







Python

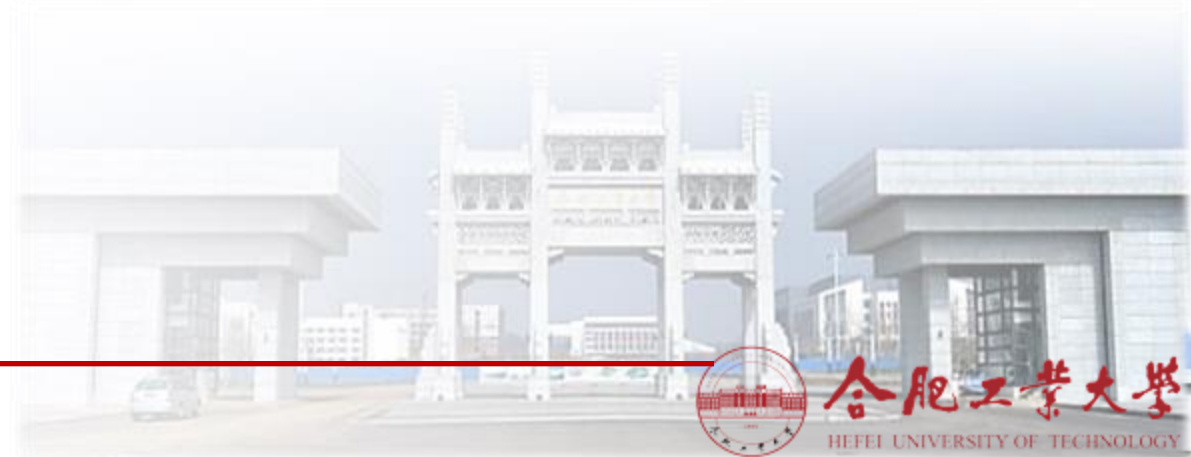
数据分析及可视化-5

字符数据处理






本章学习目标

-  熟练掌握字符串对象及其常用方法
-  熟练掌握字符串对象与其他对象互转
-  熟悉正则表达式的规则
-  熟练掌握正则表达式查找、分割与替换





字符串概述

-  ASCII采用1个字节来对字符进行编码，最多只能表示256个符号。编码GB2312、GBK和CP936都是使用2个字节表示中文，UTF-8使用3个字节表示中文。
-  不同编码格式之间相差很大，采用不同的编码格式意味着不同的表示和存储形式，把同一字符存入文件时，写入的内容可能会不同，在理解其内容时必须了解编码规则并进行正确的解码。
-  Python 3.x完全支持中文，使用Unicode编码格式，无论是一个数字、英文字母，还是一个汉字，都按一个字符对待和处理。在Python 3.x中甚至可以使用中文作为变量名。





字符串



Python支持短字符串驻留机制，对于短字符串，将其赋值给多个不同的对象时，内存中只有一个副本，多个对象共享该副本，与其他类型数具有相同的特点。然而，这一点并不适用于长字符串，长字符串不遵守驻留机制，下面的代码演示了短字符串和长字符串在这方面的区别。

```
>>> a = '1234'
>>> b = '1234'
>>> id(a) == id(b)
```

#短字符串支持内存驻留机制

True

```
>>> a="1234"*5000000
>>> b="1234"*5000000
>>> id(a) == id(b)
```

#长字符串不支持内存驻留机制

False

```
>>> s = '中国安徽省合肥市SDIBT'
>>> len(s)
```

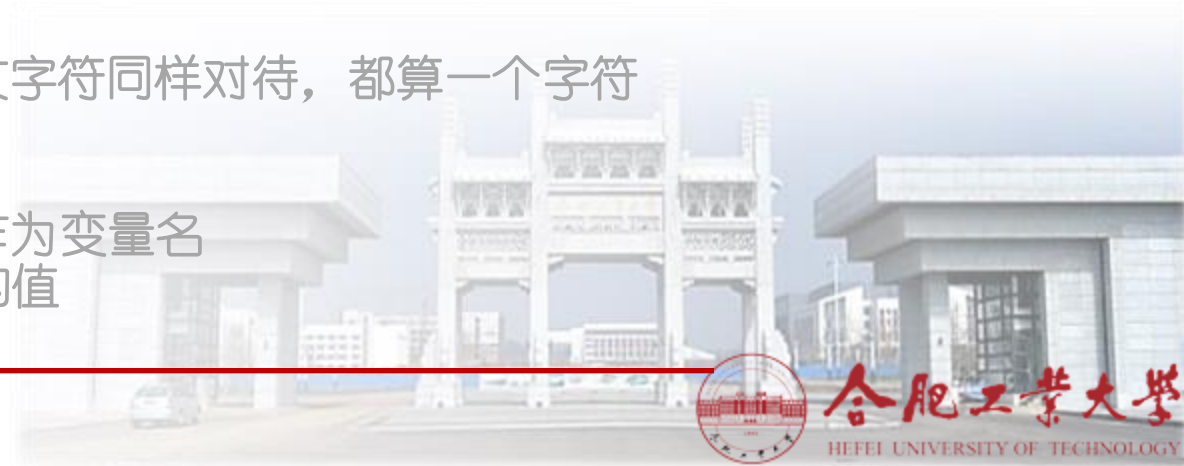
#中文与英文字符同样对待，都算一个字符

11

```
>>> 姓名 = '张三'
>>> print(姓名)
```

#使用中文作为变量名
#输出变量的值

张三



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY



字符串常用方法

字符串属于不可变对象，所有方法都是返回处理后的新字符串或新字节串，不对原字符串进行任何修改。

操作	含义
\	转义字符
%	格式
<string>.format	格式
<string>[]	索引
<string>[:]	剪切
len(<string>)	长度
<string>.upper()	字符串中字母大写
<string>.lower()	字符串中字母小写
<string>.strip()	去两边空格及去指定字符
<string>.split()	按指定字符分割字符串为数组
<string>.join()	连接两个字符串序列
<string>.find()	搜索指定字符串，若找到，返回该字符串的位置索引，否则返回-1
<string>.replace()	字符串替换
for <var> in <string>	字符串遍历





