一些内置操作可能也要求或暗示要使用元组而不是列表。

判断元素是否在列表或元组中，使用 in 和 not in 关键字。例子：

```
>>>li = [8, 20, 15]
```

```
>>>8 in li
```

输出：True

```
>>>22 not in li
```

输出：True

5.4 字典 (dict)

在实际应用中，列表往往用来存储一组性质相同的数据，如学生姓名、课程名称等，元组多用于存储一系列不可变的结构，如棋盘坐标。而实际生活中还有类似这种需求：描述一个人的基本特征（姓名、年龄、身高、体重）。若用列表可表示为：

person = ['张粟', 18, 1.75, 55]，显然，从字面上并不能清晰地理解元素 18、1.75、55 表示的意义。如果写为 name: '张粟', age: 18, height: 1.75, weight: 55，这些数字的意义就明确得多。

Python 把像上面这样用“键:值”对的形式来存储数据的容器称为字典。本节我们介绍字典的相关知识。

5.4.1 字典的基本特征

形式：dictname = {key1:value1, key2:value2, ……}， dictname 字典名，key1:value1 键值对。键和值之间用“:” 隔开，键值对与键值对之间用“,” 隔开，用“{ }” 包裹所有键值对。基本特征如下：

- (1) 键必须是不可变类型且具有唯一性。可以使用数字、字符串、元组作为键；
- (2) 字典是无序的，不支持用索引访问；
- (3) 字典的值可以是任何 Python 支持的数据类型，且可以任意嵌套；
- (4) 与列表和元组比较，字典有更快的检索速度。

5.4.2 字典的创建

1. 使用赋值语句直接创建

语法格式如下：

```
dictname = {key1:value1, key2:value2, .....}
```

其中，dictname 表示字典名，key1, key2 表示元素的键。value1, value2 表示元素的值。

2. 创建空字典

创建字典非常简单，直接使用下面的代码：

```
dictname = { } 或 dictname = dict( )
```

3. 通过映射函数创建

语法格式如下：

```
dictname = dict(zip(iterable1, iterable2))
```

iterable1, 一个可迭代对象，用于指定要生成字典的键。iterable2, 一个可迭代对象，用于指定要生成字典的值。这里不必纠结什么是可迭代对象，暂且简单理解为列表或元组。

例子：

```
name = ( 'c' , 'c++' , 'Python' , 'java' )
```

```
price = [30, 45, 50.5, 27]
```

```
book_info = dict(zip(name, price))
```

将创建{ 'c' :30, 'c++' :45, ' Python' :50.5, ' java' :27} 字典。

说明：

(1) zip()是Python的内置函数，功能是将多个可迭代对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表，元组的个数与最短的可迭代对象的长度相同。

如：

```
>>>li = list(zip([1, 2, 3], [4, 5, 6]))
```

```
>>>[(1, 4), (2, 5), (3, 6)]
```

```
>>>li = list(zip([1, 2, 3, 4], [5, 6], (7, 8, 9)))
```

```
>>>[(1, 5, 7), (2, 6, 8)]
```

(2) dict()是Python的内置函数，用于创建一个字典。

4. 通过“键=值”的形式创建

语法格式如下：

```
dictname = dict(key1=value1, key2=value2, ...)
```

key1, key2 表示元素的键，value1, value2 表示元素的值。

例子：

```
arms = dict( '唐僧' ='咒语' , '孙悟空' ='金箍棒' , '猪八戒' ='钉耙' )
```

将创建{ '唐僧' : '咒语' , '孙悟空' : '金箍棒' , '猪八戒' : '钉耙' } 字典。

5. 通过 dict 对象的 fromkeys() 方法创建值为空的字典

语法格式如下：

```
dictname = dict.fromkeys(iterable1)
```

iterable1 为字典键的可迭代对象。

例子：

```
name = [ '李宏彦' , '马云' , '马化腾' , '刘庆峰' ]
```

```
batk = dict.fromkeys(name)
```

将创建{'李宏彦': None, '马云': None, '马化腾': None, '刘庆峰': None} 空字典。

5. 通过字典推导式创建

字典推导式与列表推导式类似，基本语法格式如下：

{键表达式:值表达式 for 变量 in 列表} 或者 {键表达式:值表达式 for 变量 in 列表 if 条件}

实践：在 IDLE 的交互模式下执行下面的代码，并认真体会。

```
>>>num = {i:i*i for i in range(1,5)}
```

```
>>>num
```

