第三章 字符串

3.1 字符串的类型

3.2 字符串的基本方法

3.3 字符串的格式化

3.4 字符串的其他重要方法

字符串

- **字符串**是字符的序列，可以包含文本和数字；在Python中，一个包含单个字符的字符串被视为一个字符
- 字符串的值可以包含在一对单引号（’）或双引号（“”）中

```
>>> letter = 'A'  # 与 letter = "A" 相同
>>> numChar = '4'  # 与 numChar = "4" 相同
>>> message = "Good Morning!"  # 与 message = 'Good Morning!' 相同
>>> print(letter)
A
>>> print(numChar)
4
>>> print(message)
Good Morning!
```

字符串

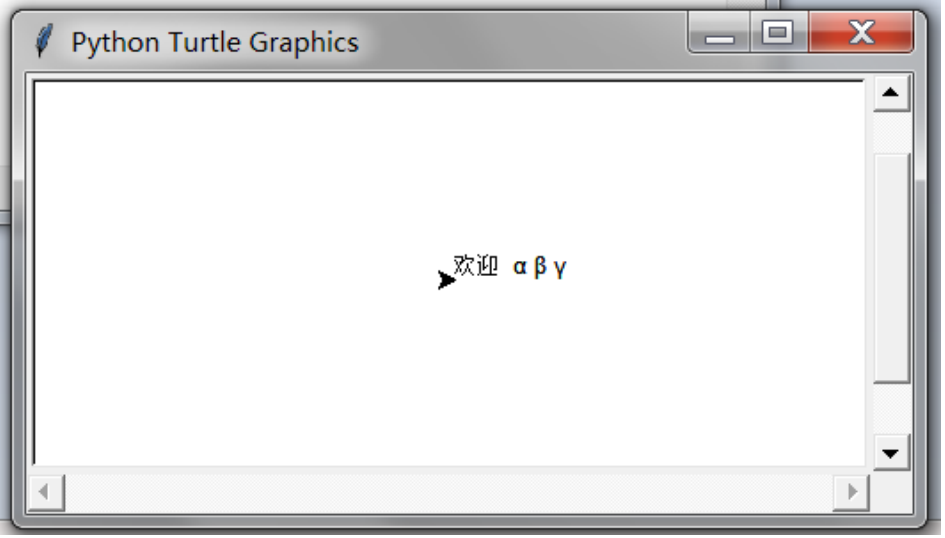
- Python提供ord函数用于返回一个给定字符串的ASCII码，同时提供chr函数用于返回对应于给定ASCII码的字符

```
>>> ch = 'a'
>>> ord(ch)
97
>>> chr(103)
'g'
>>> ord('a') - ord('A') # 任何小写字母和对应的大写字母的ASCII码差值相同
32
```

字符串

- ASCII码是Unicode的子集，Python支持Unicode，Unicode是以\u开始，后接4个16进制数，取值范围为\u0000到\uFFFF

```
>>> # 导入Python的模块turtle,该模块提供用于绘图的turtle图形原语
>>> import turtle
>>>
>>> # 在Python GUI中显示各国语言中的字符
>>> turtle.write("\u6B22\u8FCE  \u03B1 \u03B2 \u03B3")
>>>
```



字符串的类型

- Python3中表示字符序列的类型主要有两种：bytes（表示二进制数据）和str（表示文本），前者的实例包含原始的8位数值，而后者包含Unicode字符的序列
- Str可以利用某种编码方式转换为bytes，bytes也可以通过解码还原为字符串

```
>>> '€20'.encode('utf-8')  # 使用utf-8编码方式
b'\xe2\x82\xac20'
>>> b'\xe2\x82\xac20'.decode('utf-8')  # utf-8方式编码的bytes解码为字符串
'€20'
>>>
>>> '€20'.encode('iso-8859-15') # 使用另外一种编码方式
b'\xa420'
>>> b'\xa420'.decode('iso-8859-15')  # 解码为字符串
'€20'
```

