



# Python编程案例教程

# 第4章 字符串的使用

## 本章导读


本章引入了一个新的概念：数据结构。数据结构是通过某种方式（如对元素进行编号）组织在一起的数据元素的集合，这些数据元素可以是数字或者字符，甚至可以是其他数据结构。在Python中，最基本的数据结构是序列（sequence），常用的序列结构有字符串、列表和元组。本章将介绍字符串的使用，列表和元组将在第5章介绍。

本章首先介绍通用序列的操作，然后介绍字符串格式化操作和常用的字符串方法，最后通过两个典型案例的分析和实现，让读者进一步掌握字符串的使用方法。





# 学习目标

- 掌握通用序列的操作方法
  - 掌握字符串格式化操作方法
  - 掌握常用的字符串方法
  - 掌握针对字符串的程序设计方法
- 

The background of the slide features an abstract design with overlapping, semi-transparent red polygons of various shades, creating a dynamic, layered effect on the left side of the frame.

**4.1 通用序列操作**

4.2 字符串格式化操作

4.3 字符串方法

4.4 典型案例

# 4.1 通用序列操作

◆ 4.1.1 索引

◆ 4.1.2 分片

◆ 4.1.3 序列相加

◆ 4.1.4 乘法

◆ 4.1.5 长度、最小值和最大值

简单地说，序列是一块用来存放多个值的连续内存空间。

Python中常用的序列结构有字符串、列表、元组等。

所有序列类型都可以进行某些特定的操作。这些操作包括

- ◆ 索引
- ◆ 分片
- ◆ 加
- ◆ 乘
- ◆ 计算序列长度、找出最大元素和最小元素



## 4.1.1 索引

序列中的所有元素都可以通过索引（下标）来获取

从左往右，第一个元素的索引为0，第二个为1.....

索引还可以取负值，从右往左，最后一个元素的索引为-1，倒数第二个为-2.....



例如：

创建一个字符串：`str='Python'`

字符	P	y	t	h	o	n
索引（正）	0	1	2	3	4	5
索引（负）	-6	-5	-4	-3	-2	-1

假设需要获取字符串变量str中的第1个字符“P”：

`str[0]`

`str[-6]`

`'Python'[0]`



## 4.1.2 分片

分片与索引类似，都可以获取序列中的元素，区别是索引只能获取单个元素，而分片可以获取一定范围内的元素。

分片通过冒号隔开的两个索引来实现，其语法是：

`slice[start:end:step]`

step表示步长（默认为1，不能为0）

end表示结束索引（不包含结束位本身），省略时，表示到序列的最后一个元素结束

start表示起始索引，省略时，表示从序列中的第一个元素开始



python<sup>TM</sup>



## 4.1.2 分片



例如：创建序列：strs = 'abcdefg'

分片方式	描 述	结 果
strs[1:]	获取strs中从索引1开始到最后一个的所有元素	'bcdefg'
strs[:3]	获取strs中从索引0到索引3之间所有元素	'abc'
strs[1:3]	获取strs中从索引1到索引3之间所有元素	'bc'
strs[:-1]	获取strs中从索引0开始到最后一个元素之间的所有元素	'abcdef'
strs[-3:-1]	获取strs中从索引-3开始到最后一个元素之间的所有元素	'ef'
strs[-3:]	获取strs中最后三个元素	'efg'
strs[:]	获取strs中所有元素	'abcdefg'
strs[0:7:1]	获取序列strs中所有元素	'abcdefg'
strs[0:7:2] strs[::2]	将步长设置为2，获取strs从开始到结束的元素	'aceg'
strs[7:0:-1]	获取strs中从索引7到索引0之间所有元素	'gfedcb'

