Python中的字符串

1> str和bytes的区别

bytes对象只负责以二进制字节序列的形式记录所需记录的对象，至于该对象到底表示什么（比如到底是什么字符）则由相应的编码格式解码所决定

```
In [1]: type(b'hello')
Out[1]: bytes

In [2]: type('hello')
Out[2]: str
```

str 使用encode方法转化为 bytes bytes通过decode转化为str

```
In [3]: str1 = '人生苦短,我用Python!'

In [4]: type(str1)
Out[4]: str

In [5]: b = str1.encode()

In [6]: b
Out[6]:
b'\xe4\xba\xba\xe7\x94\x9f\xe8\x8b\xa6\xe7\x9f\xad,\xe6\x88\
x91\xe7\x94\xa8Python!'

In [7]: type(b)
Out[7]: bytes
```

bytes转换成str:

```
In [8]: b
Out[8]: b'\xe4\xba\xba\xe7\x94\x9f\xe8\x8b\xa6\xe7\x9f\xad,\xe6\x88\x91\xe7\x94\xa8Python!'

In [9]: type(b)
Out[9]: bytes

In [10]: b.decode()
Out[10]: '人生苦短,我用Python!'

In [11]: type(b.decode())
Out[11]: str
```

实际应用中在互联网上是通过二进制进行传输，所以就需要将str转换成bytes进行传输，而在接收中通过decode()解码成我们需要的编码进行处理数据这样不管对方是什么编码而本地是我们使用的编码这样就不会乱码。

2> 字符串常用操作复习

(1) 字符串的format格式化

较之字符串的格式化输出(使用`%`)，字符串format格式化的优势

- 不考虑变量类型

```
In [12]: name = "JunGe"

In [13]: age = 20

In [14]: print("我的名字是: {},今年{}岁。".format(name, age))
我的名字是: JunGe,今年20岁。
```

- 输出变量通过关键字参数指定，变量可以逆序

```
In [15]: name
Out[15]: 'JunGe'

In [16]: age
Out[16]: 20

In [17]: print("我的名字是:{name},今年{age}岁.".format(age=age, name=name))
我的名字是:JunGe,今年20岁.
```

- 变量可以重复使用多次

```
In [18]: name
Out[18]: 'JunGe'

In [19]: age
Out[19]: 20

In [20]: print("我的名字是:{name},今年{age}岁,今年{age}岁,今年{age}岁".format(age=age, name=name))
我的名字是:JunGe,今年20岁,今年20岁,今年20岁
```

(2) 判断字符串是否包含子串常用方法。

如，s = "hello python"，如何判断s中是否包含"python"?

- 法1： in

```
In [1]: s = "hello python"

In [2]: "python" in s
Out[2]: True
```

- 法2：index

```
In [3]: s
Out[3]: 'hello python'

In [4]: s.index("python")
Out[4]: 6
```

- 法3： find

```
In [5]: s
Out[5]: 'hello python'

In [6]: s.find("python")
Out[6]: 6
```

index和find的区别?

```
In [7]: help(s.index)
  Help on built-in function index:

index(...) method of builtins.str instance
    S.index(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.
(END)

In [9]: help(s.find)
  Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int
```

```
19
20      Return the lowest index in S where substring sub is
    found,
21      such that sub is contained within S[start:end].
     Optional
22      arguments start and end are interpreted as in slice
    notation.
23
24      Return -1 on failure.
25  (END)
26
```

若子串存在，find、index函数均可返回子串第一次出现的下标；若子串不存在，index会抛出ValueError，而find仅返回-1。

(3) 常用API函数

函数名	功能	用例
replace	替换字符串，如果替换后的字符是空字符。(删除)	s = "hello world" s.replace("world", "python")
strip	两端去空白，能去除"\n", "\t", " "(空格)	s = "\n\t zixingAI \t\t\n " s.strip()
split	以指定的字符(参数)切割字符串	s = "www.zixinga.com" s.split(".")
join	以指定的字符拼接字符串	li = ['www', 'zixinga', 'com'] ".".join(li)

如有字符串`my_str = hello world zixing and zixingAI`，以下是常见的操作

总结: 字符串拼接的各种方法

- `+`
- join
- "hello ""world"

<1> 查找与统计功能

函数名	函数原型	说明
find	mystr.find(str, start=0, end=len(mystr))	检测 str 是否包含在 mystr中，如果是返回开始的索引值，否则返回-1
index	mystr.index(str, start=0, end=len(mystr))	跟find()方法一样，只不过如果str不在 mystr中会报一个异常.
rfind	mystr.rfind(str, start=0,end=len(mystr))	类似于 find()函数，不过是从右边开始查找.
rindex	mystr.rindex(str, start=0,end=len(mystr))	类似于 index()，不过是从右边开始.
count	mystr.count(str, start=0, end=len(mystr))	返回 str在start和end之间 在mystr里面出现的次数

