第2章

Python基础知识

拓展阅读

学习目标

- 熟悉代码格式，能够归纳注释、缩进和语句换行的规范
- 熟悉标识符的命名规则，能够在程序中使用合法的标识符
- 熟悉关键字，能够列举至少 5 个关键字
- 掌握变量的定义与使用，能够熟练定义并使用变量访问数据
- 了解常用的数据类型，能够掌握每种数据类型的特点
- 掌握变量的输入与输出，能够通过 input()和 print()函数分别实现输入与输出的功能
- 掌握数字类型，能够在程序中正确处理数字类型的数据
- 掌握数字类型的转换，能够根据需要进行数字类型的转换
- 掌握运算符的使用，能够根据需要选择运算符进行运算

“万丈高楼平地起”，学习 Python 编程也是同样的道理。在开始编写 Python 程序之前，我们需要先掌握 Python 的基础知识。只有把基础打牢才能进入更高的学习阶段。本章将针对 Python 的基础知识，包括代码格式、标识符、关键字、变量、数据类型、数字类型和运算符等进行详细讲解。

2.1 代码格式

良好的代码格式可以显著提升代码的可读性。在 Python 中，代码格式不仅是一种规范，也是 Python 语法的一部分。不符合正确格式规范的 Python 代码可能无法正常执行。为了帮助读者编写规范的代码，本节将从注释、缩进和语句换行这 3 个方面对 Python 的代码格式进行讲解。

2.1.1 注释

注释是代码中的辅助性文字，用于标识代码的含义与功能，提高代码的可读性。注释在程序执行时会被 Python 解释器自动忽略，不会对程序产生任何影响。Python 程序中的注释分

为单行注释和多行注释，下面分别介绍这两种注释的格式和功能。

1. 单行注释

单行注释以“#”开头，“#”后面的内容用于说明当前行或当前行之后的代码的功能。单行注释既可以单独占一行，也可以位于注释的代码之后，与注释的代码共占一行。示例代码如下：

```
# 我是单行注释
print('志当存高远')                    # 我也是单行注释
```

为了保证注释的可读性，Python 官方建议在“#”后面先添加一个空格，再添加相应的说明文字；若单行注释与其注释的代码共占一行，注释和代码之间至少应有两个空格。

2. 多行注释

多行注释是由 3 对双引号或单引号包裹的内容，主要用于说明函数或类的功能，因此多行注释也被称为说明文档。例如，Python 的内置函数 print()对应的多行注释如下：

```
"""
Prints the values to a stream, or to sys.stdout by default.
  sep
    string inserted between values, default a space.
  end
    string appended after the last value, default a newline.
  file
    a file-like object (stream); defaults to the current sys.stdout.
  flush
    whether to forcibly flush the stream.
"""
```

2.1.2　缩进

Python 使用缩进来确定代码之间的逻辑关系和层次关系，缩进指的是一行代码之前的空白区域。Python 中可以使用两种方法来控制缩进，分别是使用空格和使用 Tab 键，其中使用空格是首选的缩进方法，一般使用 4 个空格表示一级缩进。Python 不允许混合使用空格和 Tab 键来控制缩进。添加缩进后的代码从属于其上最近的一行非缩进或非同等级别缩进的代码，示例代码如下：

```
if True:
   print("True")
else:
   print("False")
```

上述代码中，第 2 行代码从属于第 1 行代码；第 4 行代码从属于第 3 行代码。

代码缩进量的不同会导致代码语义的改变，Python 语言要求同一代码段的每行代码必须具有相同的缩进量。程序中不允许出现无意义或不规范的缩进，否则运行时会产生错误。示例代码如下：

```
if True:
    print("Answer")
    print("True")
else:
    print("Answer")
  print("False")                                    # 缩进量不一致，会导致运行错误
```

上述代码中，第 6 行与第 5 行代码的缩进量不一致，程序在运行时会出现错误，具体如下：

```
  File "E:\FastPrograms3\Chapter02\code01.py", line 6
    print("False")                                  # 缩进量不一致，会导致运行错误
                                                    ^
IndentationError: unindent does not match any outer indentation level
```