输出 {1:1, 2:4, 3:9, 4:16}

```
>>> even = {i:i*i for i in range(1,10) if i%2==0}
```

```
>>>even
```

输出 {2: 4, 4: 16, 6: 36, 8: 64}

```
>>>name = ('钱学森', '邓稼先', '华罗庚', '李四光') #用于键的元组
```

```
>>>area = ('上海', '安徽', '江苏', '湖北') #用于值的元组
```

```
>>>scientist_1 = {i:j for i,j in zip(name,area)}
```

```
>>>scientist_1
```

输出 {'钱学森': '上海', '邓稼先': '安徽', '华罗庚': '江苏', '李四光': '湖北'}

```
scientist_2 = {name[i]:area[i] for i in range(0,4)}
```

```
>>>scientist_2
```

输出 {'钱学森': '上海', '邓稼先': '安徽', '华罗庚': '江苏', '李四光': '湖北'}

结合列表推导式和字典推导式，我们归纳出推导式的一般规律：

- (1) 书写格式上是先写出元素的表达式，然后在后面写循环语句；
- (2) 本质上是一种具有变换和筛选功能的函数。

我们来分析一个使用推导式将一个嵌套列表转换成一个一维列表的例子。

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

```
>>>li_b = [j for i in li_a for j in i]
```

```
>>>li_b
```

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

li_a 是一个嵌套列表，for i in li_a for j in i 等价于双重循环：

```
for i in li_a:
```

```
    for j in i:
```

每一次内循环产生一个 j 的值作为列表的元素。

5.4.3 字典的访问

1. 访问单个元素

字典是无序的，因此不支持索引访问，可以使用“键”来访问。如：上面输出的字典。

```
scientist = {'钱学森': '上海', '邓稼先': '安徽', '华罗庚': '江苏', '李四光': '湖北'}
```

要想获取科学家钱学森的籍贯，可以使用 `scientist['钱学森']` 访问，而不能使用 `scientist[0]` 访问。

如果指定的键不存在就会抛出 `KeyError` 异常，在实际开发中，我们不能保证键一定存在，为了保证程序的健壮性，就需要避免该异常产生。下面介绍几种方法：

(1) 使用 `if` 语句对不存在的情况进行处理，设置一个默认值或提示信息，把访问 `scientist` 的代码修改为：
`native = scientist['陈景润'] if '陈景润' in scientist else '抱歉，信息不存在'`，就不会抛出异常。
Python 解释器是怎么“解释”这句话的呢？——先判断键“陈景润”是否在字典 (`scientist`) 中，如果在，就执行 `scientist['陈景润']`，不存在，就执行 `else` 后面的“抱歉，信息不存在”。实际上，这是 Python 的类似于其它编程语言中的三目运算符的写法，在第三章中已经阐述。

(2) 使用 `dict` 对象的 `get(key[, default])` 方法，把访问 `scientist` 的代码修改为：`scientist.get('陈景润', '抱歉，信息不存在')`，结果与 (1) 相同。

2. 遍历字典

(1) 使用 `for` 循环及字典的 `items()` 方法实现

字典对象的 `items()` 方法返回由字典的键值对组成的元组为元素的列表。形式如：

`[(key1, value1), (key2, value2), (key3, value3), ...]`，`key1`, `key2`, `key3` 表示字典的键，`value1`, `value2`, `value3` 表示字典的值。例子：

```
mail_list = {'lihua' : 'lihua@126.com', 'wangqiang' : 'wq@163.com', 'liuhao' : 'lh@qq.com'}
for item in mail_list.items():
    print(item)
```

输出：

```
('lihua', 'lihua@126.com')
('wangqiang', 'wq@163.com')
('liuhao', 'lh@qq.com')
```

如果需要，也可以使用下面的代码分别获得字典的键和值。

```
mail_list = {'lihua' : 'lihua@126.com', 'wangqiang' : 'wq@163.com', 'liuhao' : 'lh@qq.com'}
for key, value in mail_list.items():
    print(key, " :", value)
```

输出：

```
lihua: lihua@126.com
wangqiang:wq@163.com
liuhao:lh@qq.com
```

(2) 使用 `for` 循环及字典的 `values()` 方法、`keys()` 方法实现

字典对象的 `values()` 方法返回由字典的值组成的列表，`keys()` 方法返回由字典的键组成的列表。使用方法类似于 `items()` 方法。仍以上面的字典为例：

```
mail_list = {'lihua' : 'lihua@126.com', 'wangqiang' : 'wq@163.com', 'liuhao' : 'lh@qq.com'}
for key in mail_list.keys():
    print(key)
```

