字符串

关于基础数据类型，你已经学习了int,float,bool，他们都很简单，相信已经跃跃欲试想要加快学习进度了，别急，年轻人，如果你此前并没有什么编程功底，那么字符串的内容将很快让你感受到学习一门编程语言的该有的难度。

你将学习如下知识点

1. 字符串的概念
2. 切片操作
3. 转义字符
4. 格式化操作

你将会接触到很多全新的概念，对于有编程背景的人来说，这些概念早已轻车熟路，但对计算机小白来说，某些概念咋一看，犹如天书，我会尽量用简洁易懂的语言来阐述这些知识，同时也希望你能认真务实的学习。

字符串(str)概念

字符串是python当中最常用的数据类型，我们用它来处理文本内容，字符串是字符的有序集合，可以使用一对单引号或一对双引号，或者3对双引号来创建，python字符串的索引有正向索引和反向索引之分，通过索引你可以随机访问字符串

1. 字符串的创建

字符串是python当中最常用的数据类型，我们用它来处理文本内容，下面的代码演示了3中创建字符串的方法

```
str_1 = 'python'
str_2 = "python"
str_3 = """python"""

print(str_1, type(str_1))
print(str_2, type(str_2))
print(str_3, type(str_3))
```

在pycharm新建一个脚本，本教程如果没有要求在交互式解释器里写代码，那么默认就是在pycharm里编写代码。

上面的代码里，我分别用单引号，双引号，3对双引号创建一个字符串，使用print函数输出这3个变量的内容以及他们的类型，实际输出结果为

```
python <class 'str'>
python <class 'str'>
python <class 'str'>
```

这3种创建字符串的方法，常用的是前两种，在使用print函数时，多个数据可以用逗号分隔。

多个字符串可以使用 + 连接在一起，生成新的字符串，在交互式解释器中跟随我操作

```
>>> a = 'I'
>>> b = 'like'
>>> c = 'python'
>>> d = a + ' ' + b + ' ' + c
>>> d
'I like python'
>>> len(d)
13
```

我定义了三个变量，其类型都是字符串，最后使用 + 将他们连接在一起，在连接时，为了不让单词紧挨着，我在中间加入了' '，也是一个字符串，里面的内容是空格。

使用内置函数len()，可以获取字符串的长度，你又学会了一个新的内置函数。

2. 索引的概念

这是你在学习编程语言时接触到第一个十分重要的专业概念，后面学习列表时，你还会用到它，不理解索引，就没办法学习切片。

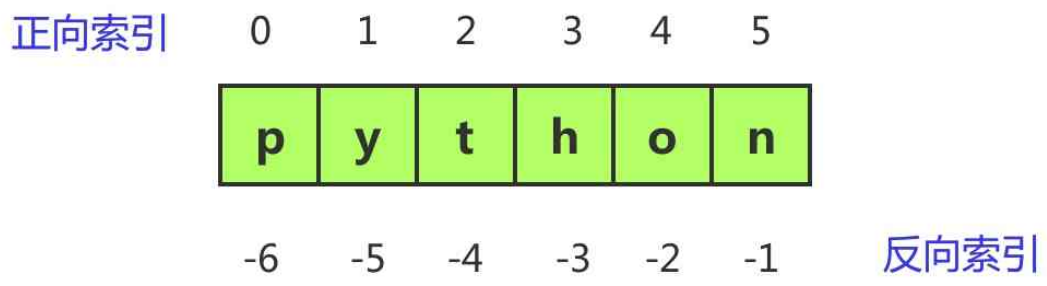
python中，字符串是字符的有序集合。这里，你主要关注有序二字。

```
'python'
'443'
'*&^%$'
```

第一个字符串里有5个字母，第二个字符串里有3个数字，第三个字符串里有5个字符，在编程语言里，这些都统称为字符，所以，字符串是字符的集合。

那么有序是怎么体现的呢？有序意味着，每一个字符都有自己的位置，专业术语叫索引，比如字符串'python'， 我现在问你，这个字符串的第3个字符是什么，你会回答说是字符t,从左向右数，的确是t，这个就是索引。

但与所熟悉的计数方式不同，编程语言里，索引都是从0开始的，因此，t在字符串python中的索引是2



不仅如此，python还支持反向索引，t的反向索引是-4。

你可以通过索引来访问字符串里的某个字符，在编辑器里跟随我操作

```
>>> a = 'python'
>>> a[0]
'p'
>>> a[-2]
'o'
>>> a[1:3]
'yt'
>>> a[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

根据索引访问字符串里的某个字符时，需要使用一对中括号[]，在其中填写索引，如果索引超出了范围，就会引发IndexError，这是初学者非常容易犯的错误。

a[1:3]，表示范围索引1到索引3这个范围内的字符，得到的结果是'yt'， 这就是切片操作，关于切片操作会有专门的文章讲解。

3. 字符串的运算

对于字符串，有以下运算

操作符	描述
+	字符串连接
*	重复字符串
[]	通过索引访问指定索引的字符
[:]	切片操作，截取字符串指定范围
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
%	格式字符串

字符串连接操作，在介绍字符串概念时已经有过讲解，下面在交互式解释器里跟随我操作，学习这些操作符

