

第3章

使用字符串、列表和元组

CS, ZJU

2018年12月

Overview

- ◎ 序列的访问及运算符
- ◎ 字符串使用
- ◎ 列表和元组使用

3.1 序列的访问及运算符

◎ 序列（Sequence）

为满足程序中复杂的数据表示，Python支持组合数据类型，可以将一批数据作为一个整体进行数据操作，这就是数据容器的概念。

容器中可包含多个数据（元素），容器中的数据（元素）有先后次序，每个元素通过用其下标（索引）来访问。序列的下标从0开始，后面下标依次为1, 2, 3, ...。

序列是其中一大类数据容器的统称，不是具体的数据类型。

◎ 常用的序列类型：列表（list），字符串（string），元组（tuple）

通用的序列操作

所有的序列类型都可以进行的操作归纳如下表所示。

表3-1 序列的操作

操作	描述
$X1+X2$	联接序列X1和X2,生成新序列
$X*n$	序列X重复n次, 生成新序列
$X[i]$	引用序列X中下标为i的成员
$X[i: j]$	引用序列X中下标为i到j-1的子序列
$X[i: j: k]$	引用序列X中下标为i到j-1的子序列, 步长为k
$len(X)$	计算序列X中成员的个数
$max(X)$	序列X中的最大值
$min(X)$	序列X中的最小值
$v \text{ in } X$	检查v是否在序列X中, 返回布尔值
$v \text{ not in } X$	检查v是否不在序列X中, 返回布尔值

访问单个数据

- 用[]来访问序列中的一个元素。
比如访问字符串中的某个字符：

```
prompt = 'hello'
```

```
print(prompt[0])
```

输出：

h

```
print(prompt[4])
```

输出：

o

访问单个数据（续）

- 假设序列中的元素个数是 n ，下标的有效范围是0到 $n-1$ ，或者-1到- n 。
- 如果下标的绝对值大于 $n-1$ ，则会发生下标越界错误。
- 如果下标的值为负数，表示从序列的最后一个元素往前引用，比如：

```
prompt = 'hello'
```

```
print(prompt[-1],prompt[-4])
```

输出

```
o e
```

访问一部分数据

- 如果要访问序列中的一部分元素，可以使用切片（slice）。切片通过冒号分隔两个下标来实现。比如访问列表中的一部分：

```
a = [2,3,5,7,11,13]
```

```
print(a[1:3])
```

输出

```
[3,5]
```

0	1	2	3	4	5
2	3	5	7	11	13
-6	-5	-4	-3	-2	-1

注：如右图所示，切片a[1:3] 表示包含从第1个下标（1）开始到第2个下标（3）前面的下标（2）为止的部分元素的子序列（列表）。

访问一部分数据（续）

- 切片使用负的下标访问

`a[1:-3]`

结果: `[3, 5]`

- 切片省略第2个下标, 表示从第1个下标的元素开始到最后一个元素的切片。

`a[2:]`

结果: `[5, 7, 11, 13]`

- 第1个下标为0时, 可以省略。

`a[:3]`

结果: `[2, 3, 5]`

`a[:-2]`

结果: `[2, 3, 5, 7]`

访问一部分数据（续2）

- 切片使用第3个参数，该参数表示切片选择元素的步长。

`a[0:5:2]`

结果是: `[2, 5, 11]`

- 切片使用第3个参数为负数时，表示逆向取切片。

`a[-1:0:-1]`

结果是: `[13, 11, 7, 5, 3]`

`a[::-1]`

结果是: `[13, 11, 7, 5, 3, 2]`

复制一个序列

- 如果将一个序列变量赋值给另外一个变量，则这2个变量表达了同一个序列。

```
a = [2, 3, 5, 7, 11, 13]
```

```
b = a
```

```
b[0] = 1
```

```
print(a)
```

输出：

```
[1, 3, 5, 7, 11, 13]
```

- 如果希望2个变量各自拥有独立的序列，可使用切片。

```
a = [2, 3, 5, 7, 11, 13]
```

```
b = a[:]
```

注：a[:]表示从头到尾的整个序列“切”出来的序列

序列的运算符

- ◎ 加号 (+) 连接2个序列

```
a = [2,3,5,7,11,13]  
b = [4,6,8,9,10,12]  
print(a + b)
```

