第5章 字符串、列表和文件

5.1 字符串数据类型

到目前为止，本书一直在讨论操作数字和图形的程序。正如你所知，计算机对于存储和操作文本信息也很重要。事实上，个人计算机最常见的用途之一就是文字处理。本章将主要介绍文本应用程序，讲解一些关于文本如何存储在计算机上的重要设计思路。

文本在程序中由字符串数据类型表示。你可以将字符串看成一个字符序列。在第2章中已经介绍过，通过用双引号将一些字符括起来形成字符串字面量。Python还允许字符串由单引号分隔。它们没有区别，但使用时一定要配对。字符串也可以保存在变量中，像其他数据一样。下面举一个示例，来说明两种形式的字符串字面量。

```
>>> str1 = "hello"
>>> str2 = 'world'
>>> print(str1,str2)
hello world
>>> type(str1)
<class 'str'>
>>> type(str2)
<class 'str'>
```

通过上面的代码演示，你已经知道了如何打印字符串，你也了解了如何获取用户输入的字符串。回想一下，input()函数可以返回用户输入的任何字符串对象。这意味着如果你希望得到一个字符串，可以使用其“原始”形式的输入。来看一下这种简单交互：

```
>>> str = input("请输入你的名字:")
请输入你的名字:张三
>>> print("你好"+str)
你好张三
```

请注意上面的示例，是如何用变量来保存用户名称，然后又如何用该变量将名称打印出来的？

对于复杂的问题处理，总是会涉及很多关于字符串的相关操作。首先来了解一下如何访问字符串的每个元素。在Python中，可以利用索引来实现这个操作。为了方便理

解，可以利用表格将索引表示出来。以“hello world!”为例，字符串的索引如图 5.1 所示。字符串索引以 0 开始，从左到右。索引的一般形式是＜string＞［＜expr＞］，其中 expr 的值确定从字符串中选择哪个字符。

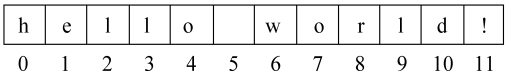

图 5.1　字符串的索引

以下是一些交互式的索引示例：

```
>>> str = "hello world!"
>>> str[0]
'h'
>>> str[5]
' '
>>> str[10]
'd'
>>> str[11]
'!'
```

请注意，在有 n 个字符的字符串中，因为索引从 0 开始，所以最后一个字符的索引为 n－1。中文字符串与英文字符串的访问方式一致。顺便说一下，Python 还允许使用负索引，从字符串的右端索引，即－1 为最右侧的一个字符，然后－2、－3 依次向左递减。

```
>>> str[-1]
'!'
>>> str[-2]
'd'
>>> str[-12]
'h'
```

有时候，可能需要选取字符串中的某个字符或某段“子字符串”，在 Python 中，可以使用“切片”的操作来实现。你可以把切片想象成在字符串中索引一系列位置的方法。切片的形式是＜string＞［＜start＞:＜end＞］。start 和 end 都应该是整型表达式。切片产生从 start 直到 end 位置(不包括 end)的子串，即数学中的一个“左闭右开”的取值范围。

还以上面 str 的示例来展示切片操作：

```
>>> str[2:4]
'll'
>>> str[2:7]
'llo w'
>>> str[:7]
'hello w'
```

```
>>> str[2:]
'llo world!'
>>> str[:]
'hello world!'
```

从上面的示例可以看出,切片操作为“左闭右开”的形式。当切片从 2 到 4 时,只选择索引为 2 和 3 的字符。最后 3 个示例表示,如果任何一个表达式缺失,字符串的开始和结束都是假定的默认值。最后的表达式实际上给出了整个字符串。

索引和切片是将字符串切成更小片段的有用操作。字符串数据类型还支持将字符串放在一起的操作。其中连接(+)和重复(*)是很常见的操作。连接(+)是通过将两个字符串“合并”在一起来构建新的字符串;重复(*)是通过字符串与多个自身连接,来构建新的字符串。另一个很有用的是 len()函数,它的作用是返回字符串中有多少个字符。由于字符串是字符序列,因此可以使用 Python 的 for 循环来遍历这些字符。

```
>>> s1 = "hello"
>>> s2 = " world"
>>> s3 = "!"
>>> print(s1+s2+s3)
hello world!
>>> print(3 * s2)
world world world
>>> print(2 * s1+3 * s2+3 * s3)
hellohello world world world!!!
>>> m = len(s1)
>>> print(m)
5
>>> for ch in range(m):
        print(s1[ch])

h
e
l
l
o
```