### 4.1.3 序列相加

可以使用加法运算符对序列进行连接操作：



例如：

```
>>> 'Hello,' + 'World'      #字符串序列连接
```

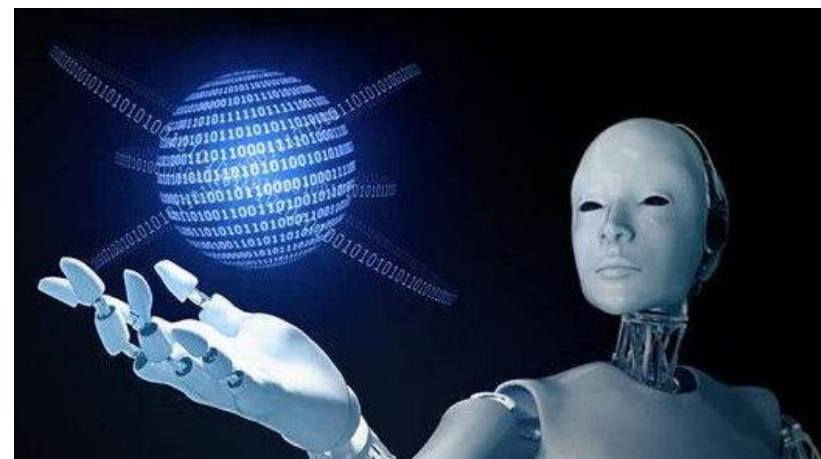
```
'Hello,World'
```

```
>>> [1,2,3] + [4,5,6]      #列表序列连接
```

```
[1, 2, 3, 4, 5, 6]
```

#### 提示

只有两种相同类型的序列才能进行连接操作，例如，字符串和列表是无法连接在一起的。



## 4.1.4 乘法

使用数字n乘以一个序列会生成新的序列，在新的序列中，原来的序列将被重复n次。



例如：

```
>>> 'hello' * 3  
'hellohellohello'
```

例：编写程序，要求利用序列的乘法运算输出如下图所示的文字格式。

```
-----  
  
I love Python!  
  
-----
```



```
strs="I love Python! "  
print(" "*10+"-"*15)      #定义字符串并赋值  
                           #利用序列的乘法运算输出  
                           10个空格和15个 "-"  
print(" "*30)              #输出30个空格  
print(" "*10+strs)         #输出10个空格和字符串  
print(" "*30)              #输出30个空格  
print(" "*10+"-"*15)       #输出10个空格和15个 "-"
```

## 4.1.5 长度、最小值和最大值

序列有3个非常好用的内建函数：len()、min()和max()函数，其参数都为序列名。

- ◆ len()函数返回序列中所包含元素的个数
- ◆ min()函数返回序列中最小的元素
- ◆ max()函数返回序列中最大的元素

例：输入一个字符串，求字符串的长度、字符串中的最大字符和最小字符。

```
strs=input("请输入一个字符串：")
str_len=len(strs)
str_max=max(strs)
str_min=min(strs)
print("该字符串长度为：",str_len)
print("该字符串中最大字符为：",str_max)
print("该字符串中最小字符为：",str_min)
```

### 提示

min()和max()函数的参数并不一定是序列，也可以是两个或两个以上的数字，例如：  
max(0,1,2,3)

### 程序运行效果



```
4-2.py x *REPL* [python] x
请输入一个字符串： abcdefg
该字符串长度为： 7
该字符串中最大字符为： g
该字符串中最小字符为： a

***Repl Closed***
Line 1, Column 17 Tab Size: 4 Pytho
```

## 4.2 字符串格式化操作

◆ 4.2.1 使用符号 “%”进行格式化

◆ 4.2.2 使用format()方法进行格式化

# 字符串格式化操作

指定 "+" 时，  
输出右对齐；  
指定 "-" 时，  
输出左对齐。

用十进制整数m来表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出；若实际位数少于定义的宽度，则补空格或0（如在m前有数字0，则补0）。

格式字符用来表示输出数据的类型。

"%[标志][0][输出最小宽度][.精度]格式字符"%变量

"%[±][0][m][.n]格式字符"%变量

如果输出数字，则表示小数的位数；  
如果输出的是字符，则表示输出字符的个数；  
若实际位数大于所定义的精度数，则截去超过的部分。

格式字符	功能说明	格式字符	功能说明
%s	字符串（采用str()显示）	%o	八进制整数
%r	字符串（采用repr()显示）	%x	十六进制整数
%c	单个字符	%f，%F	浮点数
%%	字符%	%e，%E	指数（基底为e或E）
%d，%i	十进制整数	%g，%G	以%f或%e中较短的输出宽度输出浮点数