```
>>> a = 'py'
>>> b = 'python'
>>> a + b
'pypython'
>>> a*3
'pypypy'
>>> b[3]
'h'
>>> b[0:2]
'py'
>>> a in b
True
>>> b not in b
False
```

关于 % ,格式化字符串，会有专门的文章进行讲解。

转义字符

1.什么是转义字符

文章题目虽然用了python转义字符这个词，但转义字符不是python这门语言所独有的，准确的讲，这是一个计算机专业词汇。

在计算机当中，有些字符我们无法手动书写，你可以手写出字母abcd，但你能写出来换行符么？你能看得见换行符么？

具体都有哪些转义字符呢，见下表

转义字符	描述
(在行尾时)	续行符
\	反斜杠符号
'	单引号
"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy代表的字符，例如：\o12代表换行
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

2.转义字符的实际应用

单引号，双引号在字符串里

python当中，定义一个字符串可以使用单引号，比如s = 'abcd'，那么如何在这样的字符串里写一个单引号呢？比如你需要定义一个字符串 it's a book

```
s = 'it's a book'
```

如果你是这样实现的，编辑器就会报错，因为这个字符串是用单引号括起来的，可是中间又出现一个单引号，到底哪两个单引号构成一个字符串呢？这里面就出现了歧义，而计算机最怕的就是歧义。

这种情况下，你就需要使用转义字符了

```
s = 'it\'s a book'
```

单引号的你学会了，双引号的也是相同的道理

```
s = "使用\"创建字符串"
```

如果要在字符串里使用 \ 呢，则需要写成 \

```
s = "换行符是\\n"
print(s)
```

执行代码，输出结果为

```
换行符是\n
```

写文件时使用换行符

```
lst = ['book', 'python', 'good']
```

现在要求你将列表里的单词写入到文件中，每个单词一行，写文件要用write方法，但是这个方法是不会主动添加换行符的，因此我们必须主动加上

```
lst = ['book', 'python', 'good']

with open('data', 'w')as f:
    for word in lst:
        f.write(word + "\n")
```

如果你把代码里的f.write(word + "\n") 修改成f.write(word)，文件里最终只有一行数据

读取文件时，要去掉换行符

读取文件时，不论是是用readline,还是readlines，每一行的末尾的换行符也会被读取，但这个换行符是没有什么作用的，因此需要删除

```
with open('data', 'r')as f:
    for line in f:
        print(line.strip())
```

程序输出结果是

```
book
python
good
```

字符串的strip方法可以移除字符串头尾指定的字符（默认为空格或换行符）或字符序列

如果你把print(line.strip())这行代码改成print(line)，不去除末尾的换行符，程序最终输出结果就会变成

```
book

python

good
```

输出的内容不是紧挨着的，这是因为读取到的line末尾有一个换行符，print输出原本就是换行输出的，再加上line带的换行，就会导致这样的结果。

字符串切片操作

切片操作是使用频率非常高的操作，它的一般模式[start:end)，左闭右开，索引为end的元素不会被截取。

在交互式解释器里跟随我操作

1. 指定开始和结束位置

```
>>> a = 'I like python'
>>> a[2:6]
'like'
>>> a[2:-7]
'like'
```

做切片操作时，你也可以使用反向索引

2. 指定开始位置，不指定结束位置

```
>>> a = 'I like python'
>>> a[2:]
'like python'
```

如果不指定结束位置，那么截取内容就是从开始位置到字符串末尾

3. 指定结束位置，不指定开始位置

```
>>> a = 'I like python'
>>> a[:-7]
'I like'
```

如果不指定开始位置，那么开始位置就是0

4. 切片操作允许索引越界

```
>>> a = 'I like python'
>>> len(a)
13
>>> a[2:19]
'like python'
```

字符串a的长度是13，索引从0开始，最大索引为12，19显然已经超出了合理范围，专业名词叫越界，但程序并没有报错，这是切片操作的一个特点，当索引越界时，切片操作将13作为了结束位置。

5. 指定切片间隔

```
>>> a = '123456789'
>>> a[::2]
'13579'
```

[]操作符内有两个:， 这两个要分开理解，对于第一个，我们将其理解为设置索引，:左右两侧都没有明确写明索引，这就相当于既没有指定开始索引，也没有指定结束索引，因此等价于开始索引从0开始，结束索引就是字符串的末尾。

第二个: 的作用是设置切片的间隔，每隔2个索引做一次切片处理，这样就最终得到了'13579'， 第2个: 设置切片间隔，也可以是负数，表示反向间隔

