## Python字符串内建函数

序言：

字符串属于序列类型，根据字符串内容的多少可以将字符串分为单行字符串和多行字符串。其中单行字符串可以由一对双引号(" ")或一对单引号(' ')表示，单引号和双引号等效。多行字符串可由一对三单引号(''' ''')或一对三双引号(""" """)表示，三单引号和三双引号也等效。

解决英文单词大小写问题：

## 1.0 capitalize()函数

## 描述：将字符串的第一个字母变成大写，其余字母变为小写。

语法：str.capitalize()  —> str 返回字符串

程序示例：

str1 = "i Love python"

str2 = " i Love python" #字母i前有空格

str3 = "I Love python"

print(str1.capitalize())

print(str2.capitalize())

print(str3.capitalize())

程序运行结果：

I love python

i love python

I love python

## 2.0 title()函数

## 描述：返回一个满足标题格式的字符串。即所有英文单词首字母大写，其余英文字母小写。

语法：str.title() -> str 返回一个字符串

程序示例：

str = "i love python"

print(str.title()) #将字符串str的所有单词首字母大写，其余字母小写

程序运行结果：

I Love Python

## 3.0 swapcase()函数

**描述：将字符串str中的大小写字母同时进行互换。即将字符串str中的大写字母转换为小写字母，将小写字母转换为大写字母。**

语法：str.swapcase() -> str 返回字符串

程序示例：

str1 = "I Love PYTHON"

str2 = "我爱python Python pYTHON"

print(str1.swapcase()) #将字符串str1中的大写字母转为小写字母，小写字母转换为大写字母。

print(str2.swapcase())

程序运行结果：

i lOVE python

我爱PYTHON pYTHON Python

## 4.0 lower()函数

## 描述：将字符串中的所有大写字母转换为小写字母。

语法：str.lower()  -> str  返回字符串

程序示例：

str1 = "I Love Python"

str2 = "Groß - α" #德语 大写α

print(str1.casefold())

print(str1.lower())

print(str2.casefold())

print(str2.lower())

程序运行结果：

i love python

i love python

gross - α

groß - α

注意 lower()函数和casefold()函数的区别：

lower() 方法只对ASCII编码，即‘A-Z’有效，对于其它语言中把大写转换为小写的情况无效，只能用 casefold() 函数。

## 5.0 upper()函数

## 描述：将字符串中的所有小写字母转换为大写字母。

语法： str.upper() -> str  返回字符串

程序示例：

str1 = "i love python" #全为小写字母

str2 = "I Love Python" #所有单词首字母大写

print(str1.upper())

print(str2.upper())

程序运行结果：

I LOVE PYTHON

I LOVE PYTHON

## 6.0 casefold()函数

## 描述：将字符串中的所有大写字母转换为小写字母。也可以将非英文 语言中的大写转换为小写。

注意 lower()函数和casefold()函数的区别：lower() 方法只对ASCII编码，即‘A-Z’有效，对于其它语言中把大写转换为小写的情况无效，只能用 casefold() 函数。

语法：str.casefold()   -> str 返回字符串

程序示例：

str1 = "I Love Python"

str2 = "Groß - α" #德语 大写α

print(str1.casefold())

print(str1.lower())

print(str2.casefold())

print(str2.lower())

程序运行结果：

i love python

i love python

gross - α

groß - α

解决字符串填充问题：

## 7.0 center()函数

## 描述：返回一个长度为width,两边用fillchar(单字符)填充的字符串，即字符串str居中，两边用fillchar填充。若字符串的长度大于width,则直接返回字符串str。

语法：str.center(width , "fillchar")  -> str  返回字符串       注意：引号不可省

width —— 指定字符串长度。

fillchar —— 要填充的单字符，默认为空格。

程序示例：

str = "i love python"

print(str.center(20,"\*")) #指定字符串长度为20 用单字符"\*"填充

print(str.center(1,"\*")) #指定字符串长度为1，小于原字符串的长度。

print(str.center(20,"8"))

print(str.center(20)) #fillchar 默认为空格

程序运行结果：

\*\*\*i love python\*\*\*\*

i love python

888i love python8888

i love python

## 8.0 ljust()函数

**描述：返回一个原字符串左对齐,并使用fillchar填充(默认为空格)至指定长度的新字符串。如果指定的长度小于原字符串的长度则返回原字符串。**

语法： str.ljust(width, fillchar)  -> str 返回一个新的字符串

width —— 指定字符串的输出长度。

fillchar—— 将要填充的单字符，默认为空格。

程序示例：

str = "python"

print(str.ljust(30,"\*")) #指定宽度为30，填充字符为\*

print(str.ljust(30)) #指定宽度为30，填充字符默认为空格

print(str.ljust(30),"1") #对比

程序运行结果：

python\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

python

python 1

## 9.0 rjust()函数

## 描述：返回一个原字符串右对齐,并使用fillchar填充(默认为空格)至指定长度的新字符串。如果指定的长度小于原字符串的长度则返回原字符串。

语法： str.ljust(width, fillchar)  -> str 返回一个新的字符串

width —— 指定字符串的输出长度。

fillchar—— 将要填充的单字符，默认为空格。

程序示例：

str = "python"

print(str.rjust(30,"\*")) #指定宽度为30，填充字符为\*

print(str.rjust(30)) #指定宽度为30，填充字符默认为空格

print("1",str.rjust(30)) #对比

程序运行结果：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*python

python

1 python

## 10.0 zfill()函数

## 描述：返回指定长度的字符串，使原字符串右对齐，前面用0填充到指定字符串长度。

语法：str.zfill(width) -> str 返回一个字符串

width —— 指定字符串的长度,但不能为空。若指定长度小于字符串长度，则直接输出原字符串。

程序示例：

str = "i love python"

print(str.zfill(30)) #指定字符串长度为30

print(str.zfill(2)) #指定字符串长度为2，小于原字符串长度。

print(str.zfill()) #参数width为空，报错.

程序运行结果：

00000000000000000i love python

i love python

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-22-45e4baf7a246> in <module>()

2 print(str.zfill(30)) #指定字符串长度为30

3 print(str.zfill(2)) #指定字符串长度为2，小于原字符串长度。

----> 4 print(str.zfill()) #参数width为空，报错.

