

第3节 信息的处理

本节知识

- ◆ 数据类型及其运算
- ◆ 变量与赋值

本节活动

- ◆ 字符串型与浮点型数据的理解
- ◆ 算术运算表达式的验证实验
- ◆ Python 变量名规范性分析
- ◆ 矩形周长计算程序的编程实践

从诞生之初为了实现自动计算，到后来能够实现文字、图片及声音等信息的自动处理，计算机在功能方面有了巨大的突破，并推动了信息社会的形成。究其原因，在于人们找到了实现信息编码及其运算处理的方法。

一、数据类型及其运算

计算机中的所有信息都是以二进制数据形式编码并存放在计算机中，以便程序进行读取与运算等操作。为了帮助程序识别不同类型的信息并实现高效的运算，多数计算机语言都会制定相应的数据类型与配套的运算规则。

1. 数据类型

计算机语言大多从计算机的角度细分生活中的信息，其定义的数据类型也都大同小异。以 Python 为例，数值信息就被其细分为整型与浮点型等，而文字符号信息则被定义成字符串型，另外还有专为表示真假信息而定义的布尔型（见表 2.3.1）等，与其他计算机语言相差不大。



表 2.3.1 Python 的部分数据类型

| 数据类型 | 实例表示 |
|-------------|---------------------------------|
| 整型 (int) | 数学中的整数, 如 -1, 0, 2 |
| 浮点型 (float) | 数学中的小数, 如 2.0, 3.14 |
| 字符串型 (str) | 用各种英文引号作为界定, 如 '1', "abc", "12" |
| 布尔型 (bool) | 只定义真和假两种数值, 真为 True, 假为 False |

问题讨论

生活中的各种票据, 通常会印有使用数字来表达的信息, 如图 2.3.1 所示的发票代码、发票号码与票价等。虽然同为阿拉伯数字, 但人们都可以识别与区分它们的具体含义。如果将这些信息交给计算机来处理, 却通常要用字符串型来存储发票代码与发票号码信息, 而另用浮点型或整型来存储票价信息, 请分析并讨论具体原因。



图 2.3.1 票据中的数字信息处理

2. 算术运算

计算机在实现整型或浮点型数据运算时, 必须遵循数学的算术运算规则。以 Python 为例, 其部分运算符及实例表示如表 2.3.2 所示。



表 2.3.2 Python 的部分算术运算

| 运算符 | 名 称 | 示 例 | 输 出 | 优先级别 |
|-----|-------|----------------------------|-----|------|
| + | 加法运算符 | <code>print(3 + 2)</code> | 5 | 3 |
| - | 减法运算符 | <code>print(3 - 2)</code> | 1 | 3 |
| * | 乘法运算符 | <code>print(3 * 2)</code> | 6 | 2 |
| / | 除法运算符 | <code>print(3 / 2)</code> | 1.5 | 2 |
| // | 整除运算符 | <code>print(3 // 2)</code> | 1 | 2 |
| % | 求余运算符 | <code>print(3 % 2)</code> | 1 | 2 |
| ** | 乘方运算符 | <code>print(3**2)</code> | 9 | 1 |

拓 展 阅 读

Python 代码中的整除、求余、乘方与运算优先级别

整除、求余与乘方都是数学中常见的算术运算。在 Python 代码中，整除通常是指两个整数相除之后，取商的整数部分；求余通常是指两个整数相除后取余数；乘方通常是指若干个相同因数相乘的运算。而运算优先级别是指在一个 Python 表达式中，不同运算符的先后运算执行顺序。高级别的运算符会先于低级别的执行，若运算符级别相同时，则按照从左到右顺序进行依次运算。



实验活动

算术运算表达式的验证

算术运算中的优先级别是一种统一的约定，级别高的先运行，级别低的后运行。由于书写方式与数学有所不同，新手在编写较为复杂的算术表达式代码时，常因为容易犯错而需对其进行验证。请



完成算术运算表达式的验证活动，并填写表 2.3.3。

实验内容：验证较复杂的算术运算表达式。

实验准备：安装 Python 集成开发环境的计算机、复杂算术表达式。

表 2.3.3 较复杂算术运算表达式验证记录表

| 数学算式 | | Python 表达式 | | 结论及建议 |
|-------------------|----|-----------------------------|-----|--|
| 式子 | 答案 | 代码 | 运行 | |
| $\frac{2+1}{1+2}$ | 1 | <code>print(2+1/1+2)</code> | 5.0 | 错误。正确代码为 <code>print((2+1)/(1+2))</code> |
| | | | | |
| | | | | |