```
>>> a = '123456789'
>>> a[::-2]
'97531'
```

反向间隔2个索引位置进行切片，就得到了'97531'， 如果想翻转字符串，则应该这样处理

```
>>> a = '123456789'
>>> a[::-1]
'987654321'
```

这段代码表示从索引0到字符串默认进行切片，处理的方式是反向间隔1个索引位置进行切片处理，这样就得到了字符串的翻转结果

字符串格式化

1. % 格式化字符串

在交互式解释器里跟随我操作

```
>>> '我喜欢 %s 色' % '红'
'我喜欢 红 色'
>>> a = '我是%s, 今年%d 岁'
>>> b = a % ('小明', 14)
>>> b
'我是小明, 今年14 岁'
```

你应当注意到，字符串a当中有一些内容用了一些特殊表示形式，%s, %d ，这样做的目的是为了通过格式化字符串来填充这部分内容，以便于生成想要的字符串内容。

python提供了很多字符串格式化符号，用以格式生成不同类型的数据

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
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

使用% 虽然可以格式化，但我并不推荐你用这种方法，因为这样写出来的代码可阅读性较差，更加友好的方式是使用字符串的format方法

2. format

格式化，推荐使用字符串format方法

```
string = "I like {color}".format(color='red')
print(string)

string = "我喜欢 {color},{color}让人安静".format(color='绿色')
print(string)

string = "{course}考了90分,{name}非常高兴"
string = string.format(course="语文", name="李雷")
print(string)
```

字符串里希望被替换的内容，用大括号包裹起来，在format方法的参数里，你需要设置替换的方法，比如在第二个例子中，设置color = '绿色'， 那么字符串里，所有的{color}都会被替换成绿色。

使用format方法，使得代码看起来更加语义明确，需要被填充的地方将由什么数据填充一清二楚，而如果使用% ，你不得不仔细核对每一处替换与数据的对应关系，当字符串有很多处需要替换填充时，这种核对将变成灾难。

3. f-string

f-string 是python3.6加入的一种新技术，这种技术称之为字面量格式化字符串。

```
color = '红色'
string = f'我喜欢{color}'
print(string)

info = {'languge': 'python', 'site': 'http://lenovo.com.cn'}
print(f"我正在学习info['languge'], 使用的教程网址是info['site']")
```

这种技术，会自动将前面的变量内容填充到字符串中以达到格式化字符串的目的。

关于字符串的格式化，我会在进阶教程里继续讲解，目前，作为基础教程学习者，你能掌握并合理运用本篇文章的内容就算合格。

字符串方法

python字符串提供了很多内建方法，你必须掌握这些方法，否则，将无法娴熟的处理字符串。这些方法，暂时不需要你死记硬背，但至少你应该有一些印象，在处理字符串问题时，如果做不到信手拈来，可以查阅资料，寻访百度或是谷歌，下面这这些方法的列表

1. 转换类方法

编号	方法名称	功能描述
1	capitalize()	将字符串的第一个字符转换为大写
2	center	返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格
3	encode	以 encoding 指定的编码格式编码字符串
4	join(seq)	以指定字符串作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
5	len(string)	返回字符串长度
6	ljust(width[, fillchar])	返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格
7	rjust(width[, fillchar])	返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串
8	lower()	转换字符串中所有大写字符为小写
9	upper()	转换字符串中的小写字母为大写
10	lstrip()	截掉字符串左边的空格或指定字符
11	rstrip()	删除字符串字符串末尾的空格
12	split(sep=None, maxsplit=-1)	以 sep为分隔符截取字符串，如果 maxsplit 有指定值，则仅截取 maxsplit+1 个子字符串
	strip([chars])	在字符串上执行 lstrip()和 rstrip()
13	replace(old, new[, count])	将字符串中的 old 替换成 new,如果 max 指定，则替换不超过 count 次
14	splitlines([keepends])	按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符。
15	swapcase()	将字符串中大写转换为小写，小写转换为大写
16	zfill (width)	返回长度为 width 的字符串，原字符串右对齐，前面填充0

capitalize

python字符串的capitalize方法将第一个字符转换为大写，通常是用于处理英文字符串

```
word = 'hello'
print(word.capitalize())    # Hello
```

center

python字符串的center方法返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格

方法定义如下

```
def center(self, width, fillchar=None):
    pass
```

示例代码如下

```
word = 'hello'
print(word.center(9, '*'))  # **hello**
```

encode

python字符串的encode方法可以对字符串进行编码，返回的是bytes类型数据

方法定义如下