字符串常用方法-格式



字符串转义符

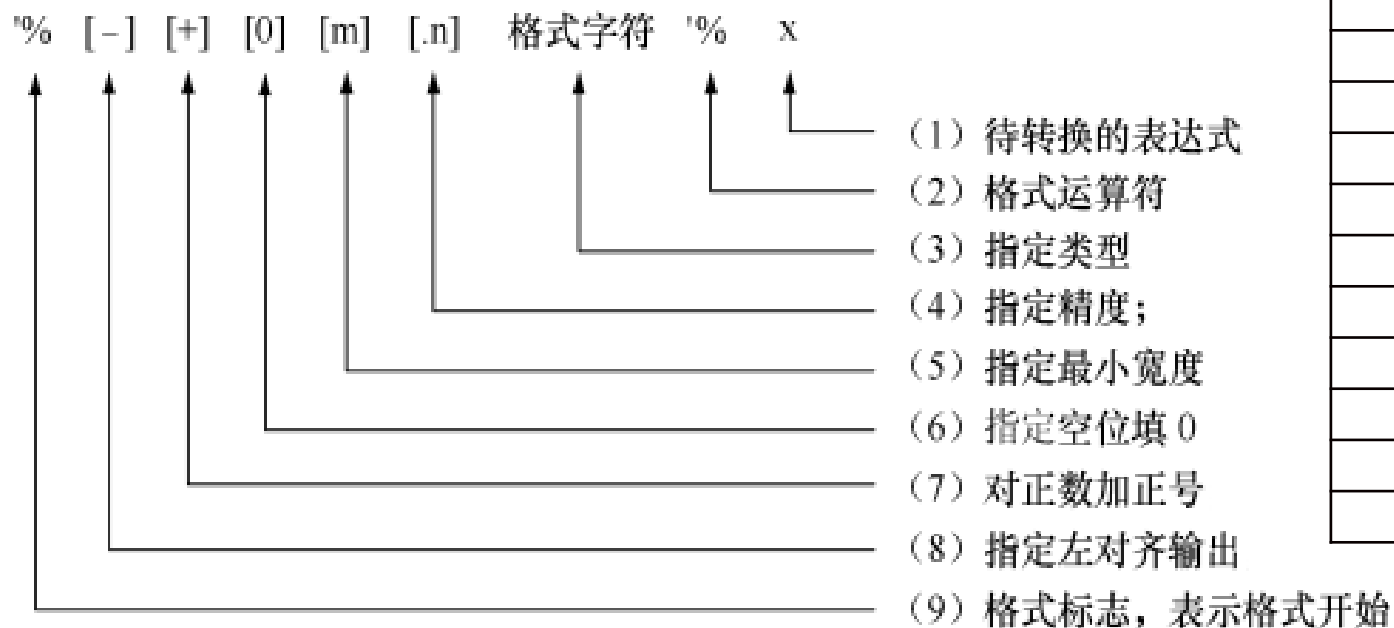
转义字符	含义
<code>\b</code>	退格，把光标移动到前一列位置
<code>\f</code>	换页符
<code>\n</code>	换行符
<code>\r</code>	回车
<code>\t</code>	水平制表符
<code>\v</code>	垂直制表符
<code>\\</code>	一个\
<code>\'</code>	单引号'
<code>\"</code>	双引号"
<code>\ooo</code>	3位八进制数对应的字符
<code>\xhh</code>	2位十六进制数对应的字符
<code>\uhhhh</code>	4位十六进制数表示的Unicode字符





Python 字符串常用方法-格式

Python 字符串格式化-%



格式字符	说明
<code>%s</code>	字符串 (采用 <code>str()</code> 的显示)
<code>%r</code>	字符串 (采用 <code>repr()</code> 的显示)
<code>%c</code>	单个字符
<code>%d</code>	十进制整数
<code>%i</code>	十进制整数
<code>%o</code>	八进制整数
<code>%x</code>	十六进制整数
<code>%e</code>	指数 (基底写为e)
<code>%E</code>	指数 (基底写为E)
<code>%f</code> 、 <code>%F</code>	浮点数
<code>%g</code>	指数(e)或浮点数 (根据显示长度)
<code>%G</code>	指数(E)或浮点数 (根据显示长度)
<code>%%</code>	字符" <code>%</code> "





字符串常用方法-格式

字符串格式化-format

```
>>> print("The number {0:}, in hex is: {0:#x}, the number {1} in oct is {1:#o}".format(5555,55))  
The number 5,555 in hex is: 0x15b3, the number 55 in oct is 0o67
```

```
>>> print( "my name is {name}, my age is {age}, and my QQ is {qq}" .format(name= "Sheng  
Zhongguo" ,age= 40,qq= "30646****" )) #使用关键参数, 与参数位置无关。
```

```
my name is Sheng Zhongguo, my age is 40, and my QQ is 30646****
```

```
>>> position = (5, 8, 13)
```

```
>>> print( "X:{0[0]};Y:{0[1]};Z:{0[2]}" .format(position))
```

#使用元组参数

```
X:5;Y:8;Z:13
```



字符串前面加字符**f**, **Python 3.6**之后的版本支持这种用法:

```
>>> width = 8
```

```
>>> height = 6
```

```
>>> print(f"Rectangle of {width}*{height}\nArea:{width*height}")
```



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY



字符串常用方法-格式

center()、ljust()、rjust()

 返回指定宽度的新字符串，原字符串居中、左对齐或右对齐出现在新字符串中，如果指定宽度大于字符串长度，则使用指定的字符（默认为空格）进行填充。

```
>>> str1='Hello world!'
```

```
>>> str1.center(20)
```

#居中对齐，以空格进行填充

```
'    Hello world!    '
```

```
>>> str1.center(20, '=')
```

#居中对齐，以字符=进行填充

```
'====Hello world!===='
```

```
>>> str1.ljust(20, '=')
```

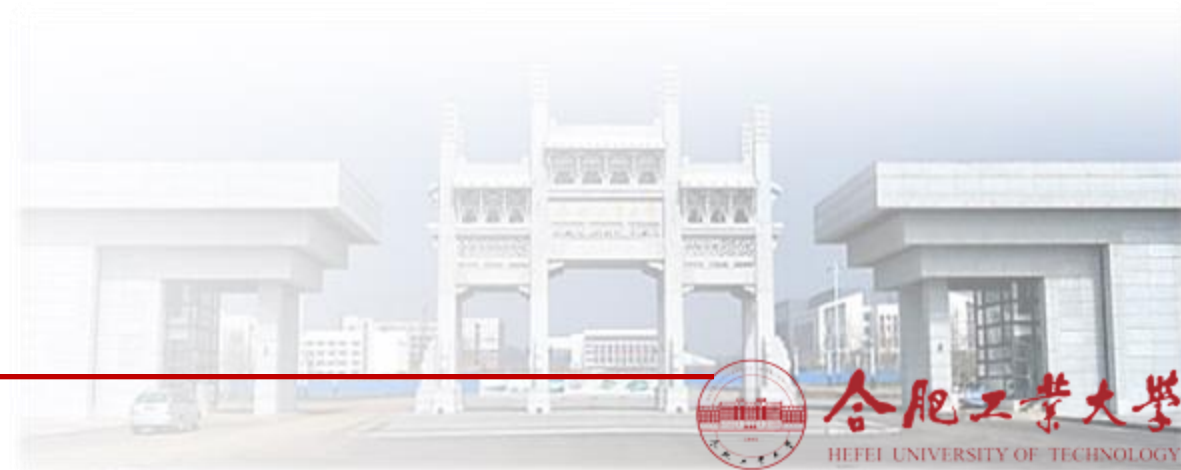
#左对齐

```
'Hello world!====='
```

```
>>> str1.rjust(20, '=')
```