输出：

```
lihua
wangqiang
```

liuhao

```
mail_list = {'lihua' : 'lihua@126.com', 'wangqiang' : 'wq@163.com', 'liuhao' : 'lh@qq.com' }
for value in mail_list.values():
    print(value)
```

输出:

```
lihua@126.com
wq@163.com
lh@qq.com
```

5.4.4 对字典的操作

1. 增加元素

(1) 直接使用语句: `dictname[key] = value` 添加, 如:

```
>>>dic = {'name': '小牧', 'age': 12}
>>>dic['e_mail'] = '111@126.com'
```

将增加一个键值对: `emial:111@126.com`.

说明: 如果新添加的键已经存在, 则将使用新值替换原来该键的值。

(2) 使用 `dict` 对象的 `setdefault(key[, default])` 方法, 如果键已经存在, 则只返回原来的值, 并不替换原来的值。

(3) 使用 `dict` 对象的 `update()` 方法, 如:

```
>>>dic = {'name': '小牧', 'age': 12}
>>>other = {'qq' : 1111111, 'age' : 15}
>>>dic.update(other)
>>>print(dic)
{'name': '小牧', 'age': 15, 'qq': 1111111}
```

可看出, `dict` 对象的 `update()` 方法是把另外一个字典更新进来, 如果键不存在则增加键值对, 若存在则用新值替换原来该键的值。

2. 修改元素

参考增加元素的方法。

3. 删除元素

(1) 使用 `del` 语句

语法格式: `del dictname[key]`。当键不存在时会抛出 `KeyError` 异常。可以使用如下代码先判断键是否存在, 然后再删除。

```
if key in dict:
    del dict[key]
```

(2) 使用字典对象的 `pop()` 方法

语法格式: `dictname.pop(key[, default])`。当键存在时先删除元素, 然后返回该键的值。当键不存在且没有指定 `default` 参数时会抛出 `KeyError` 异常。

(3) 使用字典对象的 `popitem()` 方法

语法格式: `dictname.popitem()`。随机删除字典中的一个元素 (一般删除末尾对), 并以元组的形式返回删除的键值对。如果字典为空, 就会抛出 `KeyError` 异常。

(4) 删除所有元素使用字典对象的 `clear()` 方法

语法格式: dictname.clear()

5.4.5 字典对象的常用方法

方法名称	语法	描述
clear	dictname.clear()	删除字典的所有元素
copy	dictname.copy()	返回一个字典的浅复制
fromkeys	dictname.fromkeys (seq[, value])	创建一个新字典
get	dictname.get(key[, default])	返回指定键的值, 如果键不存在, 返回 default
items	dictname.items()	返回由字典的键值对组成的元组为元素的列表
keys	dictname.keys()	返回由字典的键组成的列表
values	dictname.values()	返回由字典的值组成的列表
update	dictname.update(dict)	把字典 dict 的键值对更新到当前字典
pop	dictname.pop(key[, default])	删除并返回元素
popitem()	dictname.popitem()	随机删除元素

5.4.6 动手实践

1. 字符计数: 输入 1 串英文字母(只包含字母), 输出每种字母出现的次数。

如输入: TomBillyFloy, 输出: T:2 o:2 m:1 B:1 i:1 l:3 y:2

算法分析: 把整个过程划分为三个步骤: 从键盘读入字符串、统计字母出现的次数、输出结果。

统计过程的具体实现: 显然, 用字典保存统计结果是最好的选择, 首先创建一个空字典, 然后遍历字符串得到每个字符, 根据该字符是否已出现过进行增加和修改。

代码如下:

```
01 key_sum = {}
02 word = input("请输入字符串: ")
03 for c in word:
04     if c not in key_sum.keys():
05         key_sum[c] = 1
06     else:
07         key_sum[c] += 1
08 for key,value in key_sum.items():
09     print(key,":",value,end = " ")
```

也可以用简写形式

```
01 key_sum = {}
02 word = input("请输入字符串: ")
03 for c in word:
04     key_sum[c] = 1 if c not in key_sum.keys() else key_sum[c]+1
05 for key,value in key_sum.items():
06     print(key,":",value,end = " ")
```