2.1.3 语句换行

在 Python 中，如果一条语句过长而无法在一行内完整显示，那么可以通过换行将这条语句分成多行显示，从而提高代码的可读性。Python 官方建议一行代码的长度不要超过 79 个字符，若超过最好进行换行显示。Python 中可以通过两种方式进行语句换行：一种方式是在未结束的代码的末尾加上符号“\”；另一种方式是使用小括号包裹代码。

使用符号“\”进行换行时，“\”位于未结束的代码的末尾，会连接下面的代码，构成一个语义完整的语句。示例代码如下：

```
side_01 = 3; side_02 = 4; side_03 = 5
# 使用“\”进行换行
result = side_01 + side_02 > side_03 or \
         side_02 + side_03 > side_01 or \
         side_01 + side_03 > side_02
```

使用小括号进行换行时，只需要将代码全部放到小括号中，此时小括号中的代码会被视为一个语义完整的语句，可以跨越多行显示。示例代码如下：

```
side_01 = 3; side_02 = 4; side_03 = 5
# 使用小括号进行换行
result = (side_01 + side_02 > side_03 or
          side_02 + side_03 > side_01 or
          side_01 + side_03 > side_02)
```

需要注意的是，如果代码中有小括号、中括号或大括号包裹的内容，那么可以对这些内容直接换行，不需要使用符号“\”或者小括号，这是因为 Python 默认会将小括号、中括号或大括号包裹的多行内容自动进行隐式连接。示例代码如下：

```
# 小括号包裹的内容进行换行显示
demo_one = ('one', 'two', 'three', 'four', 'five',
            'six', 'seven', 'eight', 'nine', 'ten')
# 中括号包裹的内容进行换行显示
demo_two = ['one', 'two', 'three', 'four', 'five',
           'six', 'seven', 'eight', 'nine', 'ten']
# 大括号包裹的内容进行换行显示
demo_thr = {'one': '壹', 'two': '贰', 'three': '叁', 'four': '肆',
            'five': '伍', 'six': '陆', 'seven': '柒', 'eight': '捌',
            'nine': '玖', 'ten': '拾'}
```

2.2 标识符和关键字

2.2.1 标识符

在现实生活中，为了方便交流，人们会用不同的名称标记不同的事物。例如，人们使用橘子、苹果、柠檬等名称标记不同的水果，当提到某水果名称时，人们自然就会明白它指代的是哪种水果。一些水果及其名称标识如图 2-1 所示。

图2-1　一些水果及其名称标识

同理，为了明确某行代码使用的到底是哪个数据，代表的是哪一类信息，开发人员可以使用一些符号或名称作为程序中同一个数据或同一类信息的标识，这些符号或名称就是标识符，比如后面会介绍的变量名、函数名、类名等。

不以规矩，不能成方圆。Python 中的标识符在命名时，需要遵守一定的命名规则，具体如下。

- Python 中的标识符由字母、数字或下画线组成，且不能以数字开头。
- Python 中的标识符区分大小写。例如，andy 和 Andy 是不同的标识符。
- Python 不允许使用关键字作为标识符。关键字将在 2.2.2 小节介绍。

下面列举一些合法的标识符，具体如下：

```
student                    # 全部是小写字母
LEVEL                      # 全部是大写字母
Flower                     # 大写字母、小写字母混合
Flower123                  # 大写字母、小写字母、数字混合
Flower_123                 # 大写字母、小写字母、数字、下画线混合
```

下面列举一些不合法的标识符，具体如下：

```
from#12                    # 标识符不能包含#符号
2ndObj                     # 标识符不能以数字开头
if                         # if 是关键字，不能作为标识符
```

除了以上规则外，对于 Python 标识符的命名还有以下两点建议。

（1）见名知意。标识符应具有明确的含义，尽量做到用户一看便知道标识符的意义。例如，使用 name 标识姓名，使用 student 标识学生。

（2）命名格式规则。为了区分程序中不同地方使用的标识符，Python 有一些约定俗成的命名格式规则，涵盖了变量名、常量名、模块名、函数名和类名，具体规则如下。

- 变量名使用小写的单个单词或由下画线连接的多个单词，如 name、native_place。
- 常量名使用大写的单个单词或由下画线连接的多个单词，如 ORDER_LIST_LIMIT、GAME_LEVEL。
- 模块名、函数名使用小写的单个单词或由下画线连接的多个单词，如 low_with_under、generate_random_numbers。
- 类名使用以大写字母开头的单个或多个单词，如 Cat、CapWorld。

