第3章

数据类型

Python中的变量不需要事先声明，每个变量在使用前都必须赋值。Python中的变量不同于其他编程语言，它没有类型，所谓的“类型”是变量在内存中存储的对象的类型。

Python提供6个标准的数据类型，分别为：

- 数字(number)。
- 字符串(string)。
- 列表(list)。
- 元组(tuple)。
- 集合(set)。
- 字典(dictionary)。

其中，number、string、tuple为不可变数据；list、dictionary、set为可变数据。

3.1 数字

Python支持的数字类型有int(整型)、float(浮点型)、bool(布尔型)、complex(复数)，可以通过内置的type()函数查看变量所指的对象类型，如图3.1所示。

Python可以同时为多个变量赋值，如图3.1中的4个变量赋值，还可以写成如图3.2所示的形式。

```
a=5
b=5.1
c=True
d=1+2j
print(type(a),type(b),type(c),type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

图3.1 Python的数字数据类型

```
a,b,c,d=5,5.1,True,1+2j
print(a,b,c,d)
5 5.1 True (1+2j)
```

图3.2 Python支持同时给多个变量赋值

(1) 这里的变量a、b、c等，在程序设计语言中还有一个专业名称，称为标识符(identifier)。标识符是编程语言中允许作为对象名字的符号串。标识符分为两类：一类是系统自带的标识符，称为关键字(或保留字)，如False、None、in、def、if、try、import、global、pass等；另一类是

程序开发者自己定义的标识符。需要注意的是，不能用系统关键字来定义用户自己的标识符。

(2) Python 中标识符要求首字符必须是字母或下画线；非首字符可以是字母、数字或下画线的任意组合。

(3) Python 中一行定义多个变量有两种不同的方式：一是形如“a1,b1,c1,d1 = 1,2,3,4”；另一种采用中间用“;”隔开的形式，如“a2=5;b2=6;c2=7;d2=8”，如图 3.3 所示。

但在实际编程中并不推荐在一行定义多个变量，每一行只定义一个变量并移除分号更符号规范。

```
a1,b1,c1,d1 = 1,2,3,4
a2=5;b2=6;c2=7;d2=8
print("a1=%d b1=%d c1=%d d1=%d"%(a1,b1,c1,d1))
print("a2=%d b2=%d c2=%d d2=%d"%(a2,b2,c2,d2))
```

```
a1=1 b1=2 c1=3 d1=4
a2=5 b2=6 c2=7 d2=8
```

图 3.3　Python 支持一行定义多个变量

3.2　字符串

Python 中的字符串用单引号(')或双引号(")括起来，同时使用反斜杠(\)转义特殊字符。实际运算中，常需要对字符串进行截取操作，字符串的截取语法格式如下：

```
变量[头下标:尾下标]
```

索引值以 0 为开始值，-1 表示从末尾的开始位置。

执行以下代码：

```
s = "Python Programming"
t0,t1 = s[0:6],s[:-12]
print(t0,t1)
```

此时输出的结果为 Python Python。

一般而言，如执行 s[m:n]对字符串 s 进行截取操作，截取的子串为 s[m]到 s[n-1]的所有字符。如果省略 m，则从 s[0]开始；如果省略 n，则包含最后一个字符。

如果依次执行 s[:-12]、s[-11:]、s[7:]，则输出的结果分别为 Python、Programming、Programming。

除字符串的截取操作以外，还可以对字符串进行拼接操作，如：

```
s1,s2 = "Python"," Programming"
s1 + s2
```

输出的结果为 Python Programming。

字符串的格式化操作可以通过以下方式实现。

1. 通过“格式符+类型码”来实现

“格式符+类型码”中的“%s”表示以字符串形式显示，“%f”表示以浮点数显示，“%d”表示以整数显示。

%s 表示先占一个字符串类型的位置。占完位置之后，再以%的形式在后面补上要填充的内容，如果是多个数据，就把它们放进括号内，按顺序填充，用逗号隔开(其他类似)，如图 3.4 所示。