输出:

```
[2, 3, 5, 7, 11, 13, 4, 6, 8, 9, 10, 12]
```

- ◎ 乘号 (*) 重复序列

```
[4,0,4]*3
```

结果是: [4, 0, 4, 4, 0, 4, 4, 0, 4]

序列的运算符（续）

- ◎ 检查数据是否在序列中（in）

```
a = [2,3,5,7,11,13]  
print(3 in a)
```

输出结果：True

- 对于列表和字符串，in有所不同，比较下面例子：

```
[2,3] in [2,3,5,7,11,13]
```

结果是：False

```
[2,3] in [[2,3],5,7,11,13]
```

结果是：True

```
'e' in 'hello'
```

结果是：True

```
'he' in 'hello'
```

结果是：True

in可以检查某个字符串是否是另一个字符串的一部分。

计算序列的长度

- ◎ len()函数返回序列内部元素的个数。

len([2,3,5,7])

结果是： 4

len('hello world')

结果是： 11

计算序列的最小值、最大值

- ◎ min()和max()函数计算序列中的最小值和最大值

`min([2,3,5,7,11,13])`

结果是：2

`min('好好学习天天向上')`

结果是：

'上'

注：字符串的大小是按照其Unicode 编码来比较的。

3.2 字符串使用

- 字符串是一连串的字符，用英文单引号（'）或英文双引号（"）括起来。

`'Python is the best.'`

`"Programming is fun."`

- 引号必须成对出现；如果字符串中包含了单引号或双引号，则要用另一种引号括起来。

`"It's amazing!"`

`'He said, "You are so cool!"`

字符串使用（续）

- 长字符串

用3个引号（单引号或双引号）括起来的字符串可以包含多行字符串。

```
"""This is a test  
for multiple lines  
of text."""
```

表示包含了2个换行符的字符串：

```
'This is a test\nfor multiple lines\nof text.'
```

如果要在程序中用多行表示一个字符串，则可以在每行的结尾用反斜杠（\）结束。

```
'hello \  
world'
```

结果是：'hello world'

字符串使用（续2）

◎ 原始字符串

在一个字符串字面量前加一个字符r，表示这个字符串是原始字符串，其中的\不被当作是转义字符前缀。

```
r = r'hello\nworld'
```

```
print(r)
```

输出： `hello\nworld`

相当于 `r = 'hello\\nworld'`

字符串使用（续3）

◎ 字符串是不可修改

字符串中的数据（字符）是不能修改的。

```
s='hello'
```

```
s[0]='k'           #会得到错误
```

可以通过用新的字符串对变量重新赋值，表示新的字符串。

```
s='hello'
```

```
s='bye'
```

这样变量s表示字符串'bye'。

字符串常用方法或函数

表 3-2 字符串常用方法或函数

字符串常用方法或函数	解释
S.title()	字符串S首字母大写
S.lower(),	字符串S变小写
S.upper()	字符串S变大写
S.strip(),S.rstrip(),lstrip()	删除前后空格，删除右空格，删除左空格
S.find(sub[,start[,end]])	在字符串S中查找sub子串首次出现的位置
S.replace(old,new)	在字符串S中用new子串替换old子串
S.join(X)	将序列X合并成字符串
S.split(sep=None)	将字符串S拆分成列表
S.count(sub[,start[,end]])	计算sub子串在字符串S中出现的次数

字符串常用方法或函数（续）

◎ 查找子串 find()

- 在字符串中查找子串，返回第一次出现的位置下标（从0开始），如果找不到返回-1。

```
s = 'This is a test.'  
print(s.find('is'))
```

输出：

2

```
s = 'This is a test.'  
print(s.find('ok'))
```

输出：

-1

字符串常用方法或函数（续2）

```
s = 'This is a test.'  
print(s.find('is',3))    #指定查找开始位置
```

输出：

5

```
s = 'This is a test.'  
print(s.find('is',3,6)) #指定查找开始位置及终止位置
```

输出：

-1

字符串常用方法或函数（续3）

◎ 统计子串出现的次数count()

```
s = 'This is a test.'  
print(s.count('is'))
```

输出：

2

字符串常用方法或函数（续4）

◎ 修改大小写

- 函数title() 将字符串中每个单词的首字母变成大写字母。

```
name = 'john johnson'  
print(name.title())
```