尽管这些命名格式规则是约定俗成的，但在编程中保持一致的命名风格是一个好习惯。作为一名合格的开发人员，我们要始终保持严谨的学习态度，培养良好的编程习惯，并努力编写高质量的代码。这些努力将为我们的编程之旅奠定坚实的基础，并帮助我们成长为更好的开发人员。

2.2.2 关键字

关键字是 Python 已经使用的且不允许开发人员重复定义的标识符。Python 中一共定义了 35 个关键字，这些关键字全部存储在 keyword 模块的变量 kwlist 中，通过变量 kwlist 可获取 Python 中所有的关键字，示例代码如下：

```
import keyword
print(keyword.kwlist)
```

运行代码，将获取到的所有关键字按照一定格式排列，具体如下所示：

```
False       await       else        import      pass        None
break       except      in          raise       True        class
finally     is          return      and         continue    for
lambda      try         as          def         from        nonlocal
while       assert      del         global      not         with
async       elif        if          or          yield
```

Python 中的每个关键字都有不同的作用，通过“help("关键字")”可以查看关键字的详细信息。例如，查看关键字 import 的详细信息，示例代码如下：

```
print(help("import"))
```

运行代码，结果如下所示：

```
The "import" statement
**********************
import_stmt    ::= "import" module ["as" identifier]
    ("," module ["as" identifier])*
     | "from" relative_module "import" identifier ["as" identifier]
    ("," identifier ["as" identifier])*
     | "from" relative_module "import" "(" identifier ["as" identifier]
    ("," identifier ["as" identifier])* [","] ")"
     | "from" relative_module "import" "*"
module         ::= (identifier ".")* identifier
relative_module ::= "."* module | "."+
…
```

2.3 变量和数据类型

2.2.1 变量

计算机语言中变量的概念源于数学。在数学中，变量指用拉丁字母表示的、值不固定的数据。在计算机语言中，变量指能存储计算结果或表示值的抽象概念。程序在运行期间用到的数据会被保存在计算机的内存单元中，为了方便存取内存单元中的数据，Python 使用标识符来标识不同的内存单元，从而在程序中通过标识符来引用和操作存储在内存单元中的数据。

下面以存储数据 15 的变量和存储数据 20 的变量为例，它们对应的标识符分别为 num 和 data，变量与内存单元之间的关系如图 2-2 所示。

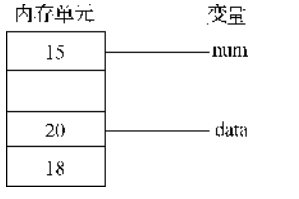

图2-2 变量与内存单元之间的关系

标识内存单元的标识符又称为变量名，Python 通过赋值运算符“=”将内存单元中存储的数据与变量名建立联系，即定义变量，具体语法格式如下：

```
变量名 = 值
```

当定义了一个变量并将一个值赋给它时，Python 会在内存中为该值分配一个合适大小的内存单元，并将变量与内存单元进行关联。例如，将内存单元中存储的值 100 与变量名 data 建立联系，代码如下：

```
data = 100
```

此时若要使用变量，则需要通过变量名访问存储在内存中与该变量关联的值。例如，通过变量名 data 访问其对应的值，代码如下：

```
print(data)
```

上述代码中的 print()是一个内置函数，该函数会将变量 data 保存的数据输出到控制台上，此处读者只需要了解 print()的功能即可，在 2.3.3 小节会对其进行详细介绍。

运行代码，结果如下所示：

```
100
```

2.3.2　数据类型

根据数据存储形式的不同，Python 中常用的数据类型可以分为两类，分别是数字类型和组合数据类型，其中组合数据类型相对复杂，包括字符串类型、列表类型、元组类型、集合类型、字典类型等。下面先介绍常用的数据类型。