```
print("%s: %d %s: %d %s: %d"%("金牌",32,"银牌",21,"铜牌",16))
```

```
金牌: 32 银牌: 21 铜牌: 16
```

图 3.4　通过“格式符+类型码”实现字符串的格式化输出

2. 通过 format()函数来实现

format()函数用来占位的是大括号,不用区分类型码(%+类型码)。具体的语法为:

```
'str.format()'
```

结果如图 3.5 所示。

```
print("{} {}:{} {}:{} {}:{}\n{} {}:{} {}:{} {}:{}\n{} {}:{} {}:{} {}:{}"\
      .format("中国","金牌",32,"银牌",21,"铜牌",16,"美国","金牌",25,"银牌",29,"铜牌",21,\
              "日本","金牌",20,"银牌",7,"铜牌",11))
```

```
中国 金牌:32 银牌:21 铜牌:16
美国 金牌:25 银牌:29 铜牌:21
日本 金牌:20 银牌:7 铜牌:11
```

图 3.5 通过 format()函数实现字符串的格式化输出

当一行内容过长时,为显示及代码阅读、调试方便,可将其写成多行语句,每行用\结尾,表示该行内容还未结束。

format()函数也接收通过参数传入的数据,如图 3.6 所示。

```
china = {"金牌":32,"银牌":21,"铜牌":16}
print("金牌:{2} 银牌:{1} 铜牌:{0}".format(china["铜牌"],china["银牌"],china["金牌"]))
```

```
金牌:32 银牌:21 铜牌:16
```

图 3.6 format()函数通过参数传入数据

上述代码中{}中的数字用于指定 format()函数的参数位置,如果不指定位置,默认按顺序对应。

3.3 列表

列表的代码格式为:

```
列表名 赋值号 中括号[数据 1,数据 2,…,数据 n]
```

这里的每一个数据称为一个元素,每个元素之间用英文的逗号隔开。列表中可以存放各种不同类型的数据。

例如,环球网报道,2021 年 8 月 3 日 0~24 时,31 个省(自治区、直辖市)和新疆生产建设兵团报告新增新冠肺炎确诊病例 96 例,其中境外输入病例 25 例(云南 7 例,福建 4 例,江苏 3 例,上海 2 例,浙江 2 例,广东 2 例,天津 1 例,山西 1 例,辽宁 1 例,河南 1 例,四川 1 例),本土病例 71 例(江苏 35 例,湖南 15 例,湖北 9 例,山东 6 例,云南 3 例,河南 2 例,福建 1 例)(https://baijiahao.baidu.com/s?id=1707122193327071221&wfr=spider&for=pc)。用列表表示江苏省的新增本土比例和境外输入病例为:

```
numberofConfirmedCases = ["江苏",35,3]
```

```
for i in numberofConfirmedCases:
    print(i)
```

```
江苏
35
3
```

图 3.7 利用 for 语句遍历列表中的数据

如果要遍历列表中的每一个数据,可以采用 for 语句构造循环来实现,如图 3.7 所示。

列表有如下几种操作。

1. 从列表中提取单个元素

列表中每个元素都有自己的位置,称为偏移量,从 0 开始计,如 list1[0]。

2. 从列表中提取多个元素

方法：

```
list[m:n]
```

冒号左右两边的数字 m、n 指列表中的偏移量，通过冒号来截取列表元素的操作称为**切片**，即将列表中的某个片段拿出来处理。列表切片的口诀为：左右空，取到头；左要取，右不取。冒号左边空，就要从偏移量为 0 的元素开始取；右边空，就要取到列表的最后一个元素。

偏移量取到的是列表中的元素，而切片则是截取了列表的某部分，所以结果还是列表。

3. 向列表中添加元素

例如，有列表 local = [35,15,9,6,3,2]，通过 append()操作向该列表中添加一个元素，如图 3.8 所示。

```
local = [35,15,9,6,3,2]
local.append(1)
local
```

```
[35, 15, 9, 6, 3, 2, 1]
```

图 3.8　通过 append()实现向列表中添加数据

如果要添加多个元素，如写成 local.append(3,1)，编译器会提示报错，显示如图 3.9 所示的错误信息。

```
local.append(3,1)
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-5-9d5b6af0ac90> in <module>
----> 1 local.append(3,1)

TypeError: append() takes exactly one argument (2 given)
```

图 3.9　Python 不支持一次性往列表中添加多个元素

(1) 用 append()给列表添加元素，每次只能添加一个元素。列表中的元素可以是字符串、数字等，也可以是列表本身，即列表内部支持嵌套。

(2) append()函数后的参数只要满足数量为 1 即可，单个列表也会视作一个元素；append()后的元素会添加在列表的末尾。

(3) append()函数并不生成一个新列表，而是在列表末尾新增一个元素。而且，列表长度可变，理论容量无限，所以支持任意的嵌套。

可将上述代码修改成如图 3.10 所示的形式。

4. 删除列表中的一个或多个元素(切片操作)

删除一个元素：

```
del list[i]
```

删除多个元素：

```
del list[m:n]
```

结果如图 3.11 所示。

```
local.append([3,1])
local
```

```
[35, 15, 9, 6, 3, 2, 1, [3, 1]]
```

图 3.10　往列表中添加一个嵌套列表元素

```
del local[6]
local
```

```
[35, 15, 9, 6, 3, 2, [3, 1]]
```

```
del local[4:-1]
local
```

```
[35, 15, 9, 6, [3, 1]]
```

图 3.11　用 del 删除列表中的一个或多个元素

3.4 字典

字典适用于表示或存储名字和数值(如分数、身高、体重等)两种数据存在一一对应的情况。如,截至2021年8月3日13:00时,我国运动员代表在2020东京奥运会获得金牌32枚、银牌21枚、铜牌16枚,名列榜首。用字典表示,如图3.12所示。

```
medals={"金牌":32,"银牌":21,"铜牌":16}
medals
```

```
{'金牌': 32, '银牌': 21, '铜牌': 16}
```

图3.12 字典的应用

3.4.1 列表和字典的区别与联系

(1) 列表外层用的是中括号,字典的外层是大括号。

(2) 列表中的元素是自成一体的,而字典的元素是由**一个个键值对**构成的,用英文冒号连接。如'金牌':32,其中把'金牌'称为键(**key**),32称为值(**value**)。

(3) 可以用len()函数来获取一个列表或者字典的长度(元素个数),括号里放列表或字典名称,如图3.13所示。

(4) 字典中的键具备唯一性,而值可重复。

(5) 列表中的数据是有序排列的,如图3.14所示;而字典中的数据是随机排列的。列表有序,要用偏移量定位;字典无序,便通过唯一的键来取值。

```
len(medals)
```

```
3
```

图3.13 用len()函数来获取列表或者字典的长度

这里的"==="是关系运算符,如果"==="左右两边相等,值为True,若不相等则为False。"==="关系运算符的用法如图3.15所示。

```
medals_1=[32,21,16]
medals_2=[21,32,16]
print(medals_1==medals_2)
```

```
False
```

图3.14 列表中的数据排列有序

```
medals_1={'金牌': 33, '银牌': 21, '铜牌': 16}
medals_2={'银牌': 21, '金牌': 33, '铜牌': 16}
print(medals_1==medals_2)
```

```
True
```

图3.15 "==="关系运算符的用法

(1) Python中用于比较的关系运算符有等于(==)、不等于(!=)、大于(>)、小于(<)、大于或等于(>=)和小于或等于(<=)。

(2) Python中表示"假"的有False、0、空字符串、空列表[]、空字典{}和None。

可以使用bool()函数来查看一个数据被判断为真还是假,如图3.16所示。这个函数的用法与type()函数相似,在bool()函数的括号中放入想要判断真假的数据,然后打印出来即可。

```
print(bool(False),bool(0),bool({}),bool(''),bool([]),bool(None))
```

```
False False False False False False
```

图3.16 用bool()函数查看数据的真假(True or False)

(3) 不管是列表还是字典,对数据的访问都是通过中括号来实现的。

(4) 列表可嵌套其他列表和字典,字典也可嵌套其他字典和列表。例如,列表中嵌套字典如图3.17所示。

要获取中国的金牌总数,正确代码如图3.18所示。

```
medals=[
    {"中国":{'金牌': 32, '银牌': 21, '铜牌': 16},
     "美国":{'金牌': 25, '银牌': 29, '铜牌': 21},
     "日本":{'金牌': 20, '银牌': 7, '铜牌': 11}
    }
]
medals
```

```
[{'中国': {'金牌': 32, '银牌': 21, '铜牌': 16},
  '美国': {'金牌': 25, '银牌': 29, '铜牌': 21},
  '日本': {'金牌': 20, '银牌': 7, '铜牌': 11}}]
```

图 3.17　列表中嵌套字典

```
medals[0]["中国"]["金牌"]
```

```
32
```

图 3.18　列表中嵌套字典正确的数据获取方式

如果写成 medals["中国"]["金牌"],则会出现错误提示信息,如图 3.19 所示。

```
TypeError                                 Traceback (most recent call last)
<ipython-input-20-59c5df968277> in <module>
----> 1 medals["中国"]["金牌"]

TypeError: list indices must be integers or slices, not str
```

图 3.19　列表中嵌套字典不正确的数据引用方式

这是因为 medals 是列表中嵌套字典,列表的索引必须为整数或整数序列,medals[0]定位到列表中的第一个数据。

例如,字典中嵌套列表,如图 3.20 所示。

```
medals={
    "中国":[32,21,16],
    "美国":[25,29,21],
    "日本":[20,7,11]
}
medals["中国"][0]
```

```
32
```

图 3.20　字典中嵌套列表

3.4.2　字典的几种操作

1. 从字典中提取元素

通过键来索引,如图 3.21 所示。

这里的 medals 后只能跟[],而不能是{}。

2. 给字典增加元素

在字典 medals 中增加键值对"总数:69",需要用到赋值语句:字典名[键]=值,如图 3.22 所示。

```
print(medals["金牌"])
```

```
32
```

图 3.21　通过键从字典中提取数据

```
medals["总数"]=69
medals
```

```
{'金牌': 32, '银牌': 21, '铜牌': 16, '总数': 69}
```

图 3.22　给字典增加元素

3. 删除字典中的元素

删除字典中键值对采用 del 语句,代码为:del 字典名[键],如图 3.23 所示。

4. 修改字典中的元素

方法与给字典中增加元素一样,直接通过赋值语句对键值对重新赋值,如图 3.24 所示。

```
del medals["总数"]
medals
```

```
{'金牌': 32, '银牌': 21, '铜牌': 16}
```

图 3.23　用 del 删除字典中的元素

```
medals["金牌"]=33
medals
```

```
{'金牌': 33, '银牌': 21, '铜牌': 16}
```

图 3.24　通过赋值语句修改字典中的元素

3.5　元组

元组和列表很相似,其不同之处,元组是用小括号来表示的。

元组和列表都是序列,提取的方式也是偏移量,如 tuple1[1]、tuple1[1:]。另外,元组也支持任意的嵌套。

图 3.25 为元组应用举例。

```
tuple1 = ('A','B')
list2 = [('A','B'),('C','D'),('E','F')]
```

```
tuple1[0]
```

```
'A'
```

```
list2[1][1]
```

```
'D'
```

图 3.25　元组应用举例