TypeError: zfill() takes exactly 1 argument (0 given)

解决统计字符次数问题：

## 11.0 count(）函数

## 描述：统计字符串里某个字符出现的次数。可以选择字符串索引的起始位置和结束位置。

语法：str.count("char", start,end)  或 str.count("char")    -> int    返回整数

str —— 为要统计的字符(可以是单字符，也可以是多字符)。

star —— 为索引字符串的起始位置，默认参数为0。

end —— 为索引字符串的结束位置，默认参数为字符串长度即len(str)。

程序示例：

str = "i love python,i am learning python"

print(str.count("i")) #star 和end 为默认参数

print(str.count("i",2)) # star值为2，end值为默认参数

print(str.count("i",2,5)) #star值为2，end值为5

print(str.count("am")) #多字符统计

程序运行结果：

3

2

0

1

解决编码问题：

## 12.0 encode()函数

## 描述：以指定的编码格式编码字符串，默认编码为 'utf-8'。

语法：str.encode(encoding='utf-8', errors='strict')     -> bytes (获得bytes类型对象)

encoding 参数可选，即要使用的编码，默认编码为 'utf-8'。字符串编码常用类型有：utf-8,gb2312,cp936,gbk等。

errors 参数可选，设置不同错误的处理方案。默认为 'strict',意为编码错误引起一个UnicodeEncodeError。 其它可能值有 'ignore', 'replace', 'xmlcharrefreplace'以及通过 codecs.register\_error() 注册其它的值。

程序示例：

str1 = "我爱祖国"

str2 = "I love my country"

print("utf8编码：",str1.encode(encoding="utf8",errors="strict")) #等价于print("utf8编码：",str1.encode("utf8"))

print("utf8编码：",str2.encode(encoding="utf8",errors="strict"))

print("gb2312编码：",str1.encode(encoding="gb2312",errors="strict"))#以gb2312编码格式对str1进行编码，获得bytes类型对象的str

print("gb2312编码：",str2.encode(encoding="gb2312",errors="strict"))

print("cp936编码：",str1.encode(encoding="cp936",errors="strict"))

print("cp936编码：",str2.encode(encoding="cp936",errors="strict"))

print("gbk编码：",str1.encode(encoding="gbk",errors="strict"))

print("gbk编码：",str2.encode(encoding="gbk",errors="strict"))

程序运行结果：

utf8编码： b'\xe6\x88\x91\xe7\x88\xb1\xe7\xa5\x96\xe5\x9b\xbd'

utf8编码： b'I love my country'

gb2312编码： b'\xce\xd2\xb0\xae\xd7\xe6\xb9\xfa'

gb2312编码： b'I love my country'

cp936编码： b'\xce\xd2\xb0\xae\xd7\xe6\xb9\xfa'

cp936编码： b'I love my country'

gbk编码： b'\xce\xd2\xb0\xae\xd7\xe6\xb9\xfa'

gbk编码： b'I love my country'

注：在python中encode和decode分别指编码和解码

## 13.0 decode()函数

## 描述：以 encoding 指定的编码格式解码字符串，默认编码为字符串编码。

语法：str.decode(encoding='utf-8', errors='strict')

encoding ——要使用的编码，如：utf-8,gb2312,cp936,gbk等。

errors ——设置不同解码错误的处理方案。默认为 'strict',意为编码错误引起一个 UnicodeDecodeError。 其它可能得值有 'ignore', 'replace'以及通过 codecs.register\_error() 注册的1其它值。

程序示例：

s = "我爱祖国"

str1 = s.encode(encoding="utf-8",errors="strict")

str2 = s.encode("gb2312") #编码错误的处理方案默认为"strict"

str3 = s.encode("gbk")

print(str1.decode(encoding="utf-8",errors="strict"))#用utf-8的解码格式，解码str1.

print(str1.decode(encoding="gbk",errors="ignore"))##如果以gbk的解码格式对str1进行解码得，将无法还原原来的字符串内容

print(str1.decode(encoding="gbk",errors="strict"))

print(str1.decode(encoding="gbk",errors="replace"))

print("\n")

print(str2.decode("gb2312"))

print(str3.decode("gbk"))

程序运行结果：

我爱祖国

鎴戠埍绁栧浗

鎴戠埍绁栧浗

鎴戠埍绁栧浗

我爱祖国

我爱祖国

 注：在python3.x中，encode()函数只能用于字符串类型，而decode()函数只能用于字节数据类型。

程序示例中 str1,str2,str3都是字节数据类型(通过encode()函数把 字符串类型s 转换为字节数据类型)。

## 14.0 expandtabs()函数

**描述：返回一个字符串的副本。使原字符串中的制表符（"\t"）的使用空间变大。使用空格来扩展空间。**

语法： str.expandtabs(tabsize=8)  —> str  返回字符串

tabsize 的默认值为8。tabsize值为0到7等效于tabsize=8。tabsize每增加1，原字符串中“\t”的空间会多加一个空格。

程序示例：

str = "i love\tpython"

print(str.expandtabs())#默认值为8

print(str.expandtabs(tabsize=8))

print(str.expandtabs())

print(str.expandtabs(2)) #tabsize值为0到7，与tabsize值为8相同

print(str.expandtabs(tabsize=2))

print(str.expandtabs(tabsize=9))

print(str.expandtabs(tabsize=10))

程序运行结果：

i love python

i love python

i love python

i love python

i love python

i love python

i love python

解决查找指定字符(子字符串)位置问题：

## 15.0 find()函数

## 描述：查找字符串中指定的子字符串sub第一次出现的位置，可以规定字符串的索引查找范围。若无则返回 -1。

语法：str.find(sub,start,end) -> int 返回整数

sum —要索引的子字符串。

start —索引的起始位置。默认值为0。

end —索引的结束位置。默认值为字符串长度len(str)。

[start,end) 不包括end。

程序示例：

str = "i love python"

print(str.find("o")) #索引子字符串"o"

print(str.find("0",4))#索引起始位置为4 索引范围为：ve python

print(str.find("o",4,12))#索引起始位置为4,结束位置为12 索引范围为：ve pytho

print(str.find(""))

print(str.find(" "))#返回第一个空格出现的位置。

print(str.find("k")) #索引子字符串"k",不存在，返回-1

程序运行结果：

3

-1

11

0

1

-1

## 16.0 rfind()函数

## 描述：查找字符串中指定的子字符串sub最后一次出现的位置，可以规定字符串的索引查找范围。若无则返回 -1。

语法：str.rfind(sub,start,end) -> int 返回整数

sum —要索引的子字符串。

start —索引的起始位置。默认值为0。

end —索引的结束位置。默认值为字符串长度len(str)。

[start,end) 不包括end。

注：rfind()函数用法与find()函数相似，rfind()函数返回指定子字符串最后一次出现的位置，find()函数返回指定子字符串第一次出现的位置。

程序示例：

str = "i love python python"

print(str.rfind("o")) #默认索引整个字符串str

print(str.rfind("o",11)) #索引范围为：on python

print(str.rfind("o",0,12)) #索引范围为 i love pytho

print(str.rfind("python")) #返回最后一次出现"python"的位置,即字母"p"最后一次出现的位置。

print(str.rfind("")) #等效于返回字符串str的长度。

print(str.rfind(" "))#返回最后一个空格出现的位置

print(str.rfind("2")) #字符串str中不存在字符"2",返回值为 -1

程序运行结果：

18

18

11

14

20

13

-1

## 17.0 index()函数

## 描述：查找字符串中第一次出现的子字符串的位置，可以规定字符串的索引查找范围[star,end)。若无则会报错。

语法：str.index(sub, start, end)  -> int  返回整数

sub —— 查找的子字符串。

start —— 索引的起始位置，默认为0。

end  —— 索引的结束位置，默认为字符串的长度。

[star,end)