如果你希望按照字母出现次数从小到大或大到小输出, 可以使用内建函数 sorted()实现, 请自行研究。

2. 矩阵交换行: 给定一个 5×5 的矩阵(数学上, 一个 r×c 的矩阵是一个由 r 行 c 列元素排列成的矩形阵列), 将第 n 行和第 m 行交换, 输出交换后的结果。输入共 6 行, 前 5 行为矩阵的每一行元素, 元素与元素之间以一个空格分开。第 6 行包含两个整数 m、n, 以一个空格分开 (1<= m, n<= 5)。输出交换之后的矩阵, 矩阵的每一行元素

占一行，元素之间以一个空格分开。

输入样例：

```
1 2 2 1 2
5 6 7 8 3
9 3 0 5 3
7 2 1 4 6
3 0 8 2 4
1 5
```

输出样例：

```
3 0 8 2 4
5 6 7 8 3
9 3 0 5 3
7 2 1 4 6
1 2 2 1 2
```

算法分析：把整个过程划分为三个步骤，从键盘读入矩阵数据、交换数据、输出数据。

实现代码如下：

```
01 matrix = {}
02 i =1
03 while i<=5:
04     temp = [int(j) for j in input().split()]
05     matrix[str(i)]=temp
06     i+=1
07 m,n = input().split()
08 matrix[m], matrix[n] = matrix[n], matrix[m]
09 for i in range(1,6):
10     for num in matrix[str(i)]:
11         print(num,end= ' ')
12     print()
```

这个问题思路简单，主要涉及到字典、列表的基本操作，读入矩阵数据时，input()函数读入一行，使用字符串对象的split()方法将其拆分为以字符串为元素的列表，矩阵的用途主要是计算，为此先把元素转换为数值，再保存到列表。输出矩阵数据时，由于字典的无序性，采用遍历键的方式输出，确保顺序正确。采用字典存储仅仅是为了练习字典的操作。如果采用列表存储，代码如下：

```
01 matrix = []
02 i =1
03 while i<=5:
04     temp = [int(j) for j in input().split()]
05     matrix.append(temp)
06     i+=1
07 m,n = input().split()
08 matrix[int(m)-1], matrix[int(n)-1] = matrix[int(n)-1], matrix[int(m)-1]
09 for value in matrix:
10     for num in value:
11         print(num,end= ' ')
12     print()
```

3. 爬楼梯：楼梯有 n 阶台阶，上楼时可以一步上 1 阶，也可以一步上 2 阶，编程计算共有多少种不同的走法？请编写一个程序，输入台阶数，输出走法及走法的数目。设台阶编号依次为 1, 2, 3, ……，从台阶 1 到台阶 2 表示为 1→2，从台阶 2 到台阶 4 表示为 2→4。

分析：考虑特殊情况：

台阶数	走法数目	走法
1	1	0→1 = 1
2	2	(0→1→2)+(0→2) = 2
3	3	(0→1→2→3)+(0→1→3)+(0→2→3) = 3
4	5	(0→1→2→3→4)+(0→2→3→4)+(0→2→4)+(0→1→3→4)+(0→1→2→4) = 5

似乎很难找到规律，我们换个角度思考，第一步可以选择上 1 阶或选择上 2 阶，当走完第一步以后，面临的问题又是选择上 1 阶或选择上 2 阶，如此反复。于是可以分为两类走法：

第一类：上 1 阶，剩余 $n-1$ 阶，共 $n-1$ 种走法。第二类：上 2 阶，剩余 $n-2$ 阶，共 $n-2$ 种走法。设 $F(n)$ 表示走完 n 阶台阶的走法的数目，可得规律 $F(n) = F(n-1) + F(n-2)$ 。

单纯求走法数目的代码如下：

```
01 def steps(n):
02     if n == 1:
03         return 1
04     elif n == 2:
05         return 2
06     else:
07         return steps(n-1)+steps(n-2)
08 num = input("请输入台阶数： ")
09 print(steps(int(num)))
```

★以下是具体走法的代码（需要用到二叉树的知识，已经超出本章范围，你可能会感到有些困难）：

```
01 class Node:
02     def __init__(self, item):
03         self.item = item
04         self.lchild = None
05         self.rchild = None
06 def creattree(root=None, i=0):
07     if i > N:
08         return None
09     else:
10         root = Node(i)
11         root.lchild = creattree(root, i+1)
12         root.rchild = creattree(root, i+2)
13     return root
14 return root
15 def disptree(root, path=''):
16     if root is None:
17         return ' '
18     if root.lchild != None or root.rchild != None:
19         path += str(root.item)+'->'
20     if root.lchild == None and root.rchild == None:
21         path += str(root.item)
22         print(path)
23     else:
24         disptree(root.lchild, path)
25         disptree(root.rchild, path)
26 N =int(input("请输入台阶数： "))
27 root =creattree()
28 disptree(root)
```