#右对齐

```
'=====Hello world!'
```





字符串常用方法

 **isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()**

 用来测试字符串是否为数字或字母、是否为字母、是否为数字字符、是否为空白字符、是否为大写字母以及是否为小写字母。

```
>>> '1234abcd'.isalnum()
```

```
True
```

```
>>> '1234abcd'.isalpha()
```

```
False
```

```
>>> '1234abcd'.isdigit()
```

```
False
```

```
>>> 'abcd'.isalpha()
```

```
True
```

```
>>> '1234'.isdigit()
```

```
True
```

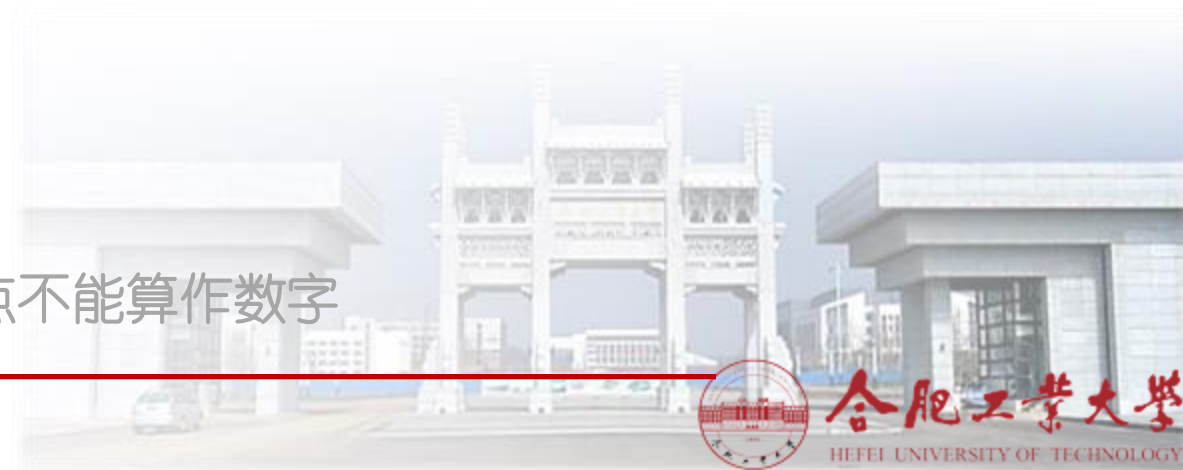
```
>>> '1234.0'.isdigit()
```

```
False
```

#全部为英文字母时返回True

#全部为数字时返回True

#小数点不能算作数字



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY



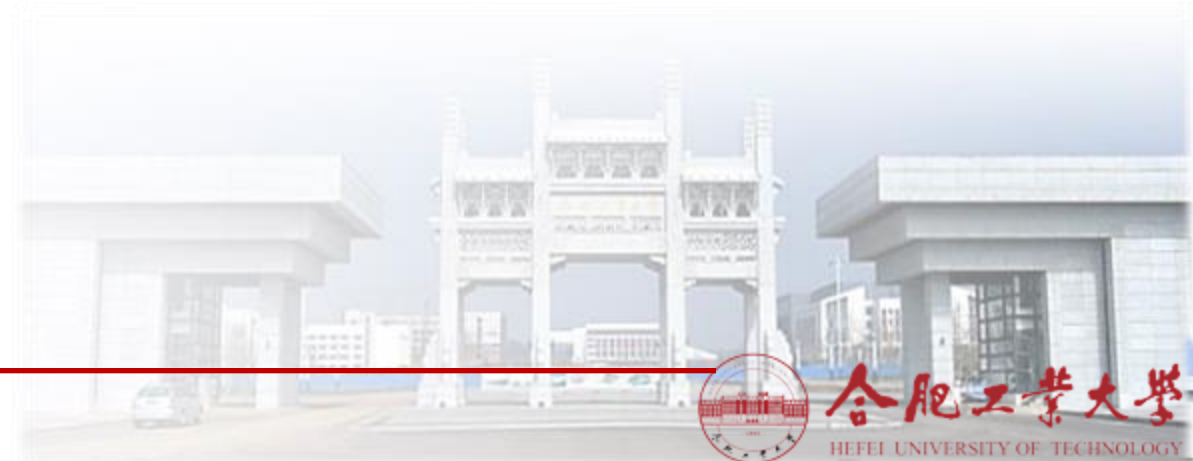
字符串常用方法

lower()、upper()、capitalize()、title()、swapcase()



对英文字符串进行转成小写、大写，字符串首字母大写，单词首字母大写及大小写字母相互转换操作。

```
>>> text="I am a student. you are a teacher."  
>>> text.lower()  
'i am a student. you are a teacher.'  
>>> text.upper()  
'I AM A STUDENT. YOU ARE A TEACHER.'  
>>> text.capitalize()  
'I am a student. you are a teacher.'  
>>> text.title()  
'I Am A Student. You Are A Teacher.'  
>>> text.swapcase()  
'i AM A STUDENT. YOU ARE A TEACHER.'
```





字符串常用方法

strip()、rstrip()、lstrip()

```
>>> s = " abc "
>>> s.strip()
'abc'
>>> s
' abc '
>>> "\n\nhello world \t\n".strip()
'hello world'
>>> "assddfa".strip("a")
'ssddf'
>>> "abfasddfab".strip("abf")
'sdd'
>>> "asddfaa".rstrip("a")
'asddf'
>>> "assddfaa".lstrip("a")
'ssddfaa'
```

#删除空白字符

#原字符变量没有改变
#删除空白字符，包括\n \t

#删除指定字符

#删除字符串两端外侧所有包含的字符

#删除字符串右端指定字符

#删除字符串左端指定字符





字符串常用方法

Startswith()、endswith()