基本的字符串操作总结如表 5-1 所示。

表 5-1　Python 的字符串操作

操　作　符	含　　义
+	连接
*	重复

续表

操 作 符	含 义
<string>[ ]	索引
<string>[:]	切片
len(<string>)	长度
for <var> in <string>	迭代遍历字符串

5.2 简单字符串处理

许多计算机系统会使用用户名和密码组合来认证系统用户。系统管理员必须为每个用户分配唯一的用户名。通常，用户名来自用户的实际姓名。在国外，有一种用于生成用户名的方案：使用用户名的第一个首字母，然后是用户姓氏的最多前七个字母。利用这种方法，Zaphod Beeblebrox 的用户名为“zbeebleb”，而 John Smith 的为“jsmith”。

可以尝试编写一个程序，读取一个人的名字并计算相应的用户名。程序将遵循基本的输入、处理、输出模式。

```
#5_1 username.py
def main():
    print("这个程序生成用户名.\n")
    #获取用户姓和名
    first=input("请输入你的姓(请以字母表示):")
    last=input("请输入你的名(请以字母表示):")
    #选择名的第一个字母和姓的前七位字母
    uname=last[0]+first[:7]
    #输出用户名
    print("你的用户名为:",uname)
main()
```

这个程序首先利用 input()函数从用户获取字符串，然后组合使用索引、切片和连接来生成用户名。下面是运行结果：

```
这个程序生成用户名

请输入你的姓(请以字母表示): zhang
请输入你的名(请以字母表示): san
你的用户名为: szhang
```

其中第一行 print 语句使用了换行符(\n)，实现输出一个空白行。这是一个简单的技巧，输出一些额外的空白行，使得结果更加清楚明了。

下面是另一个示例，打印给定月份数对应的月份缩写。程序的输入是一个整型值，代表一个月份(1～12)，输出是相应月份的缩写。例如，如果输入为 3，则输出应为 Mar，即 3 月。

根据字符串的思想，将所有月份名存储在一个字符串变量中：

```
months = "JanFebMarAprMayJunJulAugSepOctNovDec"
```

由于每个月份都是 3 个字符，如果知道一个给定的月份在字符串中开始的位置，就可以很容易提取出缩写。

```
monthAbbrev = months[pos:pos+3]
```

上面代码将获得从 pos 指示位置开始的长度为 3 的子字符串。

如何计算这个位置？先分析一下几个示例(如表 5-2 所示)，看看有什么发现？记住，字符串索引从 0 开始。

表 5-2　月份缩写字符串中的位置

月　　份	数　　字	位　　置
Jan	1	0
Feb	2	3
Mar	3	6

显然，这些位置都是 3 的倍数。为了得到正确的倍数，从月份数中减去 1，然后乘以 3。所以对于 1，可以得到(1－1)×3＝0×3＝0，对于 12，有(12－1)×3＝11×3＝33。

下面为实现的代码：

```
#5_2 months.py
def main():
    #将月份组成一个大字符串
    months = "JanFebMarAprMayJunJulAugSepOctNovDec"
    n = int(input("输入一个月份 (1-12): "))
    #计算第 n 个月的起始位置
    pos = (n-1) * 3
    #提取第 n 个月的名称
    monthAbbrev = months[pos:pos+3]
    #输出结果
    print("月份的英文缩写为", monthAbbrev + ".")
main()
```

这个示例使用“字符串作为查找表”的方法，但它有一个缺点，即仅当子串都有相同的长度时程序才是有效的。

5.3 列表作为序列

Python 列表也是一种序列，这就说明列表同样可以切片、连接和索引列表。列表的一个好处是它比字符串更通用。字符串总是字符序列，而列表可以是任意对象的序列。你可以创建数字列表或字符串列表。事实上，你甚至可以混合它们，来创建一个既包含数字又包含字符串的列表，示例如下：

```
>>>list= [1,"hello",2,"world"]
>>>list
[1, 'hello', 2, 'world']
```