如果输入 5：

0->1->2->3->4->5

0->1->2->3->5

```
0->1->2->4->5
0->1->3->4->5
0->1->3->5
0->2->3->4->5
0->2->3->5
0->2->4->5
```

这个问题的思路是，将待求解的问题分解成若干个相互联系的子问题，先求解子问题，然后从这些子问题的解得到原问题的解。这就是动态规划的思想。动态规划的思想运用在实际生活十分广泛，比如“汉诺塔问题”就是动态规划思想。也许你已发现，在数学中重点是根据递推式求通项公式，在计算机中重点是怎样求出递推式（构建递推式难度大、技巧性强）。

4.再谈 *args, **kwargs 参数：请设计一个程序，从标准输入设备读入个数不确定的整数和算术运算（+，-，*，/）进行运算。第一行输入参与运算的整数，用空格隔开，第二行输入运算符号。要求：完成运算的过程用函数实现。

输入样例：

```
1 4 6 7 8 9 3
+
```

输出样例：

```
38
```

算法分析：该问题可分为三个步骤：接收数据、运算、输出。主要步骤是运算，由于参与运算的数的个数不确定，所以可以将接收到的数据存储到列表中，然后遍历列表便可完成计算。但是运算过程要求用函数完成，怎样向函数传递个数不确定的参数呢？先运行如下代码：

```
01 def fun_calc(*args,**kwargs):
02     result = args[0]
03     length = len(args)
04     for i in range(1,length):
05         result =eval(str(result)+kwargs[' symbol' ]+args[i])
06     return result
07 nums = input("请输入参与计算的数：").split()
08 flag =input("请输入运算符：")
09 result = fun_calc(*nums,symbol=flag)
10 print(result)
```

这个问题的关键使用了*args, **kwargs 作为形式参数（当*args, **kwargs 同时出现时，*args 必须在前面）。

首先解释*args, **kwargs 作为形式参数的意义，*args 的意义是将实参中按照非关键字形式传值的“多余”的值以元组方式传递给 args，**kwargs 的意义是将实参中按照关键字形式传值的“多余”的值以字典的方式传递给 kwargs，“多余”是指把实参的值优先匹配位置参数和默认参数后剩下的参数。

例子：

```
01 def fun_test(m, *k, n=5, **p):
02     for item in k:
03         print(item,end=',')
```



```

04     print("\n"+"="*20)
05     for key,value in p.items():
06         print("{}:{}".format(key,value))
07 fun_test(3,4,5,6,n=7,d=8)

```

输出:

4,5,6,

=====

d:8

最后强调两点: (1) 为形式参数指定默认值时, 建议默认值为不可变类型, 否则会出现不可预料的后果。(2) 在运算过程中我们使用了 `eval()` 函数, `eval()` 函数的功能是将字符串当成有效的表达式来求值并返回计算结果。你能联想到什么吗? 这就是 Python 炸弹的根。

5.5 集合 (set)

集合是数学中的一个基本概念, 是由一个或多个确定的元素所构成的整体, 现在几乎渗透到了数学的各个领域。集合的元素具有确定性、互异性、无序性的特点, 元素互异性是集合最重要的特征之一。在计算机科学中同样需要具有元素互异的数据结构, Python 为此设计了两种称为集合的数据结构: 可变集合、不可变集合。可变集合是指元素可以被动态增加、修改和删除的集合, 不可变集合是指集合一旦被创建, 其元素就不能被改变的集合。本书只介绍可变集合。

形式上, 集合是用 “{ }” 包裹的一系列用 “,” 分隔的元素序列。如:

```
animal = { '老虎', '狮子', '斑马', '穿山甲' }
```

5.5.1 集合的创建

1. 使用赋值语句直接创建

语法格式如下:

```
setname = {element1, element2, .....}
```

其中, `setname` 表示集合名, `element1`, `element2` 表示集合的元素。`setname` 可以是任何符合 Python 命名规则的变量名。元素的个数没有限制, 集合中的元素不能是可变数据类型。

例如:

```
ageset = {10, 15, 20, 22, 80, 25, 22}
```

```
bookset = { '程序设计', ('c', 'c++', 'Python'), '系统理论', ('数据结构', '算法', '计算机原理') }
```