 判断字符串是否以指定字符串开始或结束

```
>>> s = 'Beautiful is better than ugly.'
```

```
>>> s.startswith('Be') #检测整个字符串
```

True

```
>>> s.startswith('Be', 5) #指定检测范围起始位置
```

False

```
>>> s.startswith('Be', 0, 5) #指定检测范围起始和结束位置
```

True

```
>>> import os
```

```
>>> [filename for filename in os.listdir('c:\\') if filename.endswith(('.bmp', '.jpg', '.gif'))]  
#将C:\下所有以.bmp、.jpg、.gif为扩展名的文件名构成一个列表
```





字符串常用方法

replace()



replace()方法：把字符串中的 subold (旧子字符串) 替换成 subnew(新子字符串)，如果指定第三个参数max，则替换不超过 max 次，替换的结果生成一新字符串。

语 法：str.replace(subold, subnew, max)

参 数：subold为旧子字符串；subnew为新子字符串；max为替换的次数，默认为替换所有的旧子字符串。

返回值：返回一个新字符串。

```
>>> str = "this is string example....wow!!! this is really string."
```

```
>>> print(str.replace("is", "was"))
```

```
thwas was string example....wow!!! thwas was really string.
```

```
>>> print(str.replace("is", "was", 2))
```

```
thwas was string example....wow!!! this is really string.
```

```
>>> print(str.replace(" is ", " was "))
```

```
this was string example....wow!!! this was really string.
```





字符串常用方法

find()



find()方法：检测字符串中是否包含子字符串，如果指定start(开始)和end(结束)范围，则检查在指定范围内是否包含，如果包含子字符串返回开始的索引值，否则返回-1。

语 法：str.find(substr, start, end)

参 数：substr为子字符串；start为指定开始查找的位置索引，默认为0；end为结束查找的位置索引，默认为字符串的长度。

返回值：返回查找子字符串的位置索引。

```
>>> text = "中国是一个发展中的国家。"
```

```
>>> print(text.find('中'))
```

```
0
```

```
>>> print(text.find('中', 2))
```

```
7
```

```
>>> print(text.find('中', 2, 5))
```

```
-1
```





字符串常用方法

split()、rsplit()



split()方法

语 法: `str.split(spt="", num=-1)`

参 数: `spt` -- 分隔符, 默认为所有的空字符, 包括空格、换行(`\n`)、制表符(`\t`)等。
`num` -- 分割次数。默认为 `-1`, 即分隔所有。

返回值: 返回分割后的字符串列表。

```
>>> text = 'Beautiful is better than ugly.'
```

```
>>> print(text.split())
```

使用空白字符进行分隔

```
['Beautiful', 'is', 'better', 'than', 'ugly.']
```

```
>>> print(text.split(maxsplit=1))
```

最多分隔一次

```
['Beautiful', 'is better than ugly.']
```

```
>>> print(text.rsplit(maxsplit=2))
```

最多分隔两次

```
['Beautiful is better', 'than', 'ugly.']
```

```
>>> print('1,2,3,4'.split(','))
```

使用逗号作为分隔符

```
['1', '2', '3', '4']
```





字符串常用方法

join()



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。

语 法: `spt.join(seq)`

参 数: `spt` - 用以连接的指定字符, 可以为空字符或其他字符。
`seq` - 要连接的元素序列。

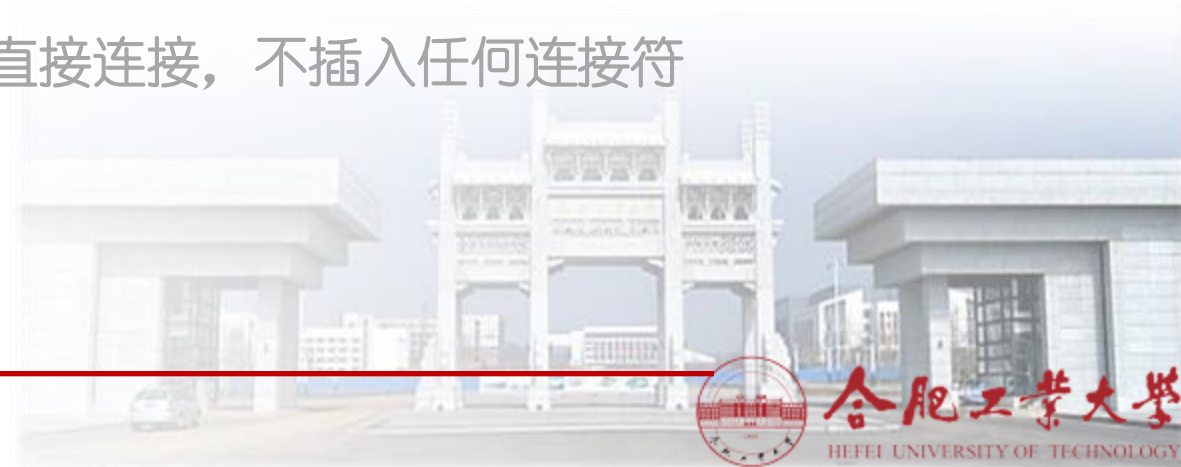
返回值: 返回通过指定字符连接序列中元素后生成的新字符串。

```
>>> ','.join(['1', '2', '3', '4'])    # 使用逗号作为连接符
```

```
'1, 2, 3, 4'
```

```
>>> ''.join(map(str, range(10)))      # 直接连接, 不插入任何连接符
```

```
'0123456789'
```





字符串常用方法

count()、index()、rindex()

 统计字符串中字符出现的次数，字符在字符串中位置索引。

```
>>> text = '处处飞花飞处处；声声笑语笑声声。'
```

```
>>> print(text.index('声'))
```

8

```
>>> print(text.rindex('处'))
```

6

```
>>> print(text.count('处'))
```

4

注：许多对象的方法实验对象若不存在，代码运行就会报错，例如字符串的index()方法，若子字符串不存在，就不能返回该子串对象在字符串中的位置索引，产生**异常出错**。避免这样的问题，可以先使用**count()**方法检测字符串是否包含子串对象，若包含，再使用index()返回子串对象的位置索引。





字符串常用方法

maketrans()、translate()



字符串对象的`maketrans()`方法用来生成字符映射表，而`translate()`方法用来根据映射表中定义的对应关系转换字符串并替换其中的字符，使用这两个方法的组合可以同时处理多个字符。