在以后的章节，将经常使用列表，即将所有的东西都放在列表中，各种操作都以列表的形式进行。

如 5.2 节读取月份的示例，还记得这个程序的缺点吗？当字符串不相等时，这种方法无法实现。现在利用列表就可以轻松地解决这个问题。

```
#5_3 months_new.py
def main():
    #将所有的月份存储为一个列表
    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun","Jul", "Aug", "Sep",
"Oct", "Nov", "Dec"]
    n = int(input("请输入月份 (1-12): "))
    print("月份的英文缩写是", months[n-1] + ".")

main()
```

列表与字符串一样，从 0 开始索引，因此在此列表中，值[0]是字符串"Jan"。一般来说，第 n 个月在位置 n－1，直接在 print 语句中用表达式 months[n－1]即可。

这个缩写问题的解决方案不仅更简单，而且更灵活。例如，修改前面的示例，打印出整个月份的名称，则只需要重新定义查找列表，就可以很轻松地解决问题了。

虽然字符串和列表都是序列，但两者之间有一个非常重要的区别：列表是可变的，而字符串是不可变的。这意味着列表中项的值可以使用赋值语句来修改。另外，需要注意的是，字符串不能在“适当位置”改变。下面看一个示例交互，说明其中的区别：

```
>>> list = [2,3,"a","b"]
>>> list[2] = 4
>>> list[0] = 5
>>> list
[5, 3, 4, 'b']
>>> str = "hello world"
```

```
>>> str[2] = "r"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

第一行建立了一个数字与字符混合的列表 list，然后将索引位置 0 赋值为 5，索引位置 2 赋值为 4，赋值后，列表被成功替换。而字符串变量 str 则不能这样进行修改。

5.4 字符串表示和消息编码

5.4.1 字符串表示

在计算机中，表示字符串的方式就是将每个字符串翻译为一个二进制的数字，整个字符串作为二进制数字序列存储在计算机存储器中。只要计算机的编码/解码过程一致，用什么数字表示任何给定字符并不重要，最终都可以得到原来的字符串。

现在计算机统一了标准编码。一个重要的标准编码为 ASCII(美国信息交换标准代码)。ASCII 用数字 0～127 来表示美式键盘上所有的字符以及被称为控制代码的某些特殊值，用于协调信息的发送和接收。例如，大写字母 A～Z 的 ASCII 码为 65～90，小写字母的 ASCII 码为 97～122。

ASCII 编码存在一个问题，就是它以美国为中心，没有涉及其他语言和文字的需要。国际标准组织已经开发了一套扩展 ASCII 编码，用来纠正这种情况。大多数现代系统正在向 Unicode 字符集转移，这是一个更大的标准，旨在包括几乎所有国家或地区的书面语言的字符。Python 字符串支持 Unicode 字符集，因此，只要你的操作系统有适当的字体来显示字符，就可以处理来自任何语言的字符。

Python 提供了几个内置函数，允许在字符与表示它们的数值之间来回切换。ord()函数返回单字符串的数字编码，而 chr()函数刚好相反。下面是一些交互的示例：

```
>>> ord("a")
97
>>> ord("A")
65
>>> chr(97)
'a'
>>> chr(90)
'Z'
```

关于计算机存储字符的原理，底层的 CPU 通常处理固定大小的内存数据。最小可寻址段通常为 8 位，称为存储器字节。单字节可以存储 $2^8=256$ 个不同的值。这足以代表每个可能的 ASCII 字符。但是单字节远远不足以存储所有 10 万个可能的 Unicode 字

符。为了解决这个问题，Unicode 字符集定义了将 Unicode 字符打包成字节序列的各种编码方案。最常见的编码称为 UTF-8。UTF-8 是一种可变长度编码方案，用单字节存储 ASCII 的字符，但可能需要最多 4 字节来表示一些更为深奥的字符。这意味着长度为 10 个字符的字符串最终将以 10～40 字节的序列存储在内存中，具体取决于字符串中使用的实际字符。然而，从拉丁字母的使用经验来看，一个字符平均需要大约一字节的存储是相对比较安全的一种方案。

5.4.2 编写编码器

下面利用 Python 的 ord()函数和 chr()函数来编写一个简单的程序，实现将消息转换为数字序列再转换回来的过程。假设要将一段话转换为 Unicode 码表示，编码的算法很简单，设计如下：

