# 第5章 数据类型

CHAPTER 5

在声明变量时会用到数据类型，前面章节已经用到过一些数据类型，例如整数和字符串等。在 Python 中所有的数据类型都是类，每一个变量都是类的“实例”。Python 中没有基本数据类型的概念，所以整数、浮点和字符串也都是类。

Python 有 6 种标准数据类型：数字、字符串、列表、元组、集合和字典，其中列表、元组、集合和字典可以保存多项数据。它们每一个都是一种数据结构，本书中把它们统称为“数据结构”类型。

本章介绍数字和字符串，列表、元组、集合和字典这 4 种数据类型在后面章节会详细介绍。

## 5.1 数字类型

Python 数字类型有 4 种：整数类型、浮点类型、复数类型和布尔类型。需要注意的是，布尔类型也是数字类型，事实上它是整数类型的一种。

### 5.1.1 整数类型

Python 整数类型为 int，整数类型的范围很大，可以表示很大的整数，最大范围只受计算机硬件的限制。

**提示** Python 3 不再区分整数和长整数，所有需要的整数都可以是长整数。

默认情况下一个整数值表示十进制，例如 16 表示的是十进制整数。其他进制，如二进制数、八进制数和十六进制整数的表示方式如下：

(1) 二进制数：以 0b 或 0B 为前缀，注意 0 是阿拉伯数字，不要认为是英文字母 o。

(2) 八进制数：以 0o 或 0O 为前缀，第一个字符是阿拉伯数字 0，第二个字符是英文字母 o 或 O。

(3) 十六进制数：以 0x 或 0X 为前缀，注意 0 是阿拉伯数字。

例如整数值 28、0b11100、0B11100、0o34、0O34、0x1C 和 0X1C 都表示同一个数字。在 Python Shell 的输出结果如下：

```
>>> 28
28
>>> 0b11100
28
```

```
>>> 0O34
28
>>> 0o34
28
>>> 0x1C
28
>>> 0X1C
28
```

### 5.1.2　浮点类型

浮点类型主要用来储存小数数值，Python 浮点类型为 float，Python 只支持双精度浮点类型，而且与本机相关。

浮点类型可以用小数表示，也可以用科学计数法表示，科学计数法中使用大写或小写的 e 表示 10 的指数，如 e2 表示 $10^2$。

在 Python Shell 中运行示例如下：

```
>>> 1.0
1.0
>>> 0.0
0.0
>>> 3.36e2
336.0
>>> 1.56e-2
0.0156
```

其中，3.36e2 表示 $3.36\times10^2$，1.56e−2 表示 $1.56\times10^{-2}$。

### 5.1.3　复数类型

复数在数学中是非常重要的概念，无论是理论物理学，还是在电气工程实践中都经常使用。但是很多计算机语言都不支持复数，而 Python 是支持复数的，这使得 Python 能够很好地进行科学计算。

Python 中复数类型为 complex，例如 1＋2j 表示的是实部为 1、虚部为 2 的复数。在 Python Shell 中运行示例如下：

```
>>> 1 + 2j
(1 + 2j)
>>> (1 + 2j) + (1 + 2j)
(2 + 4j)
```

上述代码实现了两个复数(1＋2j)的相加。

### 5.1.4　布尔类型

Python 中布尔类型为 bool，bool 是 int 的子类，它只有 True 和 False 两个值。

**注意**　任何类型数据都可以通过 bool()函数转换为布尔值，那些被认为“没有的”、“空的”值会转换为 False，其余转换为 True。如 None(空对象)、False、0、0.0、0j(复数)、''

（空字符串）、[]（空列表）、()（空元组）和{}（空字典）等都会转换为 False，其余的转换为 True。

示例代码如下：

```
>>> bool(0)
False
>>> bool(2)
True
>>> bool(1)
True
>>> bool('')
False
>>> bool(' ')
True
>>> bool([])
False
>>> bool({})
False
```

上述代码中 bool(2)和 bool(1)表达式输出的都是 True，这说明 2 和 1 都能转换为 True，在整数中只有 0 转换为 False，其他类型亦是如此。