用于特殊字符的转义字符

- 如果一个字符串包含了单引号，在打印该字符串时就不能再使用单引号将其括起来

```
>>> # 可以使用双引号
>>> str = "Let's go home!"
>>> print(str)
Let's go home!
>>> # 还可以使用转义字符(\),即反斜线,对字符串中的引号进行转义
>>> str = 'Let\'s go home!'
>>> print(str)
Let's go home!
```

- 同理，如果一个字符串包含了双引号，在打印该字符串时也不能直接使用双引号将其括起来，此时可以使用单引号或者转义字符\n

```
>>> str = '"Hello, world!" she said.'
>>> print(str)
"Hello, world!" she said.
>>> str = "\"Hello, world!\" she said."
>>> print(str)
"Hello, world!" she said.
```

长字符串

- 如果需要写一个长的字符串，需要跨多行，那么可以使用三个单/双引号代替普通引号，此时在字符串中可以同时使用单引号和双引号

```
>>> print('''This is a very long string.
It continues here.
And it's not over yet.
"Hello, world!"
Still here.''')
This is a very long string.
It continues here.
And it's not over yet.
"Hello, world!"
Still here.
>>> print("""This is a very long string.
It continues here.
And it's not over yet.
"Hello, world!"
Still here.""")
This is a very long string.
It continues here.
And it's not over yet.
"Hello, world!"
Still here.
```

长字符串

- 普通字符串也可以跨行，此时在一行的最后一个字符设为反斜线（\）

```
>>> str = "Hello, \
world!"
>>> print(str)
Hello, world!
>>>
>>> # 使用反斜线将换行符转义的用法还适用于表达式和语句
>>> 1 + 2 + 3 \
  + 4 + 5
15
>>> print("Hello, \
world!")
Hello, world!
```

原始字符串

- **原始字符串**以r开头，**不会把反斜线当作特殊字符**，使用print打印字符串时，在**原始字符串中**输入的每个字符都会与其书写的方式保持一致

```
>>> path = "C:\nowhere"
>>> path      # 直接显示指向字符串的引用
'C:\nowhere'
>>> print(path)  # 打印该字符串
C:
owhere
>>> path = r"C:\nowhere"
>>> print(path)
C:\nowhere
>>>
>>> print(r"Let\'s go!")
Let\'s go!
>>> print(r"Let's go!")
Let's go!
```

- 原始字符串中紧跟在反斜线后的字符会不经改变留在字符串中，且所有的反斜线也将留在字符串中

原始字符串

- 字符串中的**引号**可以用反斜线来进行转义，但是反斜线会保留在字符串中

```
>>> print(r"\"")  # 原始字符串由两个字符组成,反斜线和双引号
\"
>>> print(r"\")  # 此处为非法的原始字符串,此时反斜线将说引号转义,但是字符串尚缺结束引号

SyntaxError: EOL while scanning string literal
```

- 注意**不要在原始字符串的结尾输入单个反斜线**，此时Python无法判断是否应该结束该字符串

```
>>> print("C:\program files\python3.5\docs\")

SyntaxError: EOL while scanning string literal
>>> # 原始字符串不能以反斜线作为最后一个字符,因为此时反斜线(\)将对其后的引号进行转义
>>> # Python无法判断是否可以结束该字符串
>>>
>>> print(r"C:\program files\python3.5\docs"+"\\") # 打印最后一个字符是反斜线的字符串的方法
C:\program files\python3.5\docs\
>>>
```

第三章 字符串

3.1 字符串的类型

3.2 字符串的基本方法

3.3 字符串的格式化

3.4 字符串的其他重要方法

索引与分片

- 与其它序列一样，字符串支持索引与分片操作，返回结果放在一个新的对象中

```
>>> str = "Good Morning!"
>>> str[1]    # 获取字符串从左起第二个字符
'o'
>>> str[-2]  # 获取字符串从右起第二个字符
'g'
>>> str[len(str) - 2]  # 获取字符串从右起第二个字符的另外一种方法
'g'
>>> str[:]  # 获取整个字符串
'Good Morning!'
```

字符串连接

- 字符串支持使用加号（+）的连接（concatenation）操作，从而将两个字符串合并生成一个新的字符串（字符串是不可变数据对象）

```
>>> str + " Mr. Smith."  # 连接操作
'Good Morning! Mr. Smith.'
>>> str                  # 此时原字符串并没有改变,连接操作会生成新的字符串
'Good Morning!'
```

- 注意加号（+）应用于不同对象时其意义不同：如果操作数是数字，则加号代表数字数值相加；如果操作数是字符串，则加号代表字符串进行连接操作。这种特性称之为多态性（polymorphism）

成员资格判断

- 字符串支持成员资格判断，这一点与其它序列是一样的。为了检查一个字符是否在字符串中，可以使用in运算符；反之可以使用not in

```
>>> if "apples" in "Today we have bought lots of apples.": \
    print("We may have apples tomorrow morning.")
else:
        print("We will not have any apples tomorrow morning.")


We may have apples tomorrow morning.
```

第三章 字符串

3.1 字符串的类型

3.2 字符串的基本方法

3.3 字符串的格式化

3.4 字符串的其他重要方法

格式化操作符%

- 字符串格式化使用**字符串格式化操作符%**来实现
- 在%左侧放置格式化字符串（format string），右侧放置希望被格式化的值（可以使用一个值，或者使用包含多个值的元组或者字典）

```
>>> format ="Hello, %s. %s enough for you?"  # 格式化字符串
>>> values = ('world', 'Hot')     # 希望被格式化的值
>>> print(format % values)
Hello, world. Hot enough for you?
>>>
>>> values = ['Hello', 'Hot']  # 如果使用列表或其他序列代替元组来提供希望被格式化的值，则序列被解释成为一个值
>>> print(format % values)
Traceback (most recent call last):
  File "<pyshell#299>", line 1, in <module>
    print(format % values)
TypeError: not enough arguments for format string
```

格式化操作符%

- 格式化字符串中的%s部分称为转换说明符（conversion specifier），用来标记需要插入的转换值的位置，此时转换值会被格式化成字符串

```
>>> format = "%s are 100%% sure that %s will come back."
>>> # 如果格式化字符串本身包含%，那么必须使用%%
>>> values = ('We', 'he')
>>> print(format % values)
We are 100% sure that he will come back.
```

格式化操作符%

- 如果需要对实数进行格式化，可以使用f说明转换值的类型，同时在一个句点后可以加上希望保留的小数位数，即精度，精度放在表示类型的字符前面

```
>>> format = "Pi with three decimals: %.3f"
>>> from math import pi
>>> print(format % pi)
Pi with three decimals: 3.142
```

基本的转换说明符

- 转换类型：例如，d和i均可以表示带符号的十进制整数，o代表不带符号的八进制数，u表示不带符号的十进制数，x代表不带符号的十六进制数，e代表科学计数法表示的浮点数（小写），f代表十进制浮点数，s代表字符串，等等

```
>>> "Price of eggs : $%d" % 42   # 简单转换
'Price of eggs : $42'
>>> from math import pi
>>> "Pi: %f..." % pi
'Pi: 3.141593...'
>>>
>>> "%10f" % pi     # 指明转换后的字符串至少应该具有的宽度
'  3.141593'
>>> "%10.2f" % pi   # 指明转换后的字符串的宽度和精度
'      3.14'
>>> "%.*s" % (5, "Guido van Rossum") # 使用*作为字段宽度，数值从元组参数中读取
'Guido'
>>> "%.5s" % ("Guido van Rossum")
'Guido'
```

符号、对齐和用0填充

- 在字段宽度和精度之前还可以放置一个标志，该标志可以是0、+、-或者“ ”（空格）；其中0表示如果转换值长度不够，则用0来填充，此时的0与表示八进制数的0意义不同

```
>>> from math import pi
>>> "%010.2f" % pi       # 第一个0表示如果转换值的长度不满10位,则用0来填充; 后面紧跟的10表示转换后的数字至少具有该宽度,小数点后的2表示精度,f表示转换值为实数
'0000003.14'
>>>
>>> print(0o010)   # 此处打印的是八进制数010,括号中第一个字符是数字0,第二个字符是字母o
8
>>> print(0x110)   # 此处打印的是十六进制数110,括号中第一个字符是数字0,第二个字符是字母x
272
```

符号、对齐和用0填充

- 标志为减号（-）时，用来将转换值进行左对齐；标志为空格（“ ”）时意味着需要在**正数前**加上空格；而当标志为加号（+）时，表示需要在转换值前加上正、负号

```
>>> "%-10.2f" % pi  # 减号表示将转换值左对齐
'3.14
>>>
>>> print(("% 5d" % 10) + "\n" + ("% 5d" % -10)) # 空格表示正数前加空格
   10
  -10
>>>
>>> print(("%+5d" % 10) + "\n" + ("%+5d" % -10)) # 加号表示数字前加正负号
  +10
  -10
>>>
```

```
# 使用给定的宽度打印格式化后的价格列表
# 打印出来的效果如下

#    Please enter width: xx
#    ==================================
#    Item                           Price
#    ----------------------------------
#    Apples                          0.40
#    Pears                           0.50
#    Cantaloupes                     1.92
#    Dried Apricots (16 oz.)         8.00
#    Prunes (4 lbs.)                12.00
#    ==================================

width = input('Please enter width: ')
print('='* int(width))

price_width = 10
item_width = int(width) - price_width

header_format = '%-*s%*s'
header_content = (item_width, 'Item', price_width, 'Price')
print(header_format % header_content)

print('-' * int(width))

print('%-*s%*.2f' % (item_width, 'Apples', price_width, 0.4))
print('%-*s%*.2f' % (item_width, 'Pears', price_width, 0.5))
print('%-*s%*.2f' % (item_width, 'Cantaloupes', price_width, 1.92))
print('%-*s%*.2f' % (item_width, 'Dried Apricots (16 oz.)', price_width, 8))
print('%-*s%*.2f' % (item_width, 'Prunes (4 lbs.)', price_width, 12))

print('='* int(width))
```

字符串模板

- string模块提供另外一种格式化值的方法：字符串模板，其工作方式类似于Unix Shell里的变量替换

```
>>> from string import Template
>>> s = Template('$x, glorious $x!')
>>> s.substitute(x = 'slurm')  # substitute方法用'slurm'替换上面的$x
'slurm, glorious slurm!'
>>>
>>> s = Template("It's ${x}tastic!")  # 如果被替换的是某个词语的一部分,则该参数名必须用括号括起来
>>> s.substitute(x = 'slurm')
"It's slurmtastic!"
>>>
>>> s = Template("Make $$ selling!")  # 使用美元符号$时必须用$$来插入
>>> s.substitute(x = 'slurm')
'Make $ selling!'
>>>
```

字符串模板

- 除了像前面的例子一样使用关键字参数（即类似x='slurm'的样子），我们还可以使用字典变量提供值/名字对

- 字典中的值均存在特定的键下，例如items={'name':'Gumby', 'age':42}，此处'name'和'age'均为键值

```
>>> s = Template('A $thing must never $action.')
>>> d = {}      # 创建一个空字典
>>> d['thing'] = 'gentleman'   # 为第一个参数thing设置替换值
>>> d['action'] = 'show his socks'   # 为第二个参数action设置替换值
>>> s.substitute(d)     # 使用字典类型的对象作为substitute方法的参数
'A gentleman must never show his socks.'
```