说明: 在创建集合时如果有重复的元素, 会自动去掉重复的只保留一个。

2. 创建空集合

创建空集合, 直接使用代码: `setname = set()`。注意不能使用 `setname = { }`。

3. 使用 `set()` 函数创建

语法格式如下:

```
setname = set(data)
```

`data` 表示可以转换为集合的对象, 如 `range` 对象、字符串等其它任何可迭代对象。

例子:

```
age = set(range(1, 10))
```

将创建 (1, 2, 3, 4, 4, 5, 6, 7, 8, 9) 集合。

```
name = "xiaofang"
```

```
names = set(name)
```

将创建{ 'o' , ' f' , ' x' , ' n' , ' i' , ' g' , ' a' }集合。

```
li = [6,7,8,9,8]
num = set(li)
```

将创建{8,9,6,7}集合。

4. 使用集合推导式创建

集合推导式与列表推导式类似，基本语法格式如下：

{表达式 for 变量 in 列表}

或者 {表达式 for 变量 in 列表 if 条件}

例子：

```
age = { i for i in range(1,120)}
```

将创建一个集合，里面的元素分别为1, 2, 3, ...119，注意不包括120。

```
age = { i*i for i in range(1,120) if i%2==0 }
```

将创建一个由1到120之间的偶数的平方作为元素的集合，注意不包括120的平方。

我们发现，创建列表、元组、字典、集合分别可以使用内建函数list(data)、tuple(data)、dict(data)、set(data)，参数都是可迭代对象，只有字典稍有特别。在实际开发中，数据量都比较大，很少使用赋值语句创建，通过内建函数是使用最多的创建方式之一。我们只需记住这几个函数名就可以了。除元组外，列表、字典、集合也可以使用推导式创建。推导式是Python的一种独有特性，是根据某种规律生成元素，一次性生成所有元素并加载到内存中，具有语言简洁，速度快等优点。但是我们也要知道，如果数据量很大，比如几个G的数据，这将会占用大量内存降低效率，这是推导式的缺点。随着今后的学习你会接触到生成器，生成器很好地解决了这个问题。

5.5.2 集合的访问

由于集合是无序的，所以不支持使用索引和切片访问，不能访问集合中的单个元素，也不能对集合中单个元素作出修改。

使用 for 循环遍历集合

语法格式：

```
for item in setname:
    #输出 item
```

其中，item用于保存依次从集合中获取的元素的值，setname 集合名。

例子：

```
trans_tool = { '百度翻译' , ' 金山词霸' , ' Google 翻译' , ' 有道词霸' }
for item in trans_tool:
    print(item,end=" ")
```

输出：金山词霸 Google 翻译 百度翻译 有道词霸

说明：由于集合、字典元素的无序性，每次遍历结果的顺序可能会不一样。

5.5.3 对集合的操作

1. 增加元素

(1) 使用 set 对象的 add()方法，如：

```
>>>book_set = { '红楼梦' , ' 西游记' , ' 水浒传' }
>>>book_set.add(' 三国演义' )
>>>print(book_set)
```

输出：{ '红楼梦' , ' 西游记' , ' 水浒传' , ' 三国演义' }

(2) 使用 set 对象的 update() 方法, 如:

```
>>>book_set = {'红楼梦', '西游记', '水浒传'}
>>>book_set.update('三国演义')
>>>print(book_set)
```

输出: {'红楼梦', '西游记', '水浒传', '三', '国', '演', '义'}

对比这两个方法可以看出, add() 方法是将参数作为 1 个元素增加, update() 方法是将参数解包后增加。

2. 删除元素

删除集合的元素可以使用集合对象的 pop() 方法、remove() 方法、clear() 方法。

实践: 请在 IDLE 的交互模式下运行下面代码。

```
>>>car = {'宝马', '奔驰', '大众', '奥迪', '起亚'}
>>>car.remove('起亚')
>>>print(car)
输出: {'宝马', '奔驰', '大众', '奥迪'}
>>>car.pop()
>>>print(car)
输出: {'宝马', '奔驰', '奥迪'}
>>>car.clear()
>>>print(car)
```

输出: set()

说明: (1) 使用 remove() 方法删除元素时, 如果元素不存在, 则会抛出 KeyError 异常, 可以先使用 in 关键字判断元素是否存在, 然后再进行删除;

(2) pop() 方法随机删除一个元素;