## 5.2 数字类型相互转换

学习了数据类型后，读者可能会思考一个问题，数据类型之间是否可以转换呢？Python 通过一些函数可以实现不同数据类型之间的转换，如数字类型之间互相转换以及整数与字符串之间的转换。本节先讨论数字类型的互相转换。

除复数外，其他三种数字类型（整数、浮点和布尔）都可以互相进行转换，转换分为隐式类型转换和显式类型转换。

### 5.2.1 隐式类型转换

多个数字类型数据之间可以进行数学计算，若参与计算的数字类型不同，则会发生隐式类型转换。隐式类型转换规则如表 5-1 所示。

**表 5-1 隐式类型转换规则**

| 操作数 1 类型 | 操作数 2 类型 | 转换后的类型 |
| --- | --- | --- |
| 布尔 | 整数 | 整数 |
| 布尔、整数 | 浮点 | 浮点 |

布尔数值可以隐式转换为整数类型，布尔值 True 则转换为整数 1，布尔值 False 则转换为整数 0。在 Python Shell 中运行示例如下：

```
>>> a = 1 + True
>>> print(a)
2
```

```
>>> a = 1.0 + 1
>>> type(a)                                                    ①
<class 'float'>
>>> print(a)
2.0
>>> a = 1.0 + True
>>> print(a)
2.0
>>> a = 1.0 + 1 + False
>>> print(a)
2.0
```

从上述代码的运算结果可知表 5-1 所示的类型转换规则，这里不再赘述。另外，上述代码第①行使用了 type()函数，type()函数可以返回传入数据的类型，<class 'float'>说明数据是浮点类型。

### 5.2.2　显式类型转换

在不能进行隐式转换的情况下，就可以使用转换函数进行显式转换了。除复数外，三种数字类型(整数、浮点和布尔)都有自己的转换函数，分别是 int()、float()和 bool()函数。bool()函数在 5.1.4 节已经介绍过了，这里不再赘述。

int()函数可以将布尔、浮点和字符串类型转换为整数类型。布尔数值为 True 时，使用 int()函数则返回 1；布尔数值为 False 时，使用 int()函数则返回 0。浮点数值使用 int()函数则会截掉小数部分。int()函数转换字符串会在 5.3 节再介绍。

float()函数可以将布尔、整数和字符串类型转换为浮点类型。布尔数值为 True 时，使用 float()函数则返回 1.0；布尔数值为 False 时，使用 float()函数则返回 0.0；整数值使用 float()函数则会加上小数部分(.0)。float()函数转换字符串也会在 5.3 节再介绍。

在 Python Shell 中运行示例如下：

```
>>> int(False)
0
>>> int(True)
1
>>> int(19.6)
19
>>> float(5)
5.0
>>> float(False)
0.0
>>> float(True)
1.0
```

## 5.3　字符串类型

由字符组成的一串字符序列称为“字符串”，字符串是有顺序的，从左到右，索引从 0 开始依次递增。Python 中字符串类型是 str。

### 5.3.1 字符串表示方式

Python 中字符串有三种表示方式：

(1) 普通字符串：采用单引号(')或双引号(")包裹起来的字符串。

(2) 原始字符串(raw string)：在普通字符串前加 r，字符串中的特殊字符不需要转义，按照字符串的本来“面目”呈现。

(3) 长字符串：字符串中包含了换行、缩进等排版字符，可以使用三重单引号(''')或三重双引号(""")包裹起来，这就是长字符串。

**1. 普通字符串**

很多程序员习惯于使用单引号(')表示字符串。下面四行示例表示的都是 Hello World 字符串。

```
'Hello World'
"Hello World"
'\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064'      ①
"\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064"      ②
```

Python 中的字符采用 Unicode 编码，所以字符串可以包含中文等亚洲字符。代码第①行和第②行的字符串是用 Unicode 编码表示的字符串，事实上它表示的也是 Hello World 字符串，若通过 print 函数将 Unicode 编码表示的字符串输出到控制台，则会看到 Hello World 字符串。在 Python Shell 中运行示例如下：

```
>>> s = 'Hello World'
>>> print(s)
Hello World
>>> s = "Hello World"
>>> print(s)
Hello World
>>> s = '\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064'
>>> print(s)
Hello World
>>> s = "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\u0064"
>>> print(s)
Hello World
```