3. 字符串运算

字符串运算是处理字符串型数据的运算。简单的字符串运算有连接、重复及判断等，相关运算符及其实例表示如表 2.3.4 所示。

表 2.3.4 Python 的部分字符串运算

| 运算符 | 作用 | 实例表示 | 输出 |
|-----|----|--|------|
| + | 连接 | <code>print('中 '+'国')</code> # 实现两个字符串连接 | 中国 |
| * | 重复 | <code>print('H' * 2)</code> # 重复输出字符串 | HH |
| in | 判断 | <code>print('t' in 'steven')</code> # 判断 't' 是否在 'steven' 中，如果是则为 True，否则为 False | True |

字符串还有转义、格式化和切片等“运算”。

在 Python 中，可以用 “\” “'” “\” 分别输出单引号、双引号和右斜杠，其中 “\” 表示转义字符。如使用代码 “`print('\ ' \')`”，可以输出 “' '\””。常用的还有 “\n” 表示换行，“\r” 表示光标移到行首。

Python 提供了多种字符串格式化的方法，`str.format()` 是其中之一。如语句



“`r = '{} + {} = {}'.format(3, 4, 7)`”中，“`.format`”前是字符串模板，紧跟的括号中是用逗号分隔开的参数列表。字符串模板中的每对“`{}`”是一个占位符，由参数列表中的参数依序替换（见图 2.3.2），对字符串进行格式化后，`r` 的值为“`3 + 4 = 7`”。也可以在大括号内标出参数序号，占位符将被相应序号的参数替换，如使用语句“`r = '{0} + {1} = {2}'.format(3, 4, 7)`”后的结果也是 `r` 的值为“`3 + 4 = 7`”。

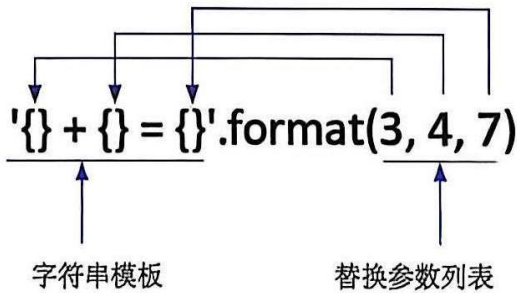


图 2.3.2 str.format() 字符串格式化示意图

字符串格式化的结果是一个字符串，常常与输出语句一起使用（见图 2.3.3）。

```
In [1]: 1 r = '{} + {} = {}'.format(3, 4, 7)
        2 print(r)
        3 + 4 = 7

In [2]: 1 print('{} + {} = {}'.format(3, 4, 7))
        3 + 4 = 7
```

图 2.3.3 输出语句中使用格式化字符串的两种方法

字符串模板中的占位符，还可以设置成各种格式，表 2.3.5 列出了部分数值格式。

表 2.3.5 字符串格式化中的部分数值格式

| 数 值 | 格 式 | 输 出 | 描 述 |
|-----|---------|--------|-------------------|
| 1 | {:.2f} | 1.00 | 保留小数点后两位 |
| - 1 | {:+.2f} | - 1.00 | 带符号保留小数点后两位 |
| 1 | {:0>2d} | 01 | 数字补零（填充左边，宽度为 2） |
| 1 | {:x<4d} | 1xxx | 数字补 x（填充右边，宽度为 4） |

字符串的切片就是获取字符串的一部分的操作，语句常用格式举例如表 2.3.6 所示。

表 2.3.6 字符串切片格式举例

| 字符串 | 举 例 | 返回值 |
|--------------|----------|------------|
| s='01234567' | s[0] | '0' |
| | s[3] | '3' |
| | s[::] | '01234567' |
| | s[0:8:1] | '01234567' |
| | s[1:7:1] | '123456' |
| | s[1:7:2] | '135' |
| | s[1::] | '1234567' |
| | s[:7:] | '0123456' |

拓展阅读

数据类型与运算符的匹配要求

通常情况下，Python 的数据类型要与其运算符相匹配，否则可能会出现错误，如图 2.3.4 所示。

```

2.3.2 数据类型与运算符不匹配.py
1 print(1 + '2')    #1为整型，'2'为字符串

Shell
Python 3.7.9 (bundled)
>>> %Run '2.3.2数据类型与运算符不匹配.py'
Traceback (most recent call last):
  File "C:\Users\makespace01\Desktop\
    print(1 + '2')    #1为整型，'2'为字符串
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
Python 3.7.9
  
```

图 2.3.4 数据类型与运算符不匹配