使用示例


In [1]: my_str = "hello world zixing and zixingAI"
In [2]: my_str.find("zixing")
Out[2]: 12
In [3]: my_str.find("zixing", 0, 10)
Out[3]: -1
In [4]: my_str.index("zixing", 0, 10)
ValueError Traceback (most recent call last)
<ipython-input-4-333b9d04cdd5> in <module>
----> 1 my_str.index("zixing", 0, 10)
ValueError: substring not found
In [5]: my_str.rfind("zixing")
Out[5]: 23
In [6]: my_str.rindex("ZI")
ValueError Traceback (most recent call last)
<ipython-input-6-48e5ce609cba> in <module>
----> 1 my_str.rindex("ZI")
ValueError: substring not found
In [7]: my_str.count("zixing")
Out[7]: 2
In [8]:

<2> 判断功能

函数名	函数原型	说明
isalpha	mystr.isalpha()	如果 mystr 所有字符都是字母则返回 True,否则返回 False
isdigit	mystr.isdigit()	如果 mystr 只包含数字则返回 True 否则返回 False.
isalnum	mystr.isalnum()	如果 mystr 所有字符都是字母或数字则返回 True,否则返回 False
isspace	mystr.isspace()	如果 mystr 中只包含空格，则返回 True，否则返回 False.
startswith	mystr.startswith(obj)	检查字符串是否是以 obj 开头，是则返回 True，否则返回 False
endswith	mystr.endswith(obj)	检查字符串是否以obj结束，如果是返回True,否则返回 False.

使用示例

```
In [17]: alp_str = "abc"

In [18]: num_str = "123"

In [19]: alnu_str = "abc123"

In [20]: alp_str.isalpha()
Out[20]: True

In [21]: num_str.isalpha()
Out[21]: False

In [22]: alnu_str.isalpha()
Out[22]: False

In [23]: alp_str.isdigit()
Out[23]: False

In [24]: num_str.isdigit()
Out[24]: True

In [25]: alnu_str.isdigit()
Out[25]: False

In [26]: alp_str.isalnum()
Out[26]: True

In [27]: num_str.isalnum()
Out[27]: True

In [28]: alnu_str.isalnum()
Out[28]: True

In [29]:
```

```
In [1]: s = "abc123"

In [2]: s.isspace()
Out[2]: False

In [3]: s = ""

In [4]: s.isspace()
Out[4]: False

In [5]: s = " "

In [6]: s.isspace()
Out[6]: True

In [7]: s = "        "

In [8]: s.isspace()
Out[8]: True

In [9]:
```

```
In [9]: my_str = "hello world zixing and zixingAI"

In [10]: my_str.startswith("hello")
Out[10]: True

In [11]: my_str.startswith("Hello")
Out[11]: False

In [12]: my_str.endswith("AI")
Out[12]: True

In [13]: my_str.endswith("ai")
Out[13]: False

In [14]:
```

<3> 分割与合并

函数名	函数原型	说明
split	mystr.split(str, max_split)	以 str 为分隔符切片 mystr，如果 maxsplit有指定值，则仅分隔 maxsplit 个子字符串
partition	mystr.partition(str)	ystr以str分割成三部分,str前, str和str后
rpartition	mystr.rpartition(str)	类似于 partition()函数,不过是从右边开始.
splitlines	mystr.splitlines()	按照行分隔，返回一个包含各行作为元素的列表
join	mystr.join(str)	mystr 中每个字符后面插入str,构造出一个新的字符串

```
In [14]: s = "hello world ha ha"

In [15]: s.split(" ")
Out[15]: ['hello', 'world', 'ha', 'ha']

In [16]: s.split(" ", 2)
Out[16]: ['hello', 'world', 'ha ha']

In [17]:
```

```
In [17]: my_str = "hello world zixing and zixingAI"

In [18]: my_str.partition("zixing")
Out[18]: ('hello world ', 'zixing', ' and zixingAI')

In [19]: my_str.rpartition("zixing")
Out[19]: ('hello world zixing and ', 'zixing', 'AI')

In [20]:
```

```
In [23]: s = "hello\nworld"

In [24]: print(s)
hello
world

In [25]: s.splitlines()
Out[25]: ['hello', 'world']

In [26]:
```

```
In [26]: s = " "

In [27]: li = ["My", "name", "is", "JunGe"]

In [28]: s.join(li)
Out[28]: 'My name is JunGe'

In [29]: s = "_"

In [30]: s.join(li)
Out[30]: 'My_name_is_JunGe'

In [31]:
```