如果想在字符串中包含一些特殊的字符，例如换行符、制表符等，需要在普通字符串中转义，在字符串前面要加上反斜杠(\)，这称为字符转义。表 5-2 所示是常用的几个转义符。

**表 5-2 转义符**

| 字符表示 | Unicode 编码 | 说　明 |
|---|---|---|
| \t | \u0009 | 水平制表符 |
| \n | \u000a | 换行 |
| \r | \u000d | 回车 |
| \" | \u0022 | 双引号 |
| \' | \u0027 | 单引号 |
| \\ | \u005c | 反斜线 |

在 Python Shell 中运行示例如下：

```
>>> s = 'Hello\n World'
>>> print(s)
Hello
 World
>>> s = 'Hello\t World'
>>> print(s)
Hello World
>>> s = 'Hello\' World'
>>> print(s)
Hello' World
>>> s = "Hello' World"                                    ①
>>> print(s)
Hello' World
>>> s = 'Hello" World'                                    ②
>>> print(s)
Hello" World
>>> s = 'Hello\\ World'                                   ③
>>> print(s)
Hello\ World
>>> s = 'Hello\u005c World'                               ④
>>> print(s)
Hello\ World
```

对于字符串中的单引号(')和双引号(")，也可以不用转义符。可以在包含单引号的字符串中使用双引号包裹字符串，见代码第①行；可以在包含双引号的字符串中使用单引号包裹字符串，见代码第②行。另外，可以使用 Unicode 编码替代需要转义的特殊字符，代码第④行与代码第③行是等价的。

**2. 原始字符串(raw string)**

在普通字符串前面加字母 r，表示字符串是原始字符串。原始字符串直接按照字符串的字面意思来使用，没有转义字符。在 Python Shell 中运行示例代码如下：

```
>>> s = 'Hello\tWorld'                                    ①
>>> print(s)
Hello World
>>> s = r'Hello\tWorld'                                   ②
>>> print(s)
Hello\tWorld
```

代码第①行是普通字符串，代码第②行是原始字符串，它们的区别只是在字符串前面加字母 r。从输出结果可见，原始字符串中的\t 没有被当作制表符使用。

**3. 长字符串**

若字符串中包含了换行、缩进等排版字符，则可以使用长字符串。在 Python Shell 中运行示例代码如下：

```
>>> s = '''Hello
 World'''
>>> print(s)
```

```
Hello
 World
>>> s = """ Hello \t                                                    ①
           World"""
>>> print(s)
 Hello
           World
```

长字符串中如果包含特殊字符，也是需要转义的，见代码第①行。

### 5.3.2 字符串格式化

在实际的编程过程中，经常会遇到将其他类型变量与字符串拼接到一起并进行格式化输出的情况。例如计算的金额需要保留小数点后四位、数字需要右对齐等，这些都需要进行字符串格式化。

字符串格式化需要使用字符串的 format()方法以及占位符。在 Python Shell 中运行示例如下：

```
>>> name = 'Mary'
>>> age = 18
>>> s = '她的年龄是{0}岁。'.format(age)                                  ①
>>> print(s)
她的年龄是 18 岁。
>>> s = '{0}芳龄是{1}岁。'.format(name, age)                             ②
>>> print(s)
Mary 芳龄是 18 岁。
>>> s = '{1}芳龄是{0}岁。'.format(age, name)                             ③
>>> print(s)
Mary 芳龄是 18 岁。
>>> s = '{n}芳龄是{a}岁。'.format(n = name, a = age)                     ④
>>> print(s)
Mary 芳龄是 18 岁。
```

字符串中可以有占位符({}表示的内容)，配合 format()方法使用会将 format()方法中的参数替换为占位符的内容。占位符可以用参数索引表示，见代码第①行、第②行和第③行；也可以使用参数的名字表示占位符，见代码第④行，n 和 a 都是参数名字。