#创建映射表，将字符“abcdef123”——对应地转换为“uvwxyz@#”

```
>>> table = str.maketrans('abcdef123', 'uvwxyz@#')
```

```
>>> s = "Python is a greate programming language. I like it!"
```

```
>>> s.translate(table)           #按映射表进行替换
```

```
'Python is u gryuty proqramming lunguugy. I liky it!'
```

```
>>> table = str.maketrans('0123456789', '零一二三四伍陆柒捌玖')
```

```
>>> print('Tel:30647359'.translate(table))
```

```
Tel:三零陆四柒三伍玖
```





字符串与其他序列转换



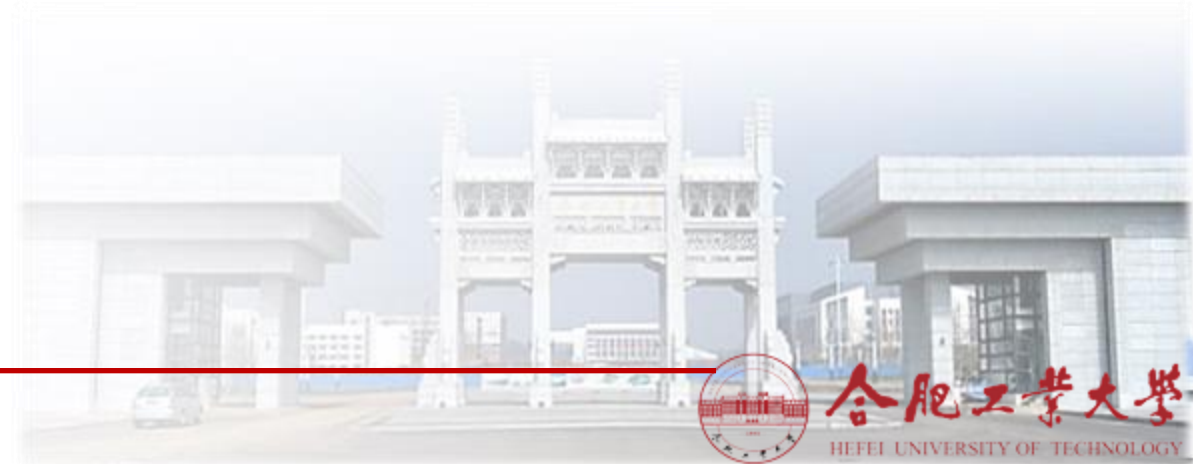
字符串和其他序列相互转换

```
>>> str1="abcdefg"
>>> s_lst=list(str1)
>>> print(s_lst)
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> s_tup=tuple(str1)
>>> print(s_tup)
('a', 'b', 'c', 'd', 'e', 'f', 'g')
>>> s_set=set(str1)
>>> print(s_set)
{'f', 'd', 'e', 'a', 'g', 'b', 'c'}
```

```
>>> "".join(s_lst)
'abcdefg'
```

```
>>> "".join(s_tup)
'abcdefg'
```

```
>>> "".join(s_set)
'abcdefg'
```





字符串应用案例

■ 【例】：凯撒加密，每个字母用后面第k个字母替换。

```
>>> import string
>>> def kaisa(s, k):
    lower = string.ascii_lowercase           #小写字母
    upper = string.ascii_uppercase          #大写字母
    before = string.ascii_letters
    after = lower[k:] + lower[:k] + upper[k:] + upper[:k]
    table = ''.maketrans(before, after)      #创建映射表
    return s.translate(table)

>>> s = "Python is a greate programming language. I like it!"
>>> kaisa(s, 3)
'Sbwkrq lv d juhduh surjudpplqj odqjxdjh. L olnh lw!'
```





字符串应用案例

【例】：编写函数实现字符串加密和解密，循环使用指定密钥，采用简单的异或算法。


```
>>> from itertools import cycle
>>> def crypt(source, key):
    result = ''
    temp = cycle(key)
    for ch in source:
        result = result + chr(ord(ch)^ord(next(temp)))
    return result


>>> source = 'Hefei University of Technology'
>>> key = '123456'
>>> encrypted = crypt(source, key)
>>> print(encrypted)
yWUQ\d\ZBPDB[GMYWgQV^_]_[R0
>>> print(crypt(encrypted, key))
Hefei University of Technology
```





正则表达式

 正则表达式使用某种**预定义的模式**去匹配一类具有共同特征的字符串，主要用于处理字符串，可以快速、准确地完成复杂的**查找**、**替换**等处理要求，在**文本编辑与处理**、**网页爬虫**之类的场合中有重要应用。

 正则表达式由丰富的符号构成，可分成二类：**普通字符**和**元字符**。普通字符即为**所表达的字符自身**，包括数字、字母和下画线字符。元字符为正则表达式中**具有特殊含义的专用字符**，可以用来规定其前导字符(位于元字符之前的字符)在目标对象中出现的模式。

 Python中，**re模块**提供了正则表达式操作所需要的功能。





正则表达式

元字符	功能说明
.	匹配除换行符以外的任意单个字符
*	匹配位于*之前的字符或子模式的0次或多次出现
+	匹配位于+之前的字符或子模式的1次或多次出现
-	在[]之内用来表示范围
	匹配位于 之前或之后的字符
^	匹配行首，匹配以^后面的字符开头的字符串
\$	匹配行尾，匹配以\$之前的字符结束的字符串
?	匹配位于?之前的0个或1个字符。 当此字符紧随任何其他限定符 (*、+、?、{n}、{n,}、{n,m}) 之后时，匹配模式是“非贪心的”。“非贪心的”模式匹配搜索到的、尽可能短的字符串，而默认的“贪心的”模式匹配搜索到的、尽可能长的字符串。例如，在字符串“oooo”中，“o+?”只匹配单个“o”，而“o+”匹配所有“o”
\	表示位于\之后的为转义字符
\num	此处的num是一个正整数，表示子模式编号。例如，“(.)\1”匹配两个连续的相同字符
\f	换页符匹配
\n	换行符匹配





正则表达式

元字符	功能说明
\r	匹配一个回车符
\b	匹配单词的边界
\B	与\b含义相反，匹配不出现在单词边界的字符
\d	匹配任何数字，相当于[0-9]
\D	与\d含义相反，等效于[^0-9]
\s	匹配任何空白字符，包括空格、制表符、换页符，与 [\f\n\r\t\v] 等效
\S	与\s含义相反
\w	匹配任何字母、数字以及下划线，相当于[a-zA-Z0-9_]
\W	与\w含义相反，与 “[^A-Za-z0-9_]” 等效
()	将位于()内的内容作为一个整体来对待
{m, n}	{ }前的字符或子模式重复至少m次，至多n次
[]	表示范围，匹配位于[]中的任意一个字符
[^xyz]	反向字符集，匹配除x、y、z之外的任何字符
[a-z]	字符范围，匹配指定范围内的任何字符
[^a-z]	反向范围字符，匹配除小写英文字母之外的任何字符