<4> 转换

函数名	函数原型	说明
replace	mystr.replace(str1, str2, count)	把 mystr 中的 str1 替换成 str2, 如果 count 指定，则替换不超过 count 次.
strip	mystr.strip()	删除mystr字符串两端的空白字符
lstrip	mystr.lstrip()	删除 mystr 左边的空白字符
rstrip	mystr.rstrip()	删除字符串末尾的空白字符
capitalize	mystr.capitalize()	把字符串的第一个字符大写
title	mystr.title()	把mystr的每个单词首字母大写
lower	mystr.lower()	转换 mystr 中所有大写字符为小写
upper	mystr.upper()	转换 mystr 中的小写字母为大写
ljust	mystr.ljust(width)	返回一个原字符串左对齐,并使用空格填充至长度 width 的新字符串
rjust	mystr.rjust(width)	返回一个原字符串右对齐,并使用空格填充至长度 width 的新字符串
center	mystr.center(width)	返回一个原字符串居中,并使用空格填充至长度 width 的新字符串

- replace

```
In [1]: s = "hello world ha ha"

In [2]: s.replace("ha", "Ha")
Out[2]: 'hello world Ha Ha'

In [3]: s.replace("ha", "Ha", 1)
Out[3]: 'hello world Ha ha'

In [4]:
```

- strip

```
In [4]: a = "\n\t itcast \t\n"

In [5]: a.strip()
Out[5]: 'itcast'

In [6]:
```

- lstrip

```
In [6]: a = "       hello"

In [7]: a.lstrip()
Out[7]: 'hello'

In [8]: a = "       hello       "

In [9]: a.lstrip()
Out[9]: 'hello       '

In [10]:
```

- rstrip

```
In [12]: a = "       hello       "

In [13]: a.rstrip()
Out[13]: '       hello'

In [14]:
```

- capitalize

```
In [14]: my_str = "hello world zixing and zixingAI"

In [15]: my_str.capitalize()
Out[15]: 'Hello world zixing and zixingai'

In [16]:
```

- title

```
In [16]: my_str = "hello world zixing and zixingAI"

In [17]: my_str.title()
Out[17]: 'Hello World Zixing And Zixingai'

In [18]:
```

- lower

```
In [18]: my_str = "HELLO world zixing and zixingAI"

In [19]: my_str.lower()
Out[19]: 'hello world zixing and zixingai'

In [20]:
```

- upper

```
In [20]: my_str = "HELLO world zixing and zixingAI"

In [21]: my_str.upper()
Out[21]: 'HELLO WORLD ZIXING AND ZIXINGAI'

In [22]:
```

- ljust

```
In [22]: my_str = "hello"

In [23]: my_str.ljust(10)
Out[23]: 'hello     '

In [24]:
```

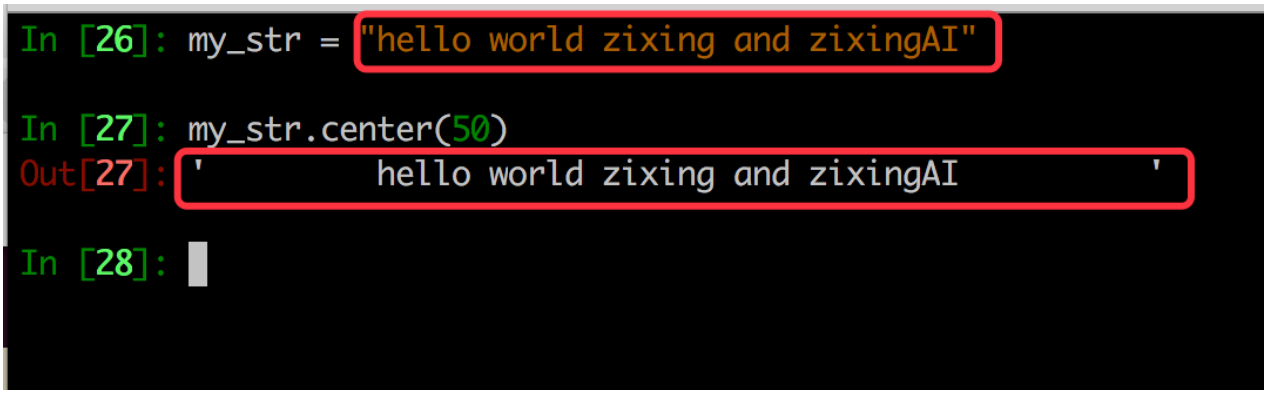
- rjust

```
In [24]: my_str = "hello"

In [25]: my_str.rjust(10)
Out[25]: '     hello'

In [26]:
```

- center


In [26]: my_str = "hello world zixing and zixingAI"
In [27]: my_str.center(50)
Out[27]: '   hello world zixing and zixingAI   '
In [28]: