程序设计

基本数据类型-字符串

字符串切片

字符串中的一段称为一个切片（slice）

```
>>> greet[0:3]
'Hel'
>>>
```

<String>[<start: end: step>]

字符串切片练习

```
name="Python语言程序设计课程"
print(name[0],name[2:-2],name[-1])
```

字符串使用实例

输入一个月份数字，返回对应月份名称缩写，这个问题的IPO模式是:

输入：输入一个表示月份的数字(1-12)

处理：利用字符串基本操作实现该功能

输出：输入数字对应月份名称的缩写

字符串使用实例

	月份	字符串中位置
Jan	1	0
Feb	2	3
Mar	3	6
Apr	4	9

字符串使用实例

```
# month.py
months="JanFebMarAprMayJunJulAugSepOctNovDec"
n=input("请输入月份数(1-12):")
pos=(int(n)-1) * 3
monthAbbrev=months[pos:pos+3]
print("月份简写是"+monthAbbrev+".")
```

```
>>>
请输入月份数(1-12):7
月份简写是Jul.
>>>
```

修改字符串大小写的方法

```
name = "ada lovelace"
print(name.title())
```

title()以首字母大写的方式显示每个单词，即将每个单词的首字母都改为大写。

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

字符串in操作

in：成员运算符 - 如果字符串中包含给定的字符返回 True

not in：成员运算符 - 如果字符串中不包含给定的字符返回 True

```
'x' in 'ssdx'
'x' not in 'ssdx'
```

字符串格式化输出

print()

format()

print格式化输出

```
#打印字符串与整数
print ("我叫 %s 今年 %d 岁!" % ('小明', 10))
#打印字符串
print("My name is %s" %("Alfred.Xue"))
#打印整数
print("I am %d years old." %(25))
#打印浮点数
print ("His height is %f m"%(1.70))
#打印浮点数（指定保留两位小数）
print ("His height is %.2f m"%(1.70))
```

print格式化输出

格式化符号:

符号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度

print格式化输出

格式化操作符辅助指令:

符号	描述
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)

print格式化输出

格式化操作符辅助指令:

```
#字段宽10，精度3
pi = 3.141592653
print('%10.3f' % pi)
#用*从后面的元组中读取字段宽度或精度
print("pi = %.*f" % (3,pi))
```

print格式化输出

print 会自动在行末加上回车, 想不换行需写成 print(i, end = '')

```
for i in range(0,6):
    print (i,end='')
```

format(方法)格式化输出

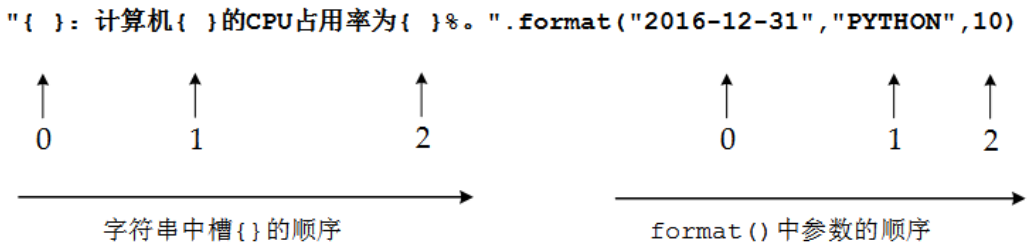
字符串format()方法的基本使用格式是:

<模板字符串>.format(<逗号分隔的参数>)

其中，模板字符串由一系列槽组成，用来控制修改字符串中嵌入值出现的位置，槽用大括号{}表示

format(方法)格式化输出

默认顺序



format(方法)格式化输出

```
# 不设置指定位置，按默认顺序
"{} {}".format("hello", "world")
# 设置指定位置
"{0} {1}".format("hello", "world")
# 设置指定位置
"{1} {0} {1}".format("hello", "world")
# 输出大括号
"{{{1} {0} {1}}}".format("hello", "world")

#设置参数
print("网站名：{name}, 地址 {url}".format(name="菜鸟教程", url="www.runoob.com"))
```

format(方法)格式化控制

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

{<参数序号>: <格式控制标记>}

format(方法)格式化控制

格式控制标记：

用来控制参数显示时的格式，包括：<填充><对齐><宽度>,<.精度><类型>6个字段，这些字段都是可选的，也可以组合使用。

:	<填充>	<对齐>	<宽度>	,	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b, c, d, o, x, X 浮点数类型 e, E, f, %

format(方法)格式化控制

宽度：当前槽设定输出字符串的宽度。若对应format中的参数比宽度大，则使用参数实际长度。否则，默认以空格填充。

```
#第0个参数，宽度30，默认左对齐，默认空格填充
s='Python'
"{0:30}".format(s)
```

format(方法)格式化控制

对齐：参数在宽度内输出时的对齐方式，分别使用<，>和^来表示左对齐，右对齐和居中对齐。

```
#第0个参数，宽度30，右对齐，默认空格填充
s='Python'
"{0:>30}".format(s)
```

format(方法)格式化控制

填充：宽度内除了参数外的字符采用什么方式表示。

```
#第0个参数，宽度30，右对齐，*号填充
s='Python'
"{0:*>30}".format(s)
```

format(方法)格式化控制

练习：

```
s='Python'
"{0:3}".format(s)
"{0:-^30}".format(s)
```

format(方法)格式化控制

精度：<.精度>，小数点开头，两个含义：浮点数的精度表示小数部分的有效输出位数；字符串的精度表示输出的最大长度。

```
#对比浮点数和字符串
a=3.1415
"{0:.2f}".format(a)
s='Python'
"{0:.2}".format(s)
```

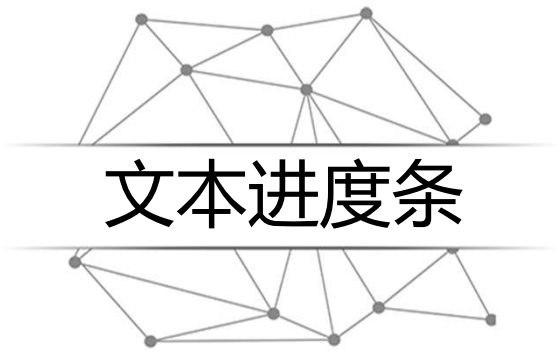
format(方法)格式化控制

类型：表示输出整数和浮点数的格式规则，如下表所示：

类型	符号	含义
整数	b	输出整数的二进制方式
整数	c	输出整数对应的Unicode字符
整数	d	输出整数的十进制方式
整数	o	输出整数的八进制方式
整数	x	输出整数的小写十六进制方式
整数	X	输出整数的大写十六进制方式
浮点数	e	输出浮点数对应的小写字母e的指数形式
浮点数	E	输出浮点数对应的大写字母E的指数形式
浮点数	f	输出浮点数的标准浮点形式
浮点数	%	输出浮点数的百分形式

format(方法)格式化控制练习

```
"{:.2f}".format(3.14159)
"{:+.2f}".format(3.14159)
"{:+.2f}".format(-1)
"{:x<4d}".format(5)
"{:,}".format(100000)
"{:.2%}".format(25)
```



文本进度条

文本进度条

利用print()函数实现简单的非刷新文本进度条，基本思想：

按照任务执行百分比将整个任务划分为100个单位，每执行N%输出一次进度条。每一行输出包含进度百分比，代表已完成的部分(**)和未完成的部分(..)的两种字符，以及一个跟随完成度前进的小箭头，风格如下：

```
%10[*****>.....................................]
```

文本进度条

time库

This module provides various time-related functions.

time.**sleep**(*secs*)

Suspend execution of the calling thread for the given number of seconds. The argument may be a floating point number to indicate a more precise sleep time. The actual suspension time may be less than that requested because any caught signal will terminate the `sleep()` following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount because of the scheduling of other activity in the system.

文本进度条

```
#e4.1TextProgressBar.py
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------")
```

单行动态刷新

```
#e4.2TextProgressBar.py
import time
for i in range(101):
    print("\r{:2}%".format(i), end="")
    time.sleep(0.05)
```

单行动态刷新

```
#e4.3TextProgressBar.py
import time
scale = 50
print("执行开始".center(scale//2,'-'))
t = time.clock()
for i in range(scale+1):
      a = '*' * i
      b = '.' * (scale - i)
      c = (i/scale)*100
      t -= time.clock()
      print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,-t),\ end='')
      time.sleep(0.05)
print("\n"+"执行结束".center(scale//2,'-'))
```