## 4.2.1 使用符号 “%” 进行格式化



例：使用符号 “%”进行格式化。

<code>a=15</code>	<code>#定义变量a并赋值</code>
<code>b=12345678.1234567</code>	<code>#定义变量b并赋值</code>
<code>strs="I love Python!"</code>	<code>#定义字符串strs并赋值</code>
<code>print("a=%05d"%a)</code>	<code>#使用m控制输出位数，空位补0</code>
<code>print("b=%8.3f"%b)</code>	<code>#输出b，用m.n控制输出的长度和小数点位数</code>
<code>print("%17s"%strs)</code>	<code>#使用m控制输出字符串长度</code>
<code>print("%17r"%strs)</code>	<code>#使用%r输出字符串</code>
<code>print("%-17.5s"%strs)</code>	<code>#使用-m.n进行左对齐、限制字符串长度和字符位数</code>

程序运行效果

```
a=00015
b=12345678.123
  I love Python!
'I love Python!'
I lov
[Finished in 0.2s]
```

8 lines, 275 characters selected



## 4.2.2 使用format()方法进行格式化

使用format()方法也可以格式化字符串，其基本格式为：

模板字符串.format(逗号分隔的参数)

模板字符串是由一系列槽（用大括号表示）组成，用来控制字符串中嵌入值出现的位置，其基本思想是将format()方法中逗号分隔的参数按照序号替换到模板字符串的槽中（序号从0开始编号）。



例如：

"你好，{1}，你这个月的工资是{0}元！".format(8500,"张三")

如果大括号中没有序号，则按照出现顺序替换



例如：

"你好，{}，你这个月的工资是{}元！".format(8500,"张三")





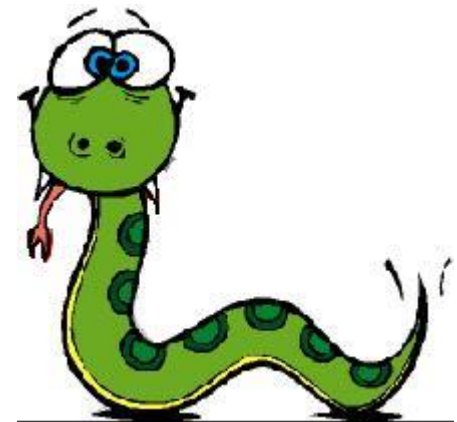
## 4.2.2 使用format()方法进行格式化

format()方法中模板字符串的槽除了包括参数序号外，还可以包括格式控制信息，此时槽的内部样式为：

{参数序号:格式控制标记}

其中格式控制标记用于控制参数显示时的格式：

填充	对齐	宽度	,	.精度	格式字符
用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	输出宽度	数字的千位分隔符	浮点数小数部分精度或字符串最大输出长度	整数类型 d,o,x,X,b,c 浮点数类型 e,E,f,%



## 4.2.2 使用format()方法进行格式化

例：使用format()方法格式化字符串。

```
a=15
```

#定义变量a并赋值

```
b=12345678.1234567
```

#定义变量b并赋值

```
strs="I love Python!"
```

#定义字符串strs并赋值

```
print("a={0:05}".format(a))
```

#输出5位数字空位补0

```
print("b={0:,.3f}".format(b))
```

#输出b，用千分位分隔并保留3位小数

```
print("{0:*^30}".format(strs))
```

#居中且使用\*填充

程序运行效果

```
a=00015
b=12,345,678.123
*****I love Python!*****
[Finished in 0.2s]
```

6 lines, 214 characters selected



The background of the slide is an abstract composition of various shades of red and pink. On the left side, there are several overlapping, semi-transparent triangular shapes that create a layered, geometric effect. The right side of the slide is a solid, light pink color. The overall design is modern and minimalist.