程序示例：

str = "i love python"

print(str.index("o")) #默认索引整个字符串

print(str.index("o",4)) #索引 ve python

print(str.index("o",4,12)) #索引 ve pytho

print(str.index("love")) #索引多个字符

print(str.index("k")) #索引字符串不存在，报错

 程序运行结果：

3

11

11

2

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-90-a880e13b1574> in <module>()

4 print(str.index("o",4,12)) #索引 ve pytho

5 print(str.index("love")) #索引多个字符

----> 6 print(str.index("k"))

ValueError: substring not found

   注：index()函数和find()函数类似，但index()函数没有找到子字符串会报错。

## 18.0 rindex()函数

## 描述：查找字符串中最后一次出现的子字符串的位置，可以规定字符串的索引查找范围[star,end)，若无则会报错。

语法：str.rindex(sub, start, end) -> int   返回整数。

sub —— 查找的子字符串。

start —— 索引的起始位置，默认为0。

end  ——  索引的结束位置，默认为字符串的长度。

[star,end)

注：rindex()函数用法与index()函数相似，rindex()函数返回指定子字符串最后一次出现的位置，index()函数返回指定子字符串第一次出现的位置。

程序示例：

str = "i love python python"

print(str.rindex("p")) #默认索引整个字符串

print(str.rindex("o",5)) #索引的范围为：e python python

print(str.rindex("o",5,13)) #索引范围为：e python

print(str.rindex("python")) #返回最后一次出现"python"的位置,即字母"p"最后一次出现的位置。

print(str.rindex("k")) #索引字符串不存在，报错

len(str),str[5:13]

程序运行结果：

14

18

11

14

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-116-44cc5c1acea1> in <module>()

4 print(str.rindex("o",5,13)) #索引范围为：e python

5 print(str.rindex("python")) #返回最后一次出现"python"的位置,即字母"p"最后一次出现的位置。

----> 6 print(str.rindex("k"))

7 len(str),str[5:13]

ValueError: substring not found

解决格式化输出问题：

## 19.0 format()函数

## 描述：返回一个格式化的字符串。

语法：str.format(\*args, \*\*kwargs) ——> str 返回字符串    [模板字符串].format(逗号分隔的参数)

或  {参数序号:格式控制标记}.format(逗号分隔的参数)

它是通过{}和:来代替%。

模板字符串是一个由槽(用大括号{}来表示)和字符串组成的字符串，用来控制字符串的显示效果。

大括号{}对应着format()中逗号分隔的参数。

format()基本用法 ：

语法：[模板字符串].format(逗号分隔的参数)

 通过位置

程序示例：

print("用法一:")

print("一日之计在于{}".format("晨"))

#如果模板字符串有多个槽{}，并且槽内没有指定的序号，则按照槽{}出现的顺序一一对应format()函数中的不同参数

print("用法二：")

print("{}之计在于{}".format("一日","晨"))

#也可以通过在槽{}中指定format()函数中参数的序号来使用，参数是从0开始编号的。

print("用法三：")

print("{0}之计在于{1}".format("一日","晨"))

print("{1}之计在于{0}".format("晨","一日"))

# 如果想要在模板字符串中输出大括号("{}"),则使用{{ 来表示 { 用 }} 来表示 }

print("用法四：")

print("{0}说得好：{{一{1}之计在于晨 一{2}之计在于春}}".format("俗话","日","年"))

#如果模板字符串中出现的槽{}和format()函数中参数个数不一致，

#则必须要在槽{}中使用序号来指定参数的使用，否者会产生IndexError的错误。

print("用法五：")

print("{0}日之计在于晨, {0}年之计在于春".format("一"))

print("{}日之计在于晨, {}年之计在于春".format("一")) #槽中没有指定参数序号，会报错。

程序运行结果：

用法一:

一日之计在于晨

用法二：

一日之计在于晨

用法三：

一日之计在于晨

一日之计在于晨

用法四：

俗话说得好：{一日之计在于晨 一年之计在于春}

用法五：

一日之计在于晨, 一年之计在于春

---------------------------------------------------------------------------

IndexError Traceback (most recent call last)

<ipython-input-20-5f7b4f063a3b> in <module>()

19 print("用法五：")

20 print("{0}日之计在于晨, {0}年之计在于春".format("一"))

---> 21 print("{}日之计在于晨, {}年之计在于春".format("一")) #槽中没有指定参数序号，会报错。

IndexError: tuple index out of range

 通过关键字参数

程序示例：

print("{year}之计在于{season}".format(year="一年",season="春"))

程序运行结果：

一年之计在于春

通过列表

程序示例：

l = ["一日","晨","一年","春"]

a = ["一日","晨"]

b = ["一年","春"]

print("{0[0]}之计在于{0[1]}, {0[2]}之计在于{0[3]}".format(l))

#format()函数中的参数可以是一个或多个列表，但要注意槽中序号的差异。

print("{0[0]}之计在于{0[1]}, {1[0]}之计在于{1[1]}".format(a,b))

程序运行结果：

一日之计在于晨, 一年之计在于春

一日之计在于晨, 一年之计在于春

通过字典

程序示例：

d1 = {"year":"一年","season":"春"}

print(" {year}之计在于{season}".format(\*\*d1))

d2 = {"time":["一日","一年"],"season":["晨","春"]}

print("{time[0]}之计在于{season[0]},{time[1]}之计在于{season[1]}".format(\*\*d2))

程序运行结果：

一年之计在于春

一日之计在于晨,一年之计在于春

format()函数格式控制方法：

语法：{参数序号:格式控制标记}.format(逗号分隔的参数)

槽中格式控制标记规则

       ：            [填充]    [对齐]    [宽度]                [,]               [.精度]             [类型]

引导符号 要填充的单个字符

< 左对齐

> 右对齐

^ 居中对齐

设定槽的输出宽度

数字的千位分隔符

适用于整数和浮点数

用于控制浮点数小数部分

的精度(长度)

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,%

[填充]——只能填充单个字符。填充字符时，一定要选择对齐方式，否者会有 Invalid format specifier 的报错

[对齐]——默认对齐方式为左对齐。

[宽度]——指的是槽的输出宽度，当槽的实际宽度比槽的设定宽度大，则输出槽的实际宽度，否者按设定的宽度输出。

[类型]——解析如下表：

整数和浮点数类型的输出格式

类型 描述 类型 描述

b 输出整数的二进制形式 e 输出浮点数对应的小写字母e的指数形式

c 输出整数对应的Unicode字符 E 输出浮点数对应的大写字母E的指数形式

d 输出整数的十进制形式 f 输出标准浮点数形式

o 输出整数的八进制形式 % 输出浮点数的百分比形式

x 输出整数的小写十六进制形式

X 输出整数的大写写十六进制形式

格式控制标记规则:[填充], [对齐], [宽度], [,], [.精度], [类型]  这六个规则是可以任意组合使用的，但要按以上顺序使用。不过一般可以分为两组。