正则表达式

- 最简单的正则表达式是普通字符串，可以匹配自身
- `'[pjcy]ython'` 可以匹配 `'python'`、`'jython'`、`'cython'`
- `'[^abc]'` 可以一个匹配任意除 `'a'`、`'b'`、`'c'` 之外的字符
- `'python|perl'` 或 `'p(ython|erl)'` 都可以匹配 `'python'` 或 `'perl'`
- `'^[a-zA-Z]{1}[a-zA-Z0-9._]{4,19}'`
匹配长度为5-20的字符串，必须以字母开头并且可带字母、数字、“_”、“.”的字符串。
- `'^(13[4-9]\d{8})|(15[01289]\d{8})'` 检查给定字符串是否为手机号码。
- `'^[a-zA-Z]+'` 检查给定字符串是否只包含英文字母大小写。
- `'[\u4e00-\u9fa5]'` 匹配给定字符串中所有汉字。
- `'^\d{18}|\d{15}$'` 检查给定字符串是否为合法身份证格式。
- `'\d{4}-\d{1,2}-\d{1,2}'` 匹配指定格式的日期，例如2016-1-31。





re模块方法与正则表达式

方法	功能说明
<code>compile(pattern[, flags])</code>	创建模式对象
<code>match(pattern, string[, flags])</code>	从字符串的 开始处 匹配模式，返回match对象或None
<code>search(pattern, string[, flags])</code>	在 整个字符串 中寻找模式，返回match对象或None
<code>findall(pattern, string[, flags])</code>	返回包含字符串中所有与给定模式匹配的项的列表
<code>finditer(pattern, string, flags=0)</code>	返回包含所有匹配项的迭代对象，其中每个匹配项都是match对象
<code>fullmatch(pattern, string, flags=0)</code>	尝试把模式作用于整个字符串，返回match对象或None
<code>split(pattern, string[, maxsplit=0])</code>	根据模式匹配项分隔字符串
<code>sub(pat, repl, string[, count=0])</code>	将字符串中所有与pat匹配的项用repl替换，返回新字符串，repl可以是字符串或返回字符串的可调用对象，作用于每个匹配的match对象
<code>subn(pat, repl, string[, count=0])</code>	将字符串中所有pat的匹配项用repl替换，返回包含新字符串和替换次数的二元组 repl可以是字符串或返回字符串的可调用对象，作用于每个匹配的match对象





re模块方法与正则表达式对象

re模块中的方法_search()

```
>>> import re    #导入re模块

>>> s="I love Python very much."
>>> pat1=' \bP\w+\b'
>>> pat2=r' \bP\w+\b'
>>> r1=re.search(pat1,s)
>>> r2=re.search(pat2,s)

>>> print(r1)
None
>>> print(r2)
<re.Match object; span=(7, 13), match='Python'>

>>> r2.span()
(7, 13)
>>> r2.group()
'Python'
```

re模块中的`search()`方法用于在一个字符串中是否存在与正则表达式相匹配的子串，如不存在，则返回`None`，如存在，则返回一个`re.Match`对象，

`re.Match`对象的`span()`方法返回一个元组，表示子串出现的开始和结束(不包含)位置索引，`group()`方法返回与正则表达式相匹配的一个或多个子串，`groups()`方法返回一个包含匹配的所有子模式内容的元组。

注意：此处的两个正则表达式模式`pat1`和`pat2`略有区别，导致`search()`方法的结果不同。其原因是在这两个正则表达式模式中存在的`\b`，当正则表达式模式前不加`r`，解释器认为是“转义字符”退格键`backspace`”，如果前面加`r`，那么解释器不会进行转义，`\b`解释为正则表达式模式中的字符串边界。





re模块方法与正则表达式对象

re模块中的方法_search()

```
>>> s = '<html><head>This is head.</head><body>This is body.</body></html>'
#这是一个网页html格式的字符串，使用正则表达式提取字符串中的“标签”及“标签”之间的“字符串”
>>> pattern = r'<(\w+)><(\w+)>(.+)</\2><(\w+)>(.+)</\4></\1>'
#使用()表示子模式，匹配子模式
>>> result = re.search(pattern, s)
>>> result.groups()                #所有子模式的元组
('html', 'head', 'This is head.', 'body', 'This is body.')
>>> result.group(0)                #与正则表达式模式相匹配的字符串
'<html><head>This is head.</head><body>This is body.</body></html>'
>>> result.group(3)                #第三个子模式
'This is head.'
>>> result.group(3, 5)             #第三、五个子模式
('This is head.', 'This is body.')
```





re模块方法与正则表达式对象

re模块中的方法_findall()

语 法: `re.findall(pattern, string[, flags])`

参 数: `pttern`: 正则表达式;

`string`: 字符串;

`flag`: 可取`re.I`(忽略字母大小写), `re.S`(使元字符"."匹配任意字符, 包括换行符), `re.U`(匹配Unicode字符), `re.M`(多行匹配模式)。

返回值: 返回是正则表达式在字符串中所有匹配结果的字符串列表。

```
>>> telNumber='Suppose my Phone No.is 0535-1234567,yours is 010-12345678,his is 025-87654321.'
```

```
>>> pat='\d{3,4}-\d{7,8}'          #查找所有的电话号码
```

```
>>> r=re.findall(pat,telNumber)
```

```
>>> print(r)
```

```
['0535-1234567', '010-12345678', '025-87654321']
```