(3) clear() 方法删除所有元素。

5.5.4 集合的运算

与数学中集合的运算类似, Python 为集合提供了交、并、差、对称差等运算。

假设有集合 A、B。

交集: 是指由既在集合 A 又在集合 B 中的元素组成的集合。运算符号: “&”。

并集: 是指由在集合 A 或在集合 B 中的元素组成的集合。运算符号: “|”。

差集: 是指由在集合 A 但不在集合 B 中的元素组成的集合。运算符号: “-”。

对称差: 是指不同时在集合 A 和集合 B 中的元素组成的集合。运算符号: “^”。

例子: 某校学业水平考试等级表 (部分)

姓名	语文	数学	英语	物理
王巧巧	A	A	B	C
隆继宗	A	B	C	A
河姑	B	C	A	B
魏文位	A	A	C	A
金华点	A	B	A	B

求: (1) 语文、数学都为 A 的同学 (2) 语文为 A 或英语为 A 的同学 (3) 语文为 A 但物理不为 A 的同学 (4)

语文、英语不同为 A 的同学。

将各科等级为 A 的构建为集合，再根据集合进行运算，以下是实现代码：

```
01 chinese = {'王巧巧', '隆继宗', '魏文位', '金华点'}
02 math = {'王巧巧', '魏文位'}
03 english = {'河姑', '金华点'}
04 physics = {'隆继宗', '魏文位'}
05 print('语文、数学都为 A 的有:', chinese & math)
06 print('语文为 A 或英语为 A 的有:', chinese | english)
07 print('语文为 A 但物理不为 A 的有:', chinese - physics)
08 print('语文、英语不同为 A 的有:', chinese ^ english)
```

输出：

语文、数学都为 A 的有： {'王巧巧', '魏文位'}

语文为 A 或英语为 A 的有： {'河姑', '王巧巧', '魏文位', '金华点', '隆继宗'}

语文为 A 但物理不为 A 的有： {'金华点', '王巧巧'}

语文、英语不同为 A 的有： {'王巧巧', '隆继宗', '河姑', '魏文位'}

5.5.5 列表、元组、字典、集合的比较

数据结构	是否可变	是否可重复	是否有序	定义符号	是否有推导式
列表(list)	是	可以	有	[]	有
元组(tuple)	否	可以	有	()	无
字典(dict)	是	可以	无	{key:value}	有
集合(set)	是	不可以	无	{}	有

在实际应用中，需要我们对这些数据结构的特点有清晰的认识，通过练习这些片段代码逐渐掌握它们的主要适用场合，我们开发的应用才能更高效地工作。如不希望对列表的元素进行修改时用元组，字典相对于列表具有更快的访问速度，在设计高并发应用时选择字典可能会优越于列表。

5.5.6 动手实践

1. 单词重复计数：输入一段英语句子，编写一个程序，统计所有单词重复的次数。

输入样例：I am a student He is also a student

输出样例：2

算法分析：为了统计单词重复的次数，我们可以把句子转换为列表，然后使用列表对象的 count() 方法依次统计各个单词出现的次数，减 1 后用字典存储，最后累加即可。也可以使用集合元素互异性的特点，先求出原列表单词的个数 m，然后将列表转换为集合，再求出集合元素的个数 n，m-n 即为所求。实现代码如下：

```
01 words = input().split()
02 se = set(words)
03 m = len(words)
04 n = len(se)
05 print(m-n)
```

2. 学生编班：新课改规定学生可以从物理、化学、生物、政治、历史、地理六门课程中任意选择三门参加

高考。学校经过志愿征集后得到如下表格：1 表示选择，0 表示不选择。

姓名	物理	化学	生物	政治	历史	地理
潘虹	1	1	0	1	0	0
覃姗	0	0	1	1	1	
王丽	1	0	1	0	0	1
.....						

如果存在同时选择某三门课的学生数大于或等于 56，就将这些学生安排为“行政班”。请编写一个程序判断是否可以开设“行政班”，并输出组合科目。

假设数据已按如下格式保存：

```
select_info = {"潘虹": [1, 1, 0, 1, 0, 0], "覃姗": [0, 0, 1, 1, 1, 0], "王丽": [1, 0, 1, 0, 0, 1]}.
```