一组为：[填充] [对齐] [宽度]   主要用于规范字符串的显示格式。

程序示例：

s = "厉害了，我的国"

print("{0:30}".format(s)) #默认为左对齐，宽度为30,序号0可省

print("{0:>30}".format(s)) #右对齐

print("{0:\*>30}".format(s)) #填充\*字符，右对齐

print("{0:\*^30}".format(s)) #居中对齐

print("{0:\*30}".format(s)) #填充字符时，没有选择对齐方式，会有 Invalid format specifier 的报错

程序运行结果：

厉害了，我的国

厉害了，我的国

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*厉害了，我的国

\*\*\*\*\*\*\*\*\*\*\*厉害了，我的国\*\*\*\*\*\*\*\*\*\*\*\*

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-86-959132fb938a> in <module>()

4 print("{0:\*>30}".format(s)) #填充\*字符，右对齐

5 print("{0:\*^30}".format(s)) #居中对齐

----> 6 print("{0:\*30}".format(s)) #填充字符时，没有选择对齐方式，会有 Invalid format specifier 的报错

ValueError: Invalid format specifier

格式控制标记规则也可以用变量来表示，即用槽{}来指定对应的格式控制标记规则。

程序示例：

s = "厉害了，我的国"

a = "\*"

b = "^"

c = 30

d = ["<","^",">"]

print("{0:{1}{2}{3}}".format(s,a,b,c)) #居中对齐，宽度为30，填充字符为\*

print("{0:{1}{2[2]}{3}}".format(s,a,d,c))#右对齐，用列表选择对齐方式。

print("{0:{1}{2[0]}{3}}".format(s,a,d,c))

  程序运行结果：

\*\*\*\*\*\*\*\*\*\*\*厉害了，我的国\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*厉害了，我的国

厉害了，我的国\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 另一组为：[,]  [.精度]  [类型]   主要于规范数字的输出格式和控制字符串的输出长度。

程序示例：

print("{0:,}".format(123456789)) # 千位分隔符

print("{:,.3}".format(123456.123)) #保留3为有效数字

print("{:.3}".format("厉害了，我的国")) # 输出前三个字符

#输出整数类型的格式化规则：

#print("输出整数的二进制形式:{0:b}\n输出整数对应的Unicode字符:{0:c}\n输出整数的十进制形式:{0:d}\n输出整数的八进制形式:{0:o}\n输出整数的小写十六进制形式:{0:x}\n输出整数的大写写十六进制形式:{0:X}".format(123456))

# 等效于一下语句：

print("输出整数的二进制形式: {:b}".format(123456))

print("输出整数对应的Unicode字符: {:c}".format(123456))

print("输出整数的十进制形式: {:d}".format(123456))

print("输出整数的八进制形式: {:o}".format(123456))

print("输出整数的小写十六进制形式: {:x}".format(123456))

print("输出整数的大写写十六进制形式: {:X}".format(123456))

#输出浮点数类型的格式化规则：

print("输出浮点数对应的小写字母e的指数形式: {:e}".format(123456.123456))

print("输出浮点数对应的大写字母E的指数形式: {:E}".format(123456.123456))

print("输出标准浮点数形式: {:f}".format(123456.123456))

print("输出浮点数的百分比形式: {:%}".format(123456.123456))

#对比 [.精度]可以控制小数部分的输出长度

print("输出浮点数对应的小写字母e的指数形式: {:.3e}".format(123456.123456))

print("输出浮点数对应的大写字母E的指数形式: {:.3E}".format(123456.123456))

print("输出标准浮点数形式: {:.3f}".format(123456.123456))

print("输出浮点数的百分比形式: {:.3%}".format(123456.123456))

程序运行结果：

123,456,789

1.23e+05

厉害了

输出整数的二进制形式: 11110001001000000

输出整数对应的Unicode字符: ?

输出整数的十进制形式: 123456

输出整数的八进制形式: 361100

输出整数的小写十六进制形式: 1e240

输出整数的大写写十六进制形式: 1E240

输出浮点数对应的小写字母e的指数形式: 1.234561e+05

输出浮点数对应的大写字母E的指数形式: 1.234561E+05

输出标准浮点数形式: 123456.123456

输出浮点数的百分比形式: 12345612.345600%

输出浮点数对应的小写字母e的指数形式: 1.235e+05

输出浮点数对应的大写字母E的指数形式: 1.235E+05

输出标准浮点数形式: 123456.123

输出浮点数的百分比形式: 12345612.346%

综合运用实例：

程序示例：

print("{:\*^30,.2f}".format(123456.1235))#用\*字符填充，居中对齐，宽度为30，千位分隔符，保留小数点后两位，输出标准浮点数

s = "一年有:"

a = 365.0000

b = "天"

print("{0:\*>8}{1:,.1f}{2:\*<5}".format(s,a,b))

程序运行结果：

\*\*\*\*\*\*\*\*\*\*123,456.12\*\*\*\*\*\*\*\*\*\*

\*\*\*\*一年有:365.0天\*\*\*\*

## 20.0 format\_map()函数

## 描述：返回字符串的格式化版本。

语法：str.format\_map(mapping)  -> str 返回字符串

程序示例：

student = {"name":["张三","李四"],"idcard":[1,2]}

print("我的名字是{name[0]}，我的学号是{idcard[0]}".format\_map(student))

print("我的名字是{name[0]}，我的学号是{idcard[1]}".format\_map(student))

print(["我的名字是{}，我的学号是{}".format(\*x) for x in zip(student["name"],student["idcard"])]) #以列表的形式输出

print("我的名字是{}，我的学号是{}".format(\*x) for x in zip(student["name"],student["idcard"]))

print(["我的名字是{}，我的学号是{}".format(\*x) for x in zip(\*map(student.get,["name","idcard"]))]) #以列表的形式输出

print("我的名字是{}，我的学号是{}".format(\*x) for x in zip(\*map(student.get,["name","idcard"])))

for i in range(len(student)):

print("我的名字是{{name[0]}}，我的学号是{{idcard[0]}}".format(i).format\_map(student))

# {{ }} 等效于{ }

程序运行结果：

我的名字是张三，我的学号是1

我的名字是张三，我的学号是2

['我的名字是张三，我的学号是1', '我的名字是李四，我的学号是2']

<generator object <genexpr> at 0x0000018F5E1CFF68>

['我的名字是张三，我的学号是1', '我的名字是李四，我的学号是2']