## 4.3 字符串方法

## 4.3 字符串方法

### ► 1. find()方法

find()方法用于在一个较长的字符串中查找子串。

- ◆ 如果找到子串，返回子串所在位置的最左端索引；
- ◆ 如果没有找到则返回-1。

```
str.find(sub[,start[,end]])
```

- ◆ str表示被查找字符串；
- ◆ sub表示查找的子串；
- ◆ start表示开始索引，省略时默认为0；
- ◆ end表示结束索引，省略时默认为字符串的长度。



## 4.3 字符串方法



例：查找子串 “like” 是否在字符串new\_str中。

```
new_str = "I like learning Python"    #创建字符串
a=new_str.find("like")                #在new_str中查找子串 “like”
b=new_str.find("like",5,15)           #在new_str的索引为5 ~ 15的字符中查找子串
print(a)                             #输出a
print(b)                             #输出b
```

程序运行效果

```
2
-1
[Finished in 0.2s]
```



Line 6, Column 1

Tab Si

提示

用于查找子串的另一个常用方法是index()方法，该方法与find()方法的用法基本一致，区别在于当查找的子串不存在时，抛出异常。

## 4.3 字符串方法

### ► 2. count()方法

count()方法用于统计字符串里某个子串出现的次数。该函数返回子串在字符串中出现的次数。

```
str.count(sub[,start[,end]])
```

- ◆ str表示被查找字符串；
- ◆ sub表示要统计的子串；
- ◆ start表示开始索引，省略时默认为0；
- ◆ end表示结束索引，省略时默认为字符串的长度。



## 4.3 字符串方法

例：创建字符串`new_str="This is a Python book!"`，使用`count()`方法找出其中“is”出现的次数。

```
new_str="This is a Python book!"    #创建字符串"This is a Python book!"
a=new_str.count('is')               #统计new_str中“is”出现的次数
b=new_str.count('is',1,6)           #设置开始和结束索引，统计“is”出现的次数
print(a)                           #输出a
print(b)                           #输出b
```

程序运行效果

```
2
1
[Finished in 0.2s]
```



Line 6, Column 1

Tab Size: 4

Python

## 4.3 字符串方法

### ▶ 3 . split()方法

split()方法以指定字符为分隔符，从字符串左端开始将其分隔成多个字符串，并返回包含分隔结果的列表。

```
str.split([delimiter,num])
```

- ◆ str表示被分隔的字符串；
- ◆ delimiter表示分隔符，省略时默认为空字符，包括空格、换行（\n）、制表符（\t）等；
- ◆ num表示分割次数，省略时默认全部分割。



## 4.3 字符串方法



例如：

创建字符串`new_str = "This is an example of cutting"`，使用`split()`进行分割。

```
>>> new_str = "This is an example of cutting" #创建字符串
>>> new_str.split()                          #以空字符为分割符将字符串全部分割
['This', 'is', 'an', 'example', 'of', 'cutting']
>>> new_str.split(' ',3)                     #以空字符为分割符将字符串分割3次
['This', 'is', 'an', 'example of cutting']
```



## 4.3 字符串方法

### ► 4 . join()方法

join()方法用于将序列中的元素以指定的字符连接，生成一个新的字符串。

```
str.join(sequence)
```

- ◆ str表示连接符，可以为空；
- ◆ sequence表示要连接的元素序列。



**例如：**创建字符串new\_str = “This is a python book!”，使用join()方法将new\_str中的字符用“-”连接。

```
>>> new_str = "This is a python book! "      #创建字符串" This is a python book!"
>>> '-'.join(new_str)                        #用 "-" 连接new_str中的字符
'T-h-i-s- -i-s- -a- -p-y-t-h-o-n- -b-o-o-k-!'
```

## 4.3 字符串方法

例：将字符串 "This is a python book!" 中的多余空格删除，即如果有连续空格只保留一个。

```
new_str = "This is a python book!"    #创建字符串
s_str=new_str.split()                 #以空字符为分割符，将new_str全部分割
print(s_str)                          #输出分割后结果
j_str=' '.join(s_str)                 #用空格连接s_str中的字符
print(j_str)                          #输出连接后的字符串
```