第三章 字符串

3.1 字符串的类型

3.2 字符串的基本方法

3.3 字符串的格式化

3.4 字符串的其他重要方法

查找字符串

- find方法在一个较长的字符串中查找子串，并**返回子串所在位置的最左端索引**，如果没有找到，则返回-1

```
>>> "With a moo-moo here, and a moo-moo there".find("moo")
7
>>> title = "Monty Python's Flying Circus"
>>> title.find("Flying")
15
>>> title.find("Dircuss")
-1
>>> "$$$ Get rich now!!! $$".find("$$") # 查找子串"$$"的最左端索引
0
>>> "$$" in "$$$ Get rich now!!! $$"
True
>>> # 操作符in返回一个字符串是否是另外一个字符串的字串,不返回位置索引
```

查找字符串

- find方法还可以接收可选的起始点和结束点参数，注意，由起始点和结束点制定的范围包含第1个索引，但是不包含第2个索引

```
>>> subject = "$$$ Get rich now!!! $$$"
>>> subject.find("rich")  # 在整个字符串中寻找子串的最左端索引
8
>>> subject.find("rich", 9)  # 从字符串subject的第9个位置作为起始点进行子串的查找
-1
>>> subject.find("rich", 4, len(subject)) # 从字符串subject的第4个位置作为起始点开始直至字符串末尾进行子串的查找
8
```

将字符串分割成序列

- split方法用来将字符串分割成序列（元素均为字符串），如果不提供用于分割的分隔符，Python会将所有空格作为分隔符

```
>>> '1+2+3+4+5'.split('+')
['1', '2', '3', '4', '5']
>>>
>>> '/usr/bin/env'.split('/')
['', 'usr', 'bin', 'env']
>>>
>>> 'using the default'.split()  # 当没有提供分隔符时,Python会使用空格作为分隔符
['using', 'the', 'default']
```

连接序列中的元素

- join方法用来连接序列（列表或元组均可）中的**字符串**，该方法是split方法的逆方法

```
>>> seq = [1, 2, 3, 4, 5]
>>> sep = '+'    # 使用加号(+)作为分隔符
>>> sep.join(seq)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    sep.join(seq)
TypeError: sequence item 0: expected str instance, int found
>>>
>>> seq = ['1', '2', '3', '4', '5']  # join方法连接的是字符串
>>> sep.join(seq)
'1+2+3+4+5'
>>>
>>> dirs = '', 'usr', 'bin', 'env'
>>> '/'.join(dirs)     # 利用分隔符/来连接一个元组中的各个字符串
'/usr/bin/env'
>>>
```

返回字符串的小写字母形式

- lower方法返回字符串的小写字母形式，如果想要编写不区分大小写的代码的话，此方法特别有用

```
>>> 'Trondheim Hammer Dance.'.lower()
'trondheim hammer dance.'
>>>
>>> if 'Gumby' in ['gumby', 'smith', 'jones']: print('Found it!')
else: print('Not exist!')

Not exist!
>>> # 事实上此时无需要求区分大小写,特别是要求用户输入时,可以考虑将输入的名字取小写字母形式,然后进行判断
>>> if 'Gumby'.lower() in ['gumby', 'smith', 'jones']: print('Found it!')
else: print('Not exist!')

Found it!
```

返回字符串的小写字母形式

- 与lower方法相关的是title方法，该方法将字符串转换成标题，即所有单词的首字母大写，而其他字母小写。title方法得到的结果可能不太自然，有时会考虑使用string模块中的capwords函数

```
>>> "that's all, folks.".title()
"That'S All, Folks."
>>>
>>> import string
>>> string.capwords("that's all, folks.")
"That's All, Folks."
>>>
```