<generator object <genexpr> at 0x0000018F5E1CF200>

我的名字是张三，我的学号是1

我的名字是张三，我的学号是1

解决判断问题(返回bool类型)

## 21.0 endswith()函数

## 描述：判断字符串是否以指定字符或子字符串结尾。

语法：str.endswith("suffix", start, end) 或

str[start,end].endswith("suffix")    用于判断字符串中某段字符串是否以指定字符或子字符串结尾。

—> bool    返回值为布尔类型（True,False）

suffix — 后缀，可以是单个字符，也可以是字符串，还可以是元组（"suffix"中的引号要省略，常用于判断文件类型）。

start —索引字符串的起始位置。

end — 索引字符串的结束位置。

str.endswith(suffix)  star默认为0，end默认为字符串的长度len(str)

  注意：空字符的情况。返回值通常为True

程序示例：

str = "i love python"

print("1:",str.endswith("n"))

print("2:",str.endswith("python"))

print("3:",str.endswith("n",0,6))# 索引 i love 是否以“n”结尾。

print("4:",str.endswith("")) #空字符

print("5:",str[0:6].endswith("n")) # 只索引 i love

print("6:",str[0:6].endswith("e"))

print("7:",str[0:6].endswith(""))

print("8:",str.endswith(("n","z")))#遍历元组的元素，存在即返回True，否者返回False

print("9:",str.endswith(("k","m")))

#元组案例

file = "python.txt"

if file.endswith("txt"):

print("该文件是文本文件")

elif file.endswith(("AVI","WMV","RM")):

print("该文件为视频文件")

else:

print("文件格式未知")

程序运行结果：

1: True

2: True

3: False

4: True

5: False

6: True

7: True

8: True

9: False

该文件是文本文件

## 22.0 startswith()函数

## 描述：判断字符串是否以指定字符或子字符串开头。

语法：str.endswith("suffix", start, end) 或

str[start,end].endswith("suffix")    用于判断字符串中某段字符串是否以指定字符或子字符串结尾。

—> bool    返回值为布尔类型（True,False）

suffix — 后缀，可以是单个字符，也可以是字符串，还可以是元组（"suffix"中的引号要省略）。

start —索引字符串的起始位置。

end — 索引字符串的结束位置。

str.endswith(suffix)  star默认为0，end默认为字符串的长度减一（len(str)-1）。

注意：空字符的情况。返回值通常也为True

程序示例：

str = "hello,i love python"

print("1:",str.startswith("h"))

print("2:",str.startswith("l",2,10))# 索引 llo,i lo 是否以“n”结尾。

print("3:",str.startswith("")) #空字符

print("4:",str[0:6].startswith("h")) # 只索引 hello,

print("5:",str[0:6].startswith("e"))

print("6:",str[0:6].startswith(""))

print("7:",str.startswith(("h","z")))#遍历元组的元素，存在即返回True，否者返回False

print("8:",str.startswith(("k","m")))

程序运行结果：

1: True

2: True

3: True

4: True

5: False

6: True

7: True

8: False

## 23.0 isalnum()函数

## 描述：检测字符串是否由字母和数字组成。

语法：str.isalnum()  -> bool  返回值为布尔类型（True,False）

str中至少有一个字符且所有字符都是字母或数字则返回 True,否则返回 False

程序示例：

str1 = "i love python 666" #有空格，不全是字母或数字

str2 = "python" #全为字母

str3 = "123" #全为数字

str4 = "python666" #字母和数字的组合

print(str1.isalnum())

print(str2.isalnum())

print(str3.isalnum())

print(str4.isalnum())

程序运行结果：

False

True

True

True

## 24.0 isalpha()函数

## 描述：检测字符串是否只由字母组成。

语法：str.isalpha()  -> bool  返回值为布尔类型（True,False）

字符串中至少有一个字符且所有字符都是字母则返回 True,否则返回 False。

程序示例：

str1 = "python" #全为字母

str2 = " python" #存在空格

str3 = "123" #全为数字

str4 = "python666" #字母和数字的组合

print(str1.isalpha())

print(str2.isalpha())

print(str3.isalpha())

print(str4.isalpha()

程序运行结果：

True

False

False

False

## 25.0 isdecimal()函数

## 描述：检查字符串是否只包含十进制字符。该方法只存在于unicode对象中。

注意:定义一个十进制字符串，只需要在字符串前添加前缀 'u' 即可。

语法： str.isdecimal()  -> bool  返回值为布尔类型（True,False）

字符串中若只包含十进制字符返回True，否则返回False。

程序示例：

str1 = u"123456"

str2 = u"python666"

str3 = "123456"

str4 = "python666"

print(str1.isdecimal())

print(str2.isdecimal())

print(str3.isdecimal())

print(str4.isdecimal())

程序运行结果：

True

False

True

False

## 26.0 isdigit()函数

## 描述：检测字符串是否只由数字组成.

语法： str.isdigit()  -> bool  返回值为布尔类型（True,False）

字符串中至少有一个字符且所有字符都是数字则返回 True,否则返回 False。

注：能判断“①”，不能判断中文数字。但 isnumeric() 函数可以。

程序示例：

str1 = "python" #全为字母

str2 = " python" #存在空格

str3 = "123" #全为数字

str4 = "python666" #字母和数字的组合

str5 = "一二三四五六七" #中文数字输出False

str6 = "①"

print(str1.isdigit())

print(str2.isdigit())

print(str3.isdigit())

print(str4.isdigit())

print(str5.isdigit())

print(str6.isdigit())

程序运行结果：

False

False

True

False

False

True

## 27.0 isidentifier()函数

## 描述：判断str是否是有效的标识符。str为符合命名规则的变量，保留标识符则返回True,否者返回False。

语法：str.isidentifier()  -> bool  返回值为布尔类型（True,False）

程序示例：

str1 = "123" #变量名为123

str2 = "def" #变量名为保留字

str3 = "\_123" #变量名有下划线开头

str4 = "student"#变量名由字母开端

print(str1.isidentifier())

print(str2.isidentifier())

print(str3.isidentifier())

print(str4.isidentifier())

程序运行结果：

False

True

True

True

## 28.0  islower()函数

## 描述：检测字符串中的字母是否全由小写字母组成。（字符串中可包含非字母字符）

语法：str.islower()  -> bool   返回值为布尔类型（True,False）

字符串中包含至少一个区分大小写的字符，且所有这些区分大小写的字符都是小写，则返回 True，否则返回 False。

程序示例：

str1 = "i love python" #字符串中的字母全为小写

str2 = "我爱python！" #字符串中的字母全为小写，也存在非字母的字符

str3 = "I love python" #字符串中有大写字符

print(str1.islower())

print(str2.islower())

print(str3.islower())