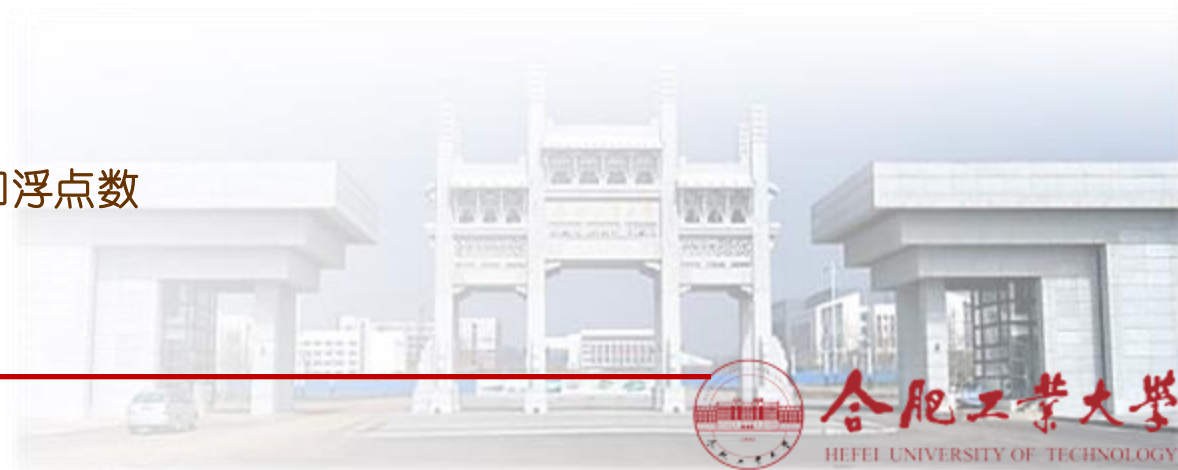
```
>>> s="有1个源代码文件,有20行代码,运行时间10.25s."
```

```
>>> pat='\d+\.?\d*'          #获取字符串中所有的整数和浮点数
```

```
>>> r=re.findall(pat,s)
```

```
>>> print(r)
```

```
['1', '20', '10.25']
```





re模块方法与正则表达式对象

re模块中的方法_findall()



贪心模式和非贪心模式(在*或+后面加上?)

```
>>> s = 'Beautiful is better than ugly.'
>>> re.findall(r'\bb.*?', s)      #此处问号?表示非贪心模式, ?前面是*, 因此 .*? 可以不匹配字符
['b']
>>> re.findall(r'\bb.+?', s)      #此处问号?表示非贪心模式, ?前面是+, 因此 .+? 只匹配1个字符
['be']
>>> re.findall(r'\bb.+', s)        #贪心模式的匹配结果, 取到整个字符串结束
['better than ugly.']
>>> re.findall(r'\bb.+?\b', s)     #此处问号?表示非贪心模式, 匹配到第一个单词结束
['better']
>>> re.findall(r'\bb.+?\b', s)     #贪心模式匹配结果, 匹配到字符串最后一个单词结束, 不包括尾部的 '.'
['better than ugly']
```



Flags参数

```
>>> re.findall(r'\bb\w+\b', s, re.I) #匹配所匹配单词中字母的大小写
['Beautiful', 'better']
```





re模块方法与正则表达式对象

re模块中的方法_findall()

```
>>> s=''' {'info': [{'name': '穆凤英', 'telephone': '13174029815', 'email': 'minjia@hotmail.com',  
'postcode': '367994'}, {'name': '赵玉', 'telephone': '18580052853', 'email': 'xiuying85@hotmail.com',  
'postcode': '838745'}, {'name': '周凯', 'telephone': '15181094459', 'email': 'konggang@gmail.com',  
'postcode': '722565'}, {'name': '胡丽', 'telephone': '15626154579', 'email': 'weili@hotmail.com',  
'postcode': '916941'}] } '''
```

#从s字符串中筛选出所有人的姓名

```
>>> namelst=re.findall('[\u4e00-\u9fa5]+',s)
```

```
>>> namelst
```

```
['穆凤英', '赵玉', '周凯', '胡丽']
```

#使用子模式(), 分别把每个人的name、telephone、email和postcode信息提取出来

```
>>> pat=r''' name': '(.+?)'.+?one': '(\d{11})'.+?ail': '(.+?)'.+?de': '(\d{6})' '''
```

```
>>> infolst=re.findall(pat,s)
```

```
>>> infolst
```

```
[('穆凤英', '13174029815', 'minjia@hotmail.com', '367994'), ('赵玉', '18580052853', 'xiuying85@hotmail.com',  
'838745'), ('周凯', '15181094459', 'konggang@gmail.com', '722565'), ('胡丽', '15626154579',  
'weili@hotmail.com', '916941')]
```





re模块方法与正则表达式对象

re模块中的方法_sub()

如果希望开头的字母
b忽略大小写，那么
正则表达式如何设计？

语 法: `re.sub(pattern, repl, string[, count=0])`

参 数: `pattern`为正则表达式;`repl`为替换匹配正则表达式的字符串; `string`为原字符串;`count`为替换次数，默认为0，表示替换所有匹配的内容。

返回值: 返回是替换后的新字符串。

```
>>> s = '''Beautiful is better than ugly.Explicit is better than implicit.Simple  
is better than complex.Complex is better than complicated.Flat is better than  
nested.Sparse is better than dense.Readability counts.'''
```

```
>>> pat=r'\bb\w*\b' #将所有以b开头的单词都用*替换
```

```
>>> re.sub(pat, '*', s)
```

```
'Beautiful is * than ugly.Explicit is * than implicit.Simple is * than  
complex.Complex is * than complicated.Flat is * than nested.Sparse is * than  
dense.Readability counts.'
```

```
>>> s = "It's a very good good good idea"
```

#处理连续的重复单词

```
>>> re.sub(r'(\w+\s)(\1+)', r'\1', s)
```

```
"It's a very good idea"
```





re模块方法与正则表达式对象

re模块中的方法_split()

语 法: `re.split(pattern, string[, maxsplit=0])`

参 数: pattern: 正则表达式;

string: 原字符串;

maxsplit: 从左向右分割次数, 默认为0, 分割所有的匹配。

返回值: 返回分割后的字符串列表。

```
>>> s = 'one,two,three.four/five\six?seven[eight]nine|ten'
>>> pat = r'[, \. / \? [\] \|]'          #用[]括起多个可能的分隔符
>>> re.split(pat, s)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

```
>>> s = 'onetwo2three3four4five5six6seven7eight8nine9ten'
>>> pat = r'\d+'                          #使用数字作为分隔符
>>> re.split(pat, s)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

```
>>> s = 'one two three four,five.six.seven,eight,nine9ten'
>>> pat = r'[\s,.\d]+'                    #允许分隔符重复
>>> re.split(pat, s)
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```





re模块方法与正则表达式对象



【实际案例】随便打开一个旅游网站，爬下该网页中信息字符串，从中用正则表达式获取该网页的景点数据。