1. 数字类型

数字类型用于表示不同种类的数值数据，分为整数类型（简称“整型”，int）、浮点型（float）、复数类型（complex）和布尔类型（bool），其中整型、浮点型和复数类型的数据分别对应数学中的整数、实数和复数；布尔类型比较特殊，用于表示真（True）或假（False）的逻辑值。

数字类型数据的示例如下：

```
0           # 整型
101         # 整型
-239        # 整型
3.1415      # 浮点型
4.2E-10     # 浮点型
-2.334E-9   # 浮点型
3.12+1.23j  # 复数类型
-1.23-93j   # 复数类型
True        # 布尔类型
False       # 布尔类型
```

2. 字符串类型

字符串类型用于表示文本数据，由单引号、双引号或者三引号包裹一系列字符。示例代码如下：

```
'Python123￥'                                    # 使用单引号包裹
"Python4*&%"                                     # 使用双引号包裹
'''Python s1 ~(())'''                            # 使用三引号包裹
```

3. 列表类型

列表类型可以保存任意数量、任意类型的元素，这些元素是有顺序、可重复的，并且可以被修改为其他元素。Python 中一般使用中括号“[]”创建列表，在中括号中可以放入多个元素，多个元素以英文逗号进行分隔，示例代码如下：

```
[1, 2, 'apple']                                  # 这是一个列表
```

4. 元组类型

元组类型与列表的作用相似，它也可以保存任意数量、任意类型的元素，这些元素是有顺序的、可重复的，但是不可以被修改。Python 中一般使用小括号“()”创建元组，在小括号中可以放入多个元素，多个元素以英文逗号分隔，示例代码如下：

```
(1, 2, 'apple')                                    # 这是一个元组
```

5. 集合类型

集合类型与列表、元组类似，它也可以保存任意数量、任意类型的元素，但是这些元素之间没有特定的顺序，并且每个元素必须是唯一的。Python 中一般使用大括号“{}”创建集合，在大括号中可以放入多个元素，多个元素以英文逗号分隔，示例代码如下：

```
{'apple', 'orange', 1}                             # 这是一个集合
```

6. 字典类型

字典类型可以保存任意数量的元素，不过元素是“Key:Value”形式的键值对，键不能重复。Python 中一般使用大括号“{}”创建字典，在大括号中可以放入多个元素，多个元素以英文逗号分隔，示例代码如下：

```
{"name": "张三", "age": 18}                        # 这是一个字典
```

多学一招：type()函数——查看变量的类型

Python 是一种动态类型语言，在定义变量时不需要指定其具体的类型。Python 解释器会根据程序执行时赋予变量的值自动确定变量的数据类型，可以通过 type()函数查看变量所存储的数据的具体类型。

例如，创建一个字典，通过 type()函数查看这个字典的数据类型，具体代码如下：

```
dict_demo = {"name": "zhangsan", "age": 18}
print(type(dict_demo))                          # 查看数据类型
```

运行代码，结果如下所示：

```
<class 'dict'>
```

由以上输出结果可知，变量 dict_demo 中保存的数据的类型是 dict。

2.3.3 变量的输入与输出

程序要实现人机交互功能，需要从输入设备接收用户输入的数据，且向显示设备输出数据。Python 提供了 input()函数和 print()函数分别实现数据的输入与输出。

1. input()函数

input()函数用于接收用户从键盘输入的数据，返回一个字符串类型的数据，其语法格式如下所示：

```
input([prompt])
```

以上语法格式中的 prompt 是 input()函数的参数，用于设置接收用户输入时的提示信息，可以省略。

下面演示 input()函数的用法，示例代码如下：

```
user_name = input('请输入账号：')      # 接收用户输入的账号
password = input('请输入密码：')       # 接收用户输入的密码
print('登录成功！')
```

运行代码，根据提示在控制台中输入账号，输入完成后按回车键，然后输入密码，再次按回车键，运行结果如下：

```
请输入账号：itcast
请输入密码：123456
登录成功!
```