程序运行结果：

True

True

False

## 29.0 isupper()函数

## 描述：检测字符串中的字母是否全由大写字母组成。（字符串中可包含非字母字符）。

语法：str.isupper()  -> bool  返回值为布尔类型（True,False）

字符串中包含至少一个区分大小写的字符，且所有这些区分大小写的字符都是大写，则返回 True，否则返回 False。

程序示例：

str1 = "I LOVE PYTHON" #全为大写字母

str2 = "i LOVE PYTHON" #存在小写字母

str3 = "我爱PYTHON" #存在非字母的字符

print(str1.isupper())

print(str2.isupper())

print(str3.isupper())

程序运行结果：

True

False

True

## 30.0 isnumeric()函数

## 描述：测字符串是否只由数字组成。这种方法是只适用于unicode对象。

注：把一个字符串定义为Unicode，只需要在字符串前添加 前缀 'u'

语法：str.isnumeric()  -> bool  返回值为布尔类型（True,False）

字符串中只包含数字字符，则返回 True，否则返回 False。

与isdigit()函数类似，但isnumeric()函数可以判断中文数字，功能更强大。

程序示例：

str1 = u"123456" #全为数字

str2 = "123456"

str3 = "python666" #字母数字组合

str4 = "一二三四五六" #中文数字

str5 = "①"

print(str1.isnumeric())

print(str2.isnumeric())

print(str3.isnumeric())

print(str4.isnumeric())

print(str5.isnumeric())

程序运行结果：

True

True

False

True

True

## 31.0 isprintable()函数

## 描述：判断字符串中是否有打印后不可见的内容。如：\n \t  等字符。

语法：  str.isprintable()  -> bool 返回值为布尔类型（True,False）

若字符串中不存在\n \t 等不可见的内容，则返回True,否者返回False。

程序示例：

str1 = "i love python" #不存在用print()打印后不可见的内容

str2 = "i love python \n" #存在用print()打印后不可见的内容 \n

str3 = "i love \t python"

print(str1.isprintable())

print(str2.isprintable())

print(str3.isprintable())

程序运行结果：

True

False

False

## 32.0 isspace()函数

## 描述： 检测字符串是否只由空格组成。

语法：str.isspace()  -> bool   返回值为布尔类型（True,False）

若字符串中只包含空格，则返回 True，否则返回 False。

程序示例：

str1 = " "#空格

str2 = "i love python"

print(str1.isspace())

print(str2.isspace())

print(str2[1].isspace()) #字符串str2 的第二个字符为空格

程序运行结果：

True

False

True

## 33.0 istitle()函数

## 描述：检测判断字符串中所有单词的首字母是否为大写，且其它字母是否为小写，字符串中可以存在其它非字母的字符。

语法：str.istitle()  -> bool  返回值为布尔类型（True,False）

若字符串中所有单词的首字母为大写，且其它字母为小写，则返回 True，否则返回 False.

程序示例：

str1 = "I Love Python" #各单词的首字母均为大写，其余字母为小写

str2 = "I love python"

str3 = "I LOVE PYTHON"

str4 = "我爱Python" #存在其它非字母字符，

print(str1.istitle())

print(str2.istitle())

print(str3.istitle())

print(str4.istitle())

程序运行结果：

True

False

False

True

解决字符串两端：

## 34.0  strip()函数

## 描述：从字符串str中去掉在其左右两边chars中列出的字符。

注：chars传入的是一个字符数组，编译器去除两端所有相应的字符，直到出现第一个在chars中不匹配的字符。详看示例。

语法： str.strip(chars)  -> str 返回一个新的字符串

chars —— 要去除的字符  默认为空格或换行符。

程序示例：

str = "123456789321"

print(str.strip("123")) #除去123，对于字符串str右边，1 在 "123"中去除，2在"123"中去除，同理3也去除，但9不在"123"中,去除结束

str1 = "my name is ymyyyy"

print(str1.strip("my")) #"my" 无空格，字母n前的空格不去掉。

print(str1.strip("my ")) #"my "有空格，所以str1左边字母n前的空格被去掉。

程序运行结果：

456789

name is

name is

## 35.0 lstrip()函数

## 描述：从字符串str中去掉在其左边chars中列出的字符。

注：chars传入的是一个字符数组，编译器去除两端所有相应的字符，直到出现第一个在chars中不匹配的字符。详看示例。

语法：str.lstrip(chars)  -> str 返回一个新的字符串

chars —— 要去除的字符  默认为空格或换行符。

程序示例：

str1 = "bacjabck123kluabc"

print(str1.lstrip("abc")) #只去掉左边的"bac",左边第4个字符j不在"abc"中，去除结束

str2 = "12578asdfgh12"

print(str2.lstrip("12")) #只去掉字符串str2左边的 12

程序运行结果：

jabck123kluabc

578asdfgh12

## 36. rstrip()函数

## 描述：从字符串str中去掉在其右边chars中列出的字符。

注：chars传入的是一个字符数组，编译器去除两端所有相应的字符，直到出现第一个在chars中不匹配的字符。详看示例。

语法：str.rstrip(chars)  -> str 返回一个新的字符串

chars —— 要去除的字符  默认为空格或换行符。

程序示例：

str1 = "abcjabck123kluabcca"

print(str1.rstrip("abc")) #只去掉右边的"abcca",右边第6个字符u不在"abc"中，去除结束

str2 = "12578asdfgh11112"

print(str2.rstrip("12")) #只去掉字符串str2右边的 11112

程序运行结果：

abcjabck123klu

12578asdfgh

解决 制表 翻译 问题：

## 37.0 maketrans()函数

## 描述：制作翻译表，删除表，常与translate()函数连用。 即：返回用于str.translate（）函数翻译的的转换表。

语法：maketrans(x, y=None, z=None, /)  返回可用于str.translate()函数的转换表

str.maketrans(x,y,z) #返回的是Unicode映射的字典。

bytes.maketrans(x,y) #返回的是bytes类型.

bytearray.maketrans(x,y) #返回的是bytes类型。

如果只有一个参数x,它必须是一个字典且为Unicode形式的映射。

如果有两个参数x和y，它们必须是长度相等的字符串，并且在结果映射中，x中的每个字符都将映射到y中相同位置的字符(Unicode形式的映射)。

如果有三个参数x,y和z.  x和y用法同上，z为指定要删除的字符串，其结果中的字符将一一映射为：None。

bytes.maketrans(x,y) 和  bytearray.maketrans(x,y) 必须要有x和y两个参数。

注：z的长度可以和x和y不同。

str.maketrans(x,y,z)形式：

程序示例：

s = "123456789"

#只有参数x，且x为字典。

map1 = str.maketrans({"1":"a","2":"b","3":"c"})

#单字符"1" "2" "3"对应的Unicode编码值分别为：49，50,51

#制作翻译表,将 字符串s 中的单字符"1"替换为单字符"a"，单字符"2"替换为单字符"b",一一对应。

print(map1,type(map1),ord("1"),ord("2"),ord("3")) #map1返回的是一个Unicode形式映射的字典

#只有参数x和参数y

map2 = str.maketrans("123","abc")