```
>>> import re, requests
>>> url="https://travelsearch.fliggy.com/index.htm?spm=181.61408.alz7d.41.539a23e40fbBSb&searchType=product&keyword=%E6%88%90%E9%83%BD&category=SCENIC"
>>> rtext=requests.get(url).text
>>> pat=re.compile(r'<h3 class="main-title">[^<]+</h3>')
>>> pat.search(r.text)
<re.Match object; span=(19780, 19811), match='<h3 class="main-title">都江堰</h3>'>
>>> lst=pat.findall(r.text)
>>> pat=re.compile(r'<[^>]+>')
>>> for i in range(len(lst0)):
>>>>>     lst0[i]=pat.sub("", lst0[i])
>>>>> lst0[:5]
['都江堰', '蜀风雅韵', '大熊猫繁育研究基地', '青城山', '成都欢乐谷']
```





re模块方法与正则表达式对象



【实际案例二】打开一腾讯视频获取其弹幕JSON数据，用正则表达式获取该数据中的相关信息

```
>>> import re, requests
>>> url10_s='https://v.qq.com/x/cover/mzc0020018dipbu.html'
>>> url_d='https://mfm.video.qq.com/danmu?otype=json&timestamp=45&target_id=6384458060%26vid%3Dd0035ka5c02&count=80'
>>> rtext=requests.get(url).text
>>> rtext[:2000]
```

```
{
  "err_code": 0,
  "err_msg": "ok",
  "peroid": "30",
  "target_id": "6384458060",
  "count": 211,
  "tol up": 0,
  "single max count": 0,
  "session_key": "0,0,0",
  "comments": [
    {
      "commentid": "6755691266521100321",
      "content": "虞书欣姐姐来啦",
      "upcount": 17,
      "isfriend": 0,
      "isop": 0,
      "isself": 0,
      "timepoint": 36,
      "headurl": "https://typic.gtimg.cn/head/41de4e63df65ee72d67ce002fcbbf22984a60d16f798948f7161de140840a93884066a34/202",
      "opername": "柚稚",
      "bb_bcolor": "",
      "bb_head": "",
      "bb_level": "http://i.gtimg.cn/qqlive/images/20170106/i1483692007_1.jpg",
      "bb_id": "",
      "rich_type": 0,
      "uservip_degree": 1,
      "content_style": {
        "color": "ffffff",
        "gradient_colors": ["FDA742", "FBF076"],
        "position": 1,
        "colorConfigId": "11",
        "lowVipDegree": 1
      },
      "commentid": "6755691597720466960",
      "content": "哈哈哈哈哈",
      "upcount": 0,
      "isfriend": 0,
      "isop": 0,
      "isself": 0,
      "timepoint": 36,
      "headurl": "http://thirdqq.qlogo.cn/g?b=sd&k=OD2Kj9Wm3Zqjec00Y7oVN&s=140&t=1560838255",
      "opername": "\u3000",
      "bb_bcolor": "",
      "bb_head": "",
      "bb_level": "",
      "bb_id": "",
      "rich_type": 0,
      "uservip_degree": 0,
      "content_style": ""
    },
    {
      "commentid": "6755692438771401379",
      "content": "2021年1月15日 11:49:51",
      "upcount": 1,
      "isfriend": 0,
      "isop": 0,
      "isself": 0,
      "timepoint": 36,
      "headurl": "",
      "opername": "",
      "bb_bcolor": "",
      "bb_head": "",
      "bb_level": "",
      "bb_id": "",
      "rich_type": 2,
      "uservip_degree": 0,
      "content_style": ""
    },
    {
      "commentid": "6755693306384468568",
      "content": "哈哈党：张颜齐",
      "upcount": 11,
      "isfriend": 0,
      "isop": 0,
      "isself": 0,
      "timepoint": 38,
      "headurl": "https://thirdwx.qlogo.cn/mmopen/vi_32/m0icS2kW3x7TWUmsDjTE0yqxhfKM0Lwn5CNwAibfLEabGolFEicC6WIaUKS812NibsQyKRYbnEyTDZUxyh6lrJ3Axw/132",
      "opername": "颜七思睿",
      "bb_bcolor": "",
      "bb_head": "https://vfiles.gtimg.cn/wupload/danmu_richmedia_online.danmu_pendant_conf/20210101_ae7mrrougbkgV2.png",
      "bb_level": "http://i.gtimg.cn/qqlive/images/20170106/i1483692012_1.jpg",
      "bb_id": "9611",
      "rich_type": 2,
      "uservip_degree": 2,
      "content_style": ""
    },
    {
      "commentid": "6755693883431255147",
      "content": "第一",
      "upcount": 5,
      "isfriend": 0,
      "isop": 0,
      "isself": 0,
      "timepoint": 35,
      "headurl": "",
      "opername": "",
      "bb_bcolor": "",
      "bb_head": ""
    }
  ]
}
```





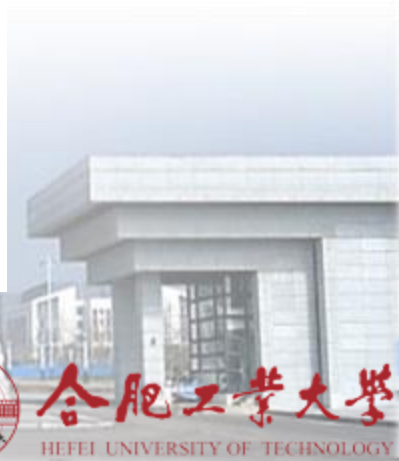
re模块方法与正则表达式对象



【实际案例二】打开一腾讯视频获取其弹幕JSON数据，用正则表达式获取该数据中的相关信息

```
>>> pat=r'"commentid": "(\\d+)", "content": (.+?), "upcount": (\\d+).+?"opername": "(.+?)".+?"uservip_degree": (\\d+)'
>>> dlst=re.findall(pat,r.text)
>>> len(dlst)
211
>>> for ls in data1st[:10]:
    print(ls)
```

```
(' 6755691266521100321', ' "虞书欣姐姐来啦"', '17', ' 柚稚', '1')
(' 6755691597720466960', ' "哈哈哈哈哈"', '0', ' ', '\\u3000', ', ', '0')
(' 6755692438771401379', ' " 2021年1月15日 11:49:51"', '1', ' "', '0')
(' 6755693306384468568', ' "哈哈党：张颜齐"', '11', ' ', '颜七思睿', ', ', '2')
(' 6755693883431255147', ' "第一"', '5', ' "', '1')
(' 6755693940909997228', ' "集合啦"', '3', ' "', '0')
(' 6755694240915979795', ' " 张颜齐理论"', '2', ' "', '0')
(' 6755695114905437061', ' "哈哈党：晨艺我来啦"', '13', ' "', '1')
(' 6755695454672057346', ' "来了姐妹们"', '2', ' ', '希望明天不上火', ', ', '0')
(' 6755696114710024570', ' " 晨艺我来了哈哈哈哈哈"', '6', ' "', '0')
```





祝学习进步!