输出样例：可以 物理 生物 历史

输出样例：不可以

算法分析：从 6 门课程中任意选择 3 门课程共有 20 种组合，需要对每一种组合进行判断。为此，首先从列表中将选择每门课程的学生分离出来，用集合保存，再通过集合的交集运算求解。实现代码如下：

```
01 # *_ coding:utf-8 *_
02 from itertools import combinations #导入求组合数模块
03 """
04 student[0]:选择物理科目的学生
05 student[1]:选择化学科目的学生
06 student[2]:选择生物科目的学生
07 student[3]:选择政治科目的学生
08 student[4]:选择历史科目的学生
09 student[5]:选择地理科目的学生
10 """
11 N =1 #同时选择课程学生数
12 is_can = False
13 select_info = {"潘虹": [1, 1, 0, 1, 0, 0], "覃姗": [0, 0, 1, 1, 1, 0], "王丽": [1, 0, 1, 0, 0, 1]}
14 object = ['物理', '化学', '生物', '政治', '历史', '地理']
15 student = []
16 #初始化列表
17 for i in range(0, 6):
18     student.append(set())
19 #分离选课学生数据，保存到集合中
20 for key, value in select_info.items():
21     for i in range(0, 6):
22         if value[i]==1:
```

```

23         student[i].add(key)
24 combin = list(combinations(range(0, 6), 3))
25 for i, j, k in combin:
26     if len(student[i] & student[j] & student[k]) >= N:
27         print('可以', object[i], object[j], object[k])
28         is_can = True
29 if not is_can:
30     print("不可以")

```

本章小结

本章按照创建、增加、查找、修改、删除的顺序介绍了列表、元组、字典、集合这几种数据结构。这些操作是对数据结构的基本操作，同时也是主要操作。我们已经知道大多数的操作都是使用对象本身的方法进行的，Python中的一切皆为对象，在后面的章节中会学习类和模块，你会发现Python有很多模块，能否掌握模块提供的方法是成功的关键。除此之外，还介绍了关于排序的几种算法、二分查找算法，引入了枚举思想、分治策略、动态规划基本思想，希望你能仔细体会这些思想。

练习题

1. **位置前移**：输入用空格隔开的 n 个不同的整数，第 1 个数移到末尾，其余各数依次往前移 1 个位置。

输入样例：5 6 7 8

输出样例：6 7 8 5

2. **最长单词**：输入一段简单英文句子（长度不超过 500），单词之间用空格分隔，没有缩写形式和其它特殊形式。找出该句子中最长的单词。如果多于一个，则输出第一个。

输入样例：I am a student of Peking University

输出样例：University

3. **约瑟夫问题**： n 个人（以编号 1, 2, 3, ..., n 分别表示）围成一圈。从第一个人开始报数，数到 m 的那个人出列；他的下一个人又从 1 开始报数，数到 m 的那个人又出列；依此规律重复下去，直到所有人出圈。依次输出出圈人的编号。 n, m 由键盘输入。

输入样例：

10

6

输出样例：6 2 9 7 5 8 1 10 4 3

4. **石头剪子布**：石头剪子布是一种猜拳游戏，深受世界人民喜爱。现用字典：

`game = {'computer' :'', 'person' :''}` 表示电脑和人的出拳情况，用 R、S、P 分别表示石头、剪子、布。

游戏规则：石头打剪刀，布包石头，剪刀剪布。现在，需要你写一个程序来判断石头剪子布游戏的结果。程序运行后，输入 10 个由 R、S、P 组成的字符串，随即电脑随机产生 10 个由 R、S、P 组成的字符串。输出电脑产生的字符串和游戏结果

输出样例：电脑出拳：RSSRPPSSR

电脑胜：6

5. **控制线划定**：高考的录取批次划分为第一批和第二批，招生录取计划数提前制定，考生参加高考后成绩按从高到低排序。假设第一批控制线是根据计划录取人数的 140% 划定，考生人数为 n ($n \leq 1000000$)，成绩各不相同，第一批计划数为 m ($1 \leq m \leq n$)，请编写一个程序划定第一批控制线。

输入样例：

699.32 650.32 660.35 580.6 603.7 530.6 701.6 480.6 536.5

3

输出样例：603.7

6. **数值求和**：现有一个元素很多的列表（如 10000000000 个），元素值都在 $1 \sim 100$ 之间，元素已按从小到大排序，请编写一个程序计算所有元素的和。提示：侧重考虑效率。

输入样例：1 1 1 2 3 3 3 3 4 5 6 6

输出样例：38

7. **非空子集**：输入用空格隔开的 n 个不同的整数，输出由这些整数作为元素的集合的所有非空子集。

输入样例：3 4 6

输出样例： [{3}, {4}, {6}, {3, 4}, {3, 6}, {4, 6}, {3, 4, 6}]