#单字符"1" "2" "3"对应的Unicode编码值如上，单字符"a" "b" "c"对应的Unicode编码值分别为：97，98,99

#制作翻译表,将字符串s中的单字符"1"替换为单字符"a"，单字符"2"替换为单字符"b",一一对应

print(map2,type(map2),ord("a"),ord("b"),ord("c"))#map2返回的也是一个Unicode形式映射的字典

#有x,y,z三个参数

map3 = str.maketrans("123","abc","56k")

#字符串"123"和"abc"含义如上。字符串"567"为 字符串s 要删除的字符,即制作删除表。

#单字符"5" "6" "k"对应的Unicode编码值分别为：53,54,107

print(map3,type(map3),ord("5"),ord("6"),ord("k")) #map3返回的也是一个Unicode形式映射的字典

程序运行结果：

49: 'a', 50: 'b', 51: 'c'} <class 'dict'> 49 50 51

{49: 97, 50: 98, 51: 99} <class 'dict'> 97 98 99

{49: 97, 50: 98, 51: 99, 53: None, 54: None, 107: None} <class 'dict'> 53 54 107

In [6]:

bytes.maketrans(x,y)形式：

程序示例：

map4 = bytes.maketrans(b"123",b"abc")

print(type(b"123"),type(b"abc"),type(map4),map4)

程序运行结果：

<class 'bytes'> <class 'bytes'> <class 'bytes'>

b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()\*+,-./0abc456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^\_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'

bytearray.maketrans(x,y)形式：

程序示例：

map5 = bytearray.maketrans(b"123",b"abc")

print(type(b"123"),type(b"abc"),type(map5),map5)

程序运行结果：

<class 'bytes'> <class 'bytes'> <class 'bytes'>

b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()\*+,-./0abc456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^\_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'

## 38.0 translate()函数

## 描述：过滤(删除)，翻译字符串。即根据maketrans()函数给出的字符映射转换表来转换字符串中的字符。

注：translate()函数是先过滤(删除)，再根据maketrans()函数返回的转换表来翻译。

语法：str.translate(table)  -> str 返回一个字符串

str.translate(table)

bytes.translate(table, deletechars)

bytearray.translate(table, deletechars)

table —— 转换表，转换表是通过maketrans()函数转换而来。

deletechars —— 字符串中要过滤(删除)的字符。

程序示例：

s = "123456789abc"

s1 = b"123456789abc"

#只有参数x，且x为字典。

map1 = str.maketrans({"1":"a","2":"b","3":"c"})

#单字符"1" "2" "3"对应的Unicode编码值分别为：49，50,51

#制作翻译表,将 字符串s 中的单字符"1"替换为单字符"a"，单字符"2"替换为单字符"b",一一对应。

print(map1,type(map1),ord("1"),ord("2"),ord("3")) #map1返回的是一个Unicode形式映射的字典

#只有参数x和参数y

map2 = str.maketrans("123","abc")

#单字符"1" "2" "3"对应的Unicode编码值如上，单字符"a" "b" "c"对应的Unicode编码值分别为：97，98,99

#制作翻译表,将字符串s中的单字符"1"替换为单字符"a"，单字符"2"替换为单字符"b",一一对应

print(map2,type(map2),ord("a"),ord("b"),ord("c"))#map2返回的也是一个Unicode形式映射的字典

#有x,y,z三个参数

map3 = str.maketrans("123","abc","56k")

#字符串"123"和"abc"含义如上。字符串"567"为 字符串s 要删除的字符,即制作删除表。

#单字符"5" "6" "k"对应的Unicode编码值分别为：53,54,107

print(map3,type(map3),ord("5"),ord("6"),ord("k")) #map3返回的也是一个Unicode形式映射的字典

map4 = bytes.maketrans(b"123",b"abc")

print(type(b"123"),type(b"abc"),type(map4),map4)

print("\n")

map5 = bytearray.maketrans(b"123",b"abc")

print(type(b"123"),type(b"abc"),type(map5),map5)

print("\n\n")

print(s.translate(map1)) #将 字符串s 中的"123" 分别替换为 "abc"

print(s.translate(map2)) #将 字符串s 中的"123" 分别替换为 "abc"

print(s.translate(map3)) #先将 字符串s 中的字符 "5","6" ,"k"过滤(删除)，再将 字符串s 中的"123" 分别翻译替换为 "abc"

print(s1.translate(map4))

print(s1.translate(map5))

print(s1.translate(map4,b"78b")) #先将s1中的b"78b" 过滤(删除)，再将 s1 中的b"123" 分别翻译替换为 b"abc"

print(s1.translate(map5,b"9")) #先将s1中的b"9" 过滤(删除)，再将 s1 中的b"123" 分别翻译替换为 b"abc"

程序运行结果：

{49: 'a', 50: 'b', 51: 'c'} <class 'dict'> 49 50 51

{49: 97, 50: 98, 51: 99} <class 'dict'> 97 98 99

{49: 97, 50: 98, 51: 99, 53: None, 54: None, 107: None} <class 'dict'> 53 54 107

<class 'bytes'> <class 'bytes'> <class 'bytes'> b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()\*+,-./0abc456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^\_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'

<class 'bytes'> <class 'bytes'> <class 'bytes'> b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()\*+,-./0abc456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^\_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff'

abc456789abc

abc456789abc

abc4789abc

b'abc456789abc'

b'abc456789abc'

b'abc4569ac'

b'abc45678abc'

解决分割字符串问题：

## 39.0 partition()函数：

## 描述：根据指定的分隔符(sep)将字符串进行分割。从字符串左边开始索引分隔符sep,索引到则停止索引。

语法： str.partition(sep) -> (head, sep, tail)  返回一个三元元组，head:分隔符sep前的字符串，sep:分隔符本身，tail:分隔符sep后的字符串。

sep —— 指定的分隔符。

如果字符串包含指定的分隔符sep，则返回一个三元元组，第一个为分隔符sep左边的子字符串，第二个为分隔符sep本身，第三个为分隔符sep右边的子字符串。

如果字符串不包含指定的分隔符sep,仍然返回一个三元元组，第一个元素为字符串本身，第二第三个元素为空字符串

程序示例：

str = "https://www.baidu.com/"

print(str.partition("://")) #字符串str中存在sep"://"

print(str.partition(",")) #字符串str中不存在sep",",返回了两个空字符串。

print(str.partition(".")) #字符串str中存在两个"." 但索引到www后的"." 停止索引。

print(type(str.partition("://"))) #返回的是tuple类型， 即元组类型

程序运行结果：

('https', '://', 'www.baidu.com/')

('https://www.baidu.com/', '', '')

('https://www', '.', 'baidu.com/')