程序运行效果

```
['This', 'is', 'a', 'python', 'book!']
This is a python book!
[Finished in 0.2s]
```



Line 5, Column 51; Build finished

Tab Siz

## 4.3 字符串方法

### ► 5 . replace()方法

replace()方法用于将字符串中的旧字符串替换成新字符串。

```
str.replace(old,new[,max])
```

- ◆ str表示被查找字符串；
- ◆ old表示将被替换的子串；
- ◆ new表示新字符串，用于替换old子串；
- ◆ max是可选参数，表示替换不超过max次，省略时默认替换所有。



**例如：**

创建字符串 "new\_str = "Monday Tuesday Wednesday Thursday Friday Saturday Sunday""，使用replace()方法替换其中的day为DAY。

```
>>> new_str = "Monday Tuesday Wednesday Thursday Friday Saturday Sunday"
>>> new_str.replace('day','DAY')           #将new_str中的day替换为DAY
'MonDAY TuesDAY WednesDAY ThursDAY FriDAY SaturDAY SunDAY'
>>> new_str.replace('day','DAY',3)         #将new_str中的day替换为DAY，限制为3次
'MonDAY TuesDAY WednesDAY Thursday Friday Saturday Sunday'
```



## 4.3 字符串方法

### ► 7. lower()和upper()方法

- ◆ lower()方法用于转换字符串中所有大写字符为小写；
- ◆ upper()方法则用于转换字符串中所有小写字符为大写。

```
str.lower()
```

```
str.upper()
```



**例如：** 创建字符串 “new\_str=’This is an EXAMPLE’”，使用lower()方法将其转换为小写，再使用upper()方法将其转换为大写。

```
>>> new_str = "This is an EXAMPLE" #创建字符串
>>> new_str.lower()                #将new_str中的大写字符转为小写
'this is an example'
>>> new_str.upper()                #将new_str中的小写字符转为大写
'THIS IS AN EXAMPLE'
```

## 4.3 字符串方法

### ► 8 . isalnum()方法

isalnum()方法用于检测字符串是否由字母和数字，或两种的一种组成。

- ◆ 如果是就返回True；
- ◆ 否则返回False。

```
str.isalnum()
```



**例如：**创建字符串new\_str = “2018example”，使用isalnum()方法判断new\_str中是否由字母或数字组成。

```
>>> new_str = "2018example"      #创建字符串
>>> new_str.isalnum()            #判断new_str中是否只有数字或字母
True
```

The background of the slide is an abstract composition of various shades of red and pink. It features overlapping, semi-transparent geometric shapes, primarily triangles and polygons, which create a sense of depth and movement. The colors range from deep, dark reds to lighter, almost white pinks, with the most intense colors concentrated on the left side of the frame.

## 4.4 典型案例



```
import time
scale=10
print("-----执行开始-----")
for i in range(scale+1):
    a="**"*i
    b=".."(scale-i)
    c=(i/scale)*100
    print("{:^3.0f}%[{}->{}]".format(c,a,b))
    time.sleep(0.1)
print("-----执行结束-----")
```

```
#导入time模块
#变量scale用于表示输出进度条的精度
#输出
#循环变量从0到10
#用 "*" 表示已完成的部分
#用 "." 表示未完成的部分
#计算完成百分比并赋值给c
#格式化输出
#暂停0.1秒
#输出
```