占位符还包括格式化控制符，可对字符串的格式进行更加精准的控制。不同的数据类型在进行格式化时需要不同的控制符，这些格式化控制符如表 5-3 所示。

**表 5-3 字符串格式化控制符**

| 控 制 符 | 说 明 |
|---|---|
| s | 字符串格式化 |
| d | 十进制整数 |
| f、F | 十进制浮点数 |
| g、G | 十进制整数或浮点数 |
| e、E | 科学计数法表示浮点数 |
| o | 八进制整数，符号是小写英文字母 o |
| x、X | 十六进制整数，x 是小写表示，X 是大写表示 |

格式化控制符位于占位符索引或占位符名字的后面，之间用冒号分隔，例如{1:d}表示索引为1的占位符格式参数是十进制整数。在Python Shell中运行示例如下：

```
>>> name = 'Mary'
>>> age = 18
>>> money = 1234.5678
>>> "{0}芳龄是{1:d}岁。".format(name, age)                    ①
'Mary芳龄是18岁。'
>>> "{1}芳龄是{0:5d}岁。".format(age, name)                    ②
'Mary芳龄是   18岁。'
>>> "{0}今天收入是{1:f}元。".format(name, money)              ③
'Mary今天收入是1234.567800元。'
>>> "{0}今天收入是{1:.2f}元。".format(name, money)            ④
'Mary今天收入是1234.57元。'
>>> "{0}今天收入是{1:10.2f}元。".format(name, money)          ⑤
'Mary今天收入是 1234.57元。'
>>> "{0}今天收入是{1:g}元。".format(name, money)
'Mary今天收入是1234.57元。'
>>> "{0}今天收入是{1:G}元。".format(name, money)
'Mary今天收入是1234.57元。'
>>> "{0}今天收入是{1:e}元。".format(name, money)
'Mary今天收入是1.234568e+03元。'
>>> "{0}今天收入是{1:E}元。".format(name, money)
'Mary今天收入是1.234568E+03元。'
>>> '十进制数{0:d}的八进制表示为{0:o},十六进制表示为{0:x}'.format(28)
'十进制数28的八进制表示为34,十六进制表示为1c'
```

上述代码第①行中{1:d}是格式化十进制整数。代码第②行中{0:5d}是指定输出长度为5的字符串，若不足则用空格补齐。代码第③行中{1:f}是格式化十进制浮点数，从输出的结果可见，小数部分过长。如果想控制小数部分可以使用代码第④行的{1:.2f}占位符，其中.2f表示保留两位小数(四舍五入)。如果想设置长度可以使用代码第⑤行的{1:10.2f}占位符，其中10表示总长度，包括小数点和小数部分，若不足则用空格补位。

### 5.3.3　字符串查找

在给定的字符串中查找子字符串是比较常见的操作。字符串类(str)中提供了find和rfind方法用于查找子字符串，返回值是查找子字符串所在的位置，没有找到则返回－1。下面具体说明find和rfind方法。

(1) str.find(sub[,start[,end]])。在索引start到end之间查找子字符串sub，如果找到则返回最左端位置的索引，如果没有找到则返回－1。start是开始索引，end是结束索引，这两个参数都可以省略。如果start省略则说明查找从字符串头开始；如果end省略则说明查找到字符串尾结束；如果全部省略则查找全部字符串。

(2) str.rfind(sub[,start[,end]])。与find方法类似，区别是如果找到子字符串则返回最右端位置的索引。如果在查找的范围内只找到一处子字符串，那么find和rfind方法的返回值是相同的。

**提示**　在Python文档中[]表示可以省略的部分，find和rfind方法的参数[,start

[,end]]表示 start 和 end 都可以省略。

在 Python Shell 中运行示例代码如下：

```
>>> source_str = "There is a string accessing example."
>>> len(source_str)                                          ①
36
>>> source_str[16]                                           ②
'g'
>>> source_str.find('r')
3
>>> source_str.rfind('r')
13
>>> source_str.find('ing')
14
>>> source_str.rfind('ing')
24
>>> source_str.find('e', 15)
21
>>> source_str.rfind('e', 15)
34
>>> source_str.find('ing', 5)
14
>>> source_str.rfind('ing', 5)
24
>>> source_str.find('ing', 18, 28)
24
>>> source_str.rfind('ingg', 5)
-1
```