<class 'tuple'>

## 40.0  rpartition()函数

## 描述：根据指定的分隔符(sep)将字符串进行分割。从字符串右边(末尾)开始索引分隔符sep,索引到则停止索引。

语法：str.rpartition(sep)  -> (head, sep, tail) 返回一个三元元组，head:分隔符sep前的字符串，sep:分隔符本身，tail:分隔符sep后的字符串。

sep —— 指定的分隔符。

如果字符串包含指定的分隔符sep，则返回一个三元元组，第一个为分隔符sep左边的子字符串，第二个为分隔符sep本身，第三个为分隔符sep右边的子字符串。

如果字符串不包含指定的分隔符sep,仍然返回一个三元元组，第一个元素为字符串本身，第二第三个元素为空字符串。

注：rpartition()函数与partition()函数用法相似，rpartition()函数从右边(末尾)开始索引，partition()函数从左边开始索引。

程序示例：

str = "https://www.baidu.com/"

print(str.rpartition("://")) #字符串str中存在sep"://"

print(str.rpartition(",")) #字符串str中不存在sep",",返回了两个空字符串。

print(str.rpartition(".")) #字符串str中存在两个"." 但索引是从右边(末尾)开始，索引到du后的"." 停止索引。

print(type(str.partition("://"))) #返回的也是tuple类型， 即元组类型

程序运行结果：

('https', '://', 'www.baidu.com/')

('', '', 'https://www.baidu.com/')

('https://www.baidu', '.', 'com/')

<class 'tuple'>

## 41.0  split()函数

## 描述：拆分字符串。通过指定分隔符sep对字符串进行分割，并返回分割后的字符串列表。

语法： str.split(sep=None, maxsplit=-1)  -> list of strings  返回 字符串列表  或str.split(sep=None, maxsplit=-1)[n]

sep —— 分隔符，默认为空格,但不能为空即(")。

maxsplit —— 最大分割参数，默认参数为-1。

[n] —— 返回列表中下标为n的元素。列表索引的用法。

程序示例：

str1 = "i love python"

str2 = "https://www.baidu.com"

str3 = "script<i love python>script"

str4 = "i \n love \n python"

print(str1.split()) #默认空格分割。

print(str2.split("."))#以"."为分隔符,maxsplit默认为-1

print(str2.split(".",-1)) #maxsplit为-1

print(str2.split(".",1)) #以"."为分隔符,分割一次。

print(str2.split(".")[1]) #分割后，输出列表中下标为1的元素

print(str3.split("<")[1].split(">")[0])

print(str4.split("\n")) #可用于去掉字符串中的"\n" "\t"等

程序运行结果：

['i', 'love', 'python']

['https://www', 'baidu', 'com']

['https://www', 'baidu', 'com']

['https://www', 'baidu.com']

baidu

i love python

['i ', ' love ', ' python']

## 42.0  rsplit()函数

## 描述：拆分字符串。通过指定分隔符sep对字符串进行分割，并返回分割后的字符串列表,类似于split()函数，只不过 rsplit()函数是从字符串右边(末尾)开始分割。

语法： str.rsplit(sep=None, maxsplit=-1)  -> list of strings  返回 字符串列表  或str.rsplit(sep=None, maxsplit=-1)[n]

sep —— 分隔符，默认为空格,但不能为空即(")。

maxsplit —— 最大分割参数，默认参数为-1。

[n] —— 返回列表中下标为n的元素。列表索引的用法。

程序示例：

str = "https://www.baidu.com"

print(str.rsplit()) #默认空格分割。

print(str.rsplit(".")) #以"."为分隔符,maxsplit默认为-1

print(str.rsplit(".",1)) #以"."为分隔符,分割一次。从字符串右边(末尾)开始分割。

print(str.rsplit(".",1)[1]) #分割后，输出列表中下标为1的元素

程序运行结果：

['https://www.baidu.com']

['https://www', 'baidu', 'com']

['https://www.baidu', 'com']

com

## 43.0 splitlines()函数

## 描述：按照('\n', '\r', \r\n'等)分隔，返回一个包含各行作为元素的列表，默认不包含换行符。

符号 描述

\n 换行符

\r 回车符

\r\n 回车+换行

语法：str.splitlines(keepends) -> list of strings  返回 字符串列表

keepends —— 默认参数为False ，译为 不保留换行符。参数为True , 译为 保留换行符。

程序示例：

s1 = """i

love

python

"""

s2 = "i\nlove\npython\n" #与s1等效。

s3 = "123\n456\r789\r\nabc"

print(s1.splitlines(True))

print(s1.splitlines()) #keepends 默认为False

print(s1.splitlines(False))

print(s2.splitlines())

print(s3.splitlines()) #遇到("\n" "\r" "\r\n"等)即分隔。

程序运行结果：

['i\n', 'love\n', 'python\n']

['i', 'love', 'python']

['i', 'love', 'python']

['i', 'love', 'python']

['123', '456', '789', 'abc']

## 44.0 join()函数

## 描述：将iterable变量的每一个元素后增加一个str字符串。

语法： str.join(iterable)  -> str  返回字符串 即：返回一个以str作为分隔符，将iterable中的各元素合并连接成一个新的字符串。

str——分隔符。可以为空。

iterable—— 要连接的变量 ，可以是 字符串，元组，字典，列表等。

程序示例：

#对列表进行操作

l = ["1","2","3"]

print(" ".join(l)) #以空格为分隔符

print(",".join(l)) #以逗号为分隔符

#对字符串进行操作

str = "python"

print(" ".join(str)) #以空格为分隔符

print(",".join(str)) #以逗号为分隔符

#对字典进行操作

d = {"a":1,"b":2,"c":3}

print(" ".join(d)) #以空格为分隔符

print(",".join(d)) #以逗号为分隔符

#对元组进行操作

s = ("1","2","3")

print(" ".join(s)) #以空格为分隔符

print(",".join(s)) #以逗号为分隔符

程序运行结果：

1 2 3

1,2,3

p y t h o n

p,y,t,h,o,n

b c a

b,c,a

1 2 3

1,2,3

解决替换问题：

## 45.0 replace()函数：

## 描述：返回字符串str的副本，所有old子字符串被替换为new字符串。

语法：str.replace(old, new, count) -> str 返回字符串str的副本

old —— 将被替换的子字符串。

new —— 新子字符串，用于替换old子字符串。

count —— 替换的次数，默认全部替换。

程序示例：

s = "i love python python "

print(s.replace("o","w")) #默认字符串中的全部"o" 全部替换为"w"

print(s.replace("o","w",2)) #只替换掉前两个子字符串 "o"

print(s.replace("python","c++")) #子字符串可以是多个字符。

print(s.replace("python","c++",1))

程序运行结果：

i lwve pythwn pythwn

i lwve pythwn python

i love c++ c++

i love c++ python

————————————————

版权声明：本文为CSDN博主「九天小牛」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_40678222/java/article/details/83032178