去除字符串的两侧空格

- strip方法返回去除两侧（不包括内部）空格（或其它字符）的字符串

```
>>> "        internal whitespace is kept  ".strip()
'internal whitespace is kept'
>>> name = 'Gumby   '  # 假设用户输入的名字后无意之中加上了不必要的空格
>>> if name.strip().lower() in ['gumby', 'smith', 'jones']: print('Found it!')
else: print('Not exist!')

Found it!
>>> '*** SPAM * for * everyone!!!     ***'.strip(' *!') # 制定需要去除的字符,将它
们列为strip方法的参数即可
'SPAM * for * everyone'
>>> # 以上例子显示了在原字符串两侧去除空格、*和!
```

```
>>> '   blank space'.lstrip()  # 删除左侧空格
'blank space'
>>> 'blank splace  '.rstrip()  # 删除右侧空格
'blank splace'
>>> '   blank space   '.lstrip().rstrip()  # 删除两侧空格
'blank space'
```

替换字符串中所有匹配项

- replace方法返回某字符串中所有匹配项被替换之后的字符串，类似于文字处理程序中的“查找与替换”功能

```
>>> 'This is a test for the replace function.'.replace('is', 'eez')
'Theez eez a test for the replace function.'
>>>
>>> 'This is a test for the replace function.'.replace('is', 'eez', 1)
'Theez is a test for the replace function.'
>>> # 上面replace方法中出现的第3个参数count（此时count=1）表示，只将原字符串中前count个匹配项进行替换
>>>
>>> 'To be or not to be will always be a question.'.replace('be', 'play', 2)
'To play or not to play will always be a question.'
```

替换字符串中单个字符

- 与replace方法相比，translate方法**只处理单个字符**而非字符串，replace方法的优势在于**可以同时进行多个字符的替换**

- 假设我们需要将英文文本转换为带有德国口音的版本，也就是说需要将字符c替换为k，字符s替换成z

```
>>> table = str.maketrans('cs', 'kz')
>>> s = 'This is an incredible test'.translate(table)
>>> s
'Thiz iz an inkredible tezt'
```

- 以上str.maketrans方法中两个参数为字符串，二者等长，表示第一个字符串中每个字符都用第二个字符串中相同位置上的字符替换

小结

- 字符串的值可以包含在一对单引号、双引号或三引号中
- ord函数返回给定字符串的ASCII码，而chr函数返回对应于给定ASCII码的字符
- 原始字符串不会把反斜线当作特殊字符，使用print打印时，原始字符串中输入的每个字符都会与其书写的方式保持一致
- 原始字符串不能以单个反斜线作为结尾
- 所有标准的序列操作（索引、分片、序列重复、判断成员资格、求长度、最大和最小值）对字符串均适用

小结

- 字符串格式化操作符%可以用来实现字符串格式化，左侧为格式化字符串，右侧为希望被格式化的值
- 可以对值进行各种方式的格式化，包括左对齐、设定字段宽度或精度、左边填充数字0等等
- 字符串有很多种方法，包括find、split、join、lower、strip、replace和translate

本章节涉及到的函数

chr(n)	返回对应于给定ASCII码的字符
ord(c)	返回一个给定字符串的ASCII码
string.capwords(s[,sep])	使用split方法将待处理字符串分成若干词，将其首字母大写，然后使用join将各词合成一个字符串
str.maketrans(from, to)	静态方法，创建用于替换的转换表

利用输入的信用卡余额信息打印格式化后的帐户余额列表，
打印出来的效果如下:

Account Summary

No.	Credit Card	Balance
4895 1234 5678 1234	visa infinity card	-1326.40
5442 1234 5678 1234	cash back mastercard	+529.50
5309 1234 5678 1234	cash back visa card	+3947.12

要求:

- 表的标题居中,列标题中卡号(No.)和信用卡名称(Credit Card)左侧对齐，列标题中余额右侧对齐，数据和对应的列标题的排列要求一致。
- 如果账户余额为负数,表明有剩余金额可用,若账户余额为正数,则表明有欠款,该数据要求加上必要的正负号,且保留到小数点后2位。
- 程序仅接收余额信息作为输入，输入该信息时如果不小心在其中添加了不必要的空格，例如“ – 132 6.4 ”，程序要能够处理这样的情况并如上所示显示余额信息。