2. print()函数

print()函数用于向控制台中输出数据，它可以输出任何类型的数据，其语法格式如下：

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

以上语法格式中各参数的含义如下。

- *objects：表示输出的数据。输出多个数据时，数据需要用英文逗号分隔。
- sep：可选参数，用于设定数据之间使用的分隔符，默认值为空格。
- end：可选参数，用于设定输出结果以什么结尾，默认值为换行符 \n。
- file：可选参数，表示数据要写入的文件对象，默认值为 sys.stdout（表示标准输出文件，默认情况下程序会将结果输出到控制台）。
- flush：可选参数，表示是否刷新标准输出流，默认值为 False（表示不刷新）。

接下来，通过输出个人基本信息的示例演示 print()函数的用法，示例代码如下：

```
name = '小明'
print('姓名：', name)      # 输出多个数据，第一个是字符串类型的数据，第二个是变量保存的数据
print('年龄：22')          # 输出一个字符串类型的数据
address = '北京'
print('地址：', address)   # 输出多个数据，第一个是字符串类型的数据，第二个是变量保存的数据
```

运行代码，结果如下所示：

```
姓名： 小明
年龄：22
地址： 北京
```

2.4　实训案例

2.4.1　输出购物小票

输出购物小票

购物小票又称购物收据，是指消费者购买商品时由商场或其他商业机构提供给用户的消费凭证。购物小票中一般包含用户购买的商品名称、数量、单价及总额等信息。某用户在某商场购买商品的购物小票如图 2-3 所示。

图2–3　购物小票

本案例要求编写代码，依次接收用户从键盘输入的商品价格，并根据价格输出图 2-3 所示的购物小票。

2.4.2 输出植树证书

蚂蚁森林是某公司客户端发起的“碳账户”的一款公益活动，用户通过步行、地铁出行、在线消费等行为，可在蚂蚁森林中获取能量，当能量达到一定数值后，用户可以在支付宝中申请种植一棵树，申请成功后会收到某公司发放的一张植树证书。植树证书中包含树苗编号、申请时间、种植地点等信息，具体如图 2-4 所示。

输出植树证书

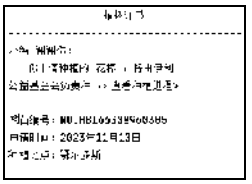

图2-4 植树证书

本案例要求编写代码，根据用户输入的昵称和植物名称实现输出图 2-4 所示植树证书的功能。

2.5 数字类型

2.5.1 整型

整型用于表示整数，例如 100、101、-100、-101 等。Python 中整型数据的大小没有限制，只要计算机的内存足够大，用户就无须考虑内存溢出问题。

整型常用的记数方式有 4 种，分别是二进制、八进制、十进制和十六进制，默认的记数方式为十进制，其中二进制数以“0B”或“0b”开头，八进制数以数字“0o”或“0O”开头，十六进制数以“0x”或“0X”开头。接下来，分别以 4 种记数方式表示整型数据 5，示例代码如下：

```
5                                    # 十进制数
0b101                                # 二进制数
0o5                                  # 八进制数
0x5                                  # 十六进制数
```

为了方便用户使用不同进制的数据，Python 中提供了一些用于转换数据进制的函数，分别是 bin()、oct()、int()、hex()，这些函数都可以将一种进制的整型数据转换为其他进制的整型数据，如表 2-1 所示。

表 2-1 用于整型数据的进制转换的函数

函数	说明
bin(x)	将 x 转换为二进制数
oct(x)	将 x 转换为八进制数
int(x)	将 x 转换为十进制数
hex(x)	将 x 转换为十六进制数

下面演示表 2-1 中各函数的用法，示例代码如下：

```
decimal = 10                         # 十进制数
bin_num = 0b1010                     # 二进制数
print(bin(decimal))                  # 将十进制数 10 转换为二进制数
print(oct(decimal))                  # 将十进制数 10 转换为八进制数
print(int(bin_num))                  # 将二进制数 0b1010 转换为十进制数
print(hex(decimal))                  # 将十进制数 10 转换为十六进制数
```

运行代码，结果如下所示：

```
0b1010
0o12
10
0xa
```