输出：

John Johnson

- 函数upper() 将字符串中所有字母变成大写字母。
- 函数lower() 将字符串中所有字母变成小写字母。

字符串常用方法或函数（续5）

◎ 删除两端的空格

- 函数rstrip() 去掉字符串右边的空格。

```
name = "Python "  
name.rstrip()
```

结果是：

```
'Python'
```

- 函数lstrip() 去掉字符串左边的空格。
- 函数strip() 去掉字符串左右两边的空格。

字符串常用方法或函数（续6）

◎ 替换字符串中的子串 `replace()`

```
s = 'This is a test.'  
t = s.replace('is', 'eez')  
print(t)
```

输出：

Theez eez a test.

将数字转换成字符串

将数字类型的数据（整数、浮点数和复数）转换成字符串，有下列三种方法。

1. 函数str()

```
age = 23
```

```
print('Happy Birthday ' + str(age) + '!')
```

输出：

```
Happy Birthday 23!
```

注:不能写成: `print('Happy Birthday ' + age + '!')`

将数字转换成字符串（续）

2. 字符串格式化运算符 %

<带有转换说明符%的字符串> % (<需要转换的值>[,<需要转换的值>])

'Happy Birthday %d!' % (23)

结果是:

'Happy Birthday 23!'

格式占位符

被转换的数据

也可以有多个转换的值

'Happy Birthday %d, %s!' % (23, 'John Johnson')

结果是:

'Happy Birthday 23, John Johnson!'

将数字转换成字符串（续2）

表3-3 Python字符串的格式占位符

占位符	含义
%c	单个字符，替换成只有一个字符的字符串或将表示一个字符的Unicode码转成一个字符替换进来
%s	字符串
%d	整数
%u	无符号整数
%o	八进制数
%x	十六进制数
%X	字母大写的十六进制数
%f	浮点数
%e	科学计数法表示的浮点数
%E	大写的E表示的科学计数法
%g	综合的%f和%e，系统自动决定是否使用科学计数法
%G	大写表示的%g

将数字转换成字符串（续3）

3. format()函数

是字符串的一个函数，也是用来形成格式化的字符串。使用{}来表示占位符。

```
age = 23
```

```
'My age is {}'.format(age)
```

结果是：

```
'My age is 23'
```

格式占位符

被转换的数据

将数字转换成字符串（续4）

- format()函数支持多个占位符，可以为占位符指定的被转换数据的索引。

`'my name is {} ,age {}'.format('Mary',18)`

结果是：

`'my name is Mary ,age 18'`

`'my name is {1} ,age {0}'.format(10,'Mary')`

结果是：

`'my name is Mary ,age 10'`

将数字转换成字符串（续5）

- format()函数也可以指定填充、对齐和宽度，以及精度和进制。它的一般格式是：

{<索引>:<填充字符><对齐方式><宽度.精度><格式>}

例：

```
{0:*>10}'.format(10)  ##右对齐
'{0:*<10}'.format(10)  ##左对齐
'{0:*^10}'.format(10)  ##居中对齐
'{0:.2f}'.format(1/3)
'{0:b}'.format(10)      #二进制
'{0:o}'.format(10)      #八进制
'{0:x}'.format(10)      #16进制
'{:,}'.format(12345678901) #千分
```

```
'*****10'
'10*****'
'****10****'
'0.33'
'1010'
'12'
'a'
'12,345,678,901'
```

f-string (3.6增加,f-fast)

- ◎ `name = "Eric"`
- ◎ `age = 74`
- ◎ `print(f"Hello, {name}. You are {age}.")`
- ◎ 输出结果
- ◎ Hello, Eric. You are 74.

简单运算

- ◎ `>>> f"{2*7}"`

- ◎ `'14'`

- ◎ `>>> comedian = {'name': 'Eric Idle', 'age': 74}`

- ◎ `>>> f"The comedian is {comedian['name']}, aged {comedian['age']}."`

输入四个字符串，求这些字符串的最大长度

- ⦿ `length=0`
- ⦿ `for i in range(4):`
- ⦿ `a=len(input())`
- ⦿ `if a>length:`
- ⦿ `length=a`
- ⦿ `print(length)`