## 程序运行效果

```
-----执行开始-----
% 0 [->.....]
%10 [**->.....]
%20 [****->.....]
%30 [*****->.....]
%40 [*****->.....]
%50 [*****->.....]
%60 [*****->.....]
%70 [*****->.....]
%80 [*****->.....]
%90 [*****->..]
%100[*****->]
-----执行结束-----
[Finished in 1.3s]
```

```
user_name = input("请输入用户名（以“_”开头，3-30个字符）：")
password = input("请输入密码（下划线开头、后面由数字和字母共同组成，8-16个字符）：")
if user_name[0] != '_':
    print("用户名请使用下划线开头")
elif 3 > len(user_name) or 30 < len(user_name):
    print("用户名长度超出限制")
elif 8 > len(password) or 16 < len(password):
    print("密码长度超出限制")
elif password.find('_') == -1:
    print("密码中未输入下划线")
else:
    psswords = password.replace('_', '1')
    if psswords.isalnum():
        print("恭喜您，注册成功！用户名：", user_name, "，密码：", password)
    else:
        # passwords中有数字或字母以外的字符
        print("密码中有其他符号，注册失败！") # 输出 "密码中有其他符号..."
```

程序运行效果

```
#如果user_name的首字符不是 "_"
#输出 "用户名请使用下划线开头"
#如果user_name长度小于3或大于30
#输出 "用户名长度超出限制"
#如果password长度小于8或大于16
#输出 "密码长度超出限制"
#如果password中不存在 "_"
#输出 "密码中未输入下划线"
#以上条件都不满足
#将password中的下划线替换为1
#passwords中是否只有数字或字母
```

**感谢您的观看**



```
def MyString_isContain(SrcStr,SubStr):  
#字符串出现的索引列表  
def MyString_subStrPosition(SrcStr,SubStr)  
#字符串出现的频率字典  
def MyString_subStrListCount(SrcStr,SubStrList)  
def MyString_firstIndexOf(SrcStr,SubStr):  
def MyString_lastIndexOf(SrcStr,SubStr):  
def MyString_ReplaceFirst(SrcStr,OldStr,NewStr):  
    return SrcStr  
#字符串替换,换所有  
def MyString_ReplaceAll(SrcStr,OldStr,NewStr):  
    return SrcStr
```

## #字符串截断为列表

```
def MyString_Split2List(SrcStr,SplitStr,isTrimEmpty,isTrimSame):
```

```
#把字符串切割为键值对字典：a=1;b=2;c=3  转为arr["a"]=1 arr["b"]=2  
arr["c"]=3
```

```
def MyString_Split2Dict(SrcStr,SplitStr,isTrimEmpty):
```

```
#键值列表转字符串：key1(value1);key2(value2);key3(value3)
```

```
def MyString_List2Str(List,SplitStr,isTrimEmpty=False):
```

```
#键值字典转字符串：key1=value1;key2=value2;key3=value3
```

```
def MyString_Dict2Str1(Dict,SplitStr):
```

```
#键值字典转字符串：key1(value1);key2(value2);key3(value3)
```

```
def MyString_Dict2Str2(Dict,SplitStr):
```

```
#键值字典转Json字符串：{key1:"value1";key2:"value2";key3:"value3"}
```

```
def MyString_Dict2JsonStr(Dict,SplitStr):
```

```
def MyString_subLeftString(SrcStr,SplitStr,isIncludeSplitStr=False):
```

```
def MyString_subRightString(SrcStr,SplitStr,isIncludeSplitStr=False):
```

```
def MyString_subMiddleString(SrcStr,SplitStr1,SplitStr2):
```

```
def MyString_substring1(SrcStr,startIndex):
```

```
def MyString_substring2(SrcStr,startIndex,endIndex):
```

```
def MyString_substring3(SrcStr,startIndex,subStrLength):
```

```
def MyString_AllStrip1(SrcStr,stripStr):  
def MyString_AllStrip2(SrcStr,strList):  
def MyString_Strip(SrcStr,stripStr):  
def MyString_LStrip(SrcStr,stripStr):  
def MyString_RStrip(SrcStr,stripStr):  
def MyString_ToUpper(SrcStr):  
def MyString_ToLower(SrcStr):  
def MyString_Length(SrcStr,isTrim):  
def MyString_randomString(len) :  
def MyString_ConvertPinyin(inStr) :  
def MyString_Reverse(inStr)  
def MyString_isDecimal(inStr)  
def MyString_isDigit(inStr)  
def MyString_isAlphi(inStr)  
def MyString_isAlphiAndDigit(inStr)  
def MyString_isAlphiOrDigit(inStr)
```