上述代码第①行中的 len(source_str)返回字符串长度，注意 len 是函数，不是查找字符串的一个方法，它的参数是字符串。代码第②行中的 source_str[16]访问字符串中索引为 16 的字符。

上述字符串查找方法比较类似，这里重点解释一下 source_str.find('ing',5)和 source_str.rfind('ing',5)表达式。从图 5-1 可见，ing 字符串出现过两次，索引分别是 14 和 24。source_str.find('ing',5)返回最左端索引 14，source_str.rfind('ing',5)返回最右端索引 24。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T | h | e | r | e |  | i | s |  | a |  | s | t | r | i | n | g |  | a | c | c | e | s | s | i | n | g |  | e | x | a | m | p | l | e | . |

图 5-1　source_str 字符串索引

**提示**　函数与方法的区别是，方法是定义在类中的函数，在类的外部调用时需要通过类或对象调用，例如上述代码 source_str.find('r')就是调用字符串对象 source_str 的 find 方法，find 方法是在 str 类中定义的。而一般情况下函数不是在类中定义的，也称为顶层函数，它们不属于任何一个类，调用时直接使用函数即可，例如上述代码中的 len(source_str)就调用了 len 函数，只不过它的参数是字符串对象 source_str。

### 5.3.4　字符串与数字相互转换

在实际的编程过程中，经常会用到字符串与数字相互转换。下面从两个不同的方面介绍字符串与数字相互转换。

**1. 字符串转换为数字**

字符串转换为数字可以使用 int()和 float()实现，5.2.2 节介绍了这两个函数如何实现数字类型之间的转换。事实上这两个函数也可以接收字符串参数，如果字符串能成功转换为数字，则返回数字，否则引发异常。

在 Python Shell 中运行示例代码如下：

```
>>> int('9')
9
>>> int('9.6')
Traceback(most recent call last):
  File "<pyshell#2>", line 1, in <module>
    int('9.6')
ValueError: invalid literal for int() with base 10: '9.6'
>>> float('9.6')
9.6
>>> int('AB')
Traceback(most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int('AB')
ValueError: invalid literal for int() with base 10: 'AB'
>>>
```

默认情况下 int()函数都将字符串参数当作十进制数字进行转换，所以 int('AB')会转换失败。int()函数也可以指定基数(进制)，在 Python Shell 中运行示例如下：

```
>>> int('AB', 16)
171
```

**2. 数字转换为字符串**

数字转换字符串有很多种方法，5.3.2 节介绍了字符串格式化可以将数字转换为字符串。另外，Python 中提供的 str()函数也可将数字转化为字符串。

str()函数可以将任何类型的数字转换为字符串。在 Python Shell 中运行示例代码如下：

```
>>> str(3.24)
'3.24'
>>> str(True)
'True'
>>> str([])
'[]'
>>> str([1,2,3])
'[1, 2, 3]'
>>> str(34)
'34'
```

从上述代码可知 str()函数很强大，什么类型都可以转换。但缺点是不能格式化，如果格式化字符串需要使用 format()函数。在 Python Shell 中运行示例代码如下：

```
>>> '{0:.2f}'.format(3.24)
'3.24'
>>> '{:.1f}'.format(3.24)
'3.2'
>>> '{:10.1f}'.format(3.24)
'       3.2'
```

**提示** 在格式化字符串时，如果只有一个参数，占位符索引可以省略。

## 5.4 本章小结

本章主要介绍了 Python 中的数据类型。读者需要重点掌握数字类型与字符串类型，熟悉数字类型的互相转换，以及数字类型与字符串之间的转换。

## 5.5 同步练习

1. 在 Python 中字符串的表示方式是(　　)。

A. 采用单引号(')包裹　　B. 采用双引号(")包裹

C. 采用三重单引号('')包裹　　D. 以上都不是

2. 下列表示数字正确的是(　　)。

A. 29　　B. 0X1C　　C. 0x1A

D. 1.96e－2　　E. 9_600_000

3. 判断对错：Python 中布尔类型只有 True 和 False 两个值。(　　)

4. 判断对错：bool()函数可以将 None、0、0.0、0j(复数)、''(空字符串)、[](空列表)、()(空元组)和{}(空字典)这些数值都转换为 False。(　　)