```
※ ------伪代码------
※ 对消息进行编码
※ for ch in message:
※ 打印字符的 Unicode 码
```

用户获取消息，只需要一个 input()函数即可。

实现循环首先需要针对消息的每个字符进行操作。回想一下前面学习过的使用 for 循环遍历一系列对象的方法。由于字符串是一种序列，因此可以用 for 循环遍历消息的所有字符：

```
for ch in message:
```

最后将每个字符转换为数字。最简单的方法就是对消息中的每个字符转换为 Unicode 码，直接使用 ord()函数即可实现。

下面是消息编码的最终代码：

```
#5_4 encodes_Unicode.py
def main():
    print("将文本消息转换为序列")
    print("消息的 Unicode 码\n")
    #获取编码消息
    message = input("输入待编码的文本: ")
    print("\nUnicode 编码:")
    #循环消息并打印出 Unicode 码
    for ch in message:
        print(ord(ch), end=" ")
    print()
main()
```

输出结果如下：

```
将文本消息转换为序列
消息的 Unicode 码

输入待编码的文本：轻轻的我走了，正如我轻轻的来

Unicode 编码为：
36731 36731 30340 25105 36208 20102 65292 27491 22914 25105 36731 36731 30340 26469
```

5.5 字符串方法

5.5.1 编写解码器

5.4 节通过编写编码器了解到，要想理解字符的 Unicode 码所表示的内容，就需要对应的解码器。接下来一起看看如何解决这个问题。解码器程序提示用户输入一系列 Unicode 码，然后打印出带有相应字符的文本消息。这个程序具有几个挑战性的难度，下面就通过学习，一起来解决这些问题。

解码器程序的总体轮廓看起来与编码器程序非常类似。解码器会在字符串中收集消息的字符，并在程序结束时打印出整条消息。为此，需要用一个累加器变量。下面是解码器算法：

```
※ ------伪代码------
※ 得到要解码的数字序列
※ message = ""
※ for each number in the input:
※ 将数字转换为相应的 Unicode 码
※ 将字符添加到消息的末尾
※ 打印 message
```

在循环之前，累加器变量 message 被初始化为空字符串，即不包含字符的字符串（""）。每次通过循环，输入的数字被转换为相应的字符，并附加到之前构造的 message 末尾。

为了得到要解码的数字序列，还需要依靠更多的字符串操作，具体步骤如下：首先利用输入将整个数字序列读入为单个字符串；其次，可以将大字符串拆分为一系列较小的字符串，每个字符串代表一个数字；最后，通过遍历更小的字符串列表，将每个字符串转换为一个数字，并使用该数字来产生相应的 Unicode 字符。下面是完整的算法：

```
※ ------伪代码------
※ 以 string,inString 的形式获取数字序列
```

```
※ 将 inString 分解成一系列小字符串
※ message = ""
※ for each number in the input:
※ 将数字转换为相应的 Unicode 码
※ 将字符添加到消息的末尾
※ 打印 message
```

对于解码器，可以使用 split()函数。此方法的作用是将字符串拆分为子串列表。默认情况下，它会在遇到空格时拆分字符串。当然也可以指定其他分隔符。下面是一个示例：

```
>>> myString = "hello world!"
>>> myString.split()
['hello', 'world!']
>>> String = "a,b,c,d"
>>> String.split(",")
['a', 'b', 'c', 'd']
```

也可以利用 split()函数获取多个输入数据。例如，可以获取单个输入字符串中的一个点的 x 值和 y 值，使用 split()函数将其转换为列表，然后索引得到的列表，获取单个字符串部分，具体的示例如下：

```
>>> coords = input("输入点的坐标(x,y): ").split(",")
输入点的坐标(x,y): 3,5
>>> coords
['3', '5']
>>> coords[0]
'3'
```

再回到解码器的问题，可以使用类似的技术。由于程序应该接收编码器程序产生的相同格式，即一系列具有空格的 Unicode 码，使用 split()函数就可以十分轻松地解决问题，提升编程的效率。

同样，如果不是数字列表，而是字符串列表，只不过这些字符串包含数字，那么，又将如何提取这些数字呢？下面通过举一个示例来解释说明一下。在这个示例中的字符串都是整型字面量，因此可以把 int()函数应用于每一个字符串，将其转换为 Unicode 码，然后再进行解码处理。

使用 split()函数和 int()函数，编写解码器如下：

```
#5_5 decodes_Unicode.py
def main():
    print("将 Unicode 码序列转换为文本字符串")
    print("文本字符串\n")
```