```
def encode(self, encoding='utf-8', errors='strict'):
```

默认使用utf-8进行编码

```
word = 'hello world'
byte_word = word.encode(encoding='utf-8')
print(byte_word)   # b'hello world'
print(type(byte_word))  # <class 'bytes'>
```

join

python字符串的join方法用于连接多个字符串,通常作用于列表

```
lst = ['hello', ' ', 'world']
print(''.join(lst))  # hello world
print('*'.join(lst)) # hello* *world
```

len

len函数是内置函数，并不是字符串的方法，放在这里讲是因为它很重要, len函数返回对象的长度

```
word = 'python'
print(len(word))  # 5
```

ljust

python字符串的ljust方法返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格

方法定义

```
def ljust(self, width, fillchar=None):
    pass
```

示例代码

```
word = 'python'
print(word.ljust(9, '*'))   # python***
```

rjust

python字符串的rjust方法返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串

方法定义如下

```
def rjust(self, width, fillchar=None):
    pass
```

示例代码

```
word = 'python'
print(word.rjust(9, '*'))
```

lower

python字符串的lower方法将字符串里所有区分大小写的字符转换为小写

```
word = 'PYTHON'
print(word.lower()) # python
```

upper

python字符串的upper方法将字符串中的小写字母转为大写

```
word = 'hello world'
print(word.upper())  # HELLO WORLD
```

lstrip

python字符串的lstrip方法可以截掉字符串左边的空格或指定字符

```
word = ' python'
print(word.lstrip())    # python
```

默认截取掉空格符，你也可以指定需要被截取的字符

```
word = 'apython'
print(word.lstrip('a'))    # python
```

rstrip

python字符串的rstrip方法删除字符串字符串末尾的空格或指定字符串

```
word = 'python  '
print(word.rstrip())  # python
```

split

python字符串的split方法用于以sep为分隔符截取字符串

方法定义如下

```
def split(self, sep=None, maxsplit=-1):
    pass
```

示例代码

```
word = 'python'
print(word.split('t'))      # ['py', 'hon']
```

split默认以空格做分隔符，maxsplit可以指定最大截取次数

```
word = 'bababababa'
print(word.split('a', 3))  # ['b', 'b', 'b', 'baba']
```

strip

python字符串的strip方法在字符串上执行 lstrip()和 rstrip()， 示例代码如下

```
word = ' python '
print(word.strip()) # python
```

replace

python字符串的replace方法将字符串中的 old 替换成 new,如果 max 指定，则替换不超过 count 次

方法定义如下

```
def replace(self, old, new, count=None):
    pass
```

先来看一个简单的示例

```
word = 'python'
print(word.replace('p', 'cp'))  # cpython
```

你可以指定替换的次数

```
word = 'aaaa'
print(word.replace('a', 'b', 3))  # bbba
```

splitlines

python字符串的splitlines方法按照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，如果为 True，则保留换行符

示例代码

```
word = 'py\rt\nhon'
print(word.splitlines())  # ['py', 't', 'hon']
```

swapcase

python字符串的swapcase方法将字符串中大写转换为小写，小写转换为大写

```
word = 'Python'
print(word.swapcase()) # pYTHON
```

zfill

python字符串的zfill方法返回长度为 width 的字符串，原字符串右对齐，前面填充0

```
word = 'hello world'
print(word.zfill(20))  # 000000000hello world
```

2. 查询类方法

编号	方法名称	功能描述
1	count	返回子串出现的次数
2	find	查找子串sub在字符串中的位置，如果找不到返回-1
3	rfind(sub[, start[, end]])	类似于 find()函数，不过是从右边开始查找
4	index	跟find()方法一样，只不过如果sub不在字符串中会报一个异常
5	rindex(sub[, start[, end]])	类似于 index()，不过是从右边开始

count

python字符串的count方法返回子串sub在字符串中出现的次数

方法定义如下

```
def count(self, sub, start=None, end=None):
```

先来看一个简单的例子

```
word = 'hello world'
print(word.count('o'))   # 2
```

字符串word中有两个子串'o'

需要特别注意的是，count方法可以指定查询的范围

```
word = 'hello world'
count = word.count('o', 5)
print(count)
```

第二个参数可以指定查找的起始索引位置，当我将start设置为5后，就只能找到一个子串'o'， 实践中，你还以同时设置起始位置和结束位置

find

python字符串的find方法用于查找子串sub在字符串中的索引，如果找不到返回-1

方法定义如下

```
def find(self, sub, start=None, end=None):
    pass
```

简单示例

```
word = 'hello world'
print(word.find('world'))  # 6
```

find方法可以指定查询的起始索引和结束索引，指定这其中一个或者全部都指定，该方法将在指定范围内查找

```
word = 'hello world'

print(word.find('world', 3, 9))  # -1
print(word.find('world', 7))  # -1
```

rfind

python字符串的rfind方法功能类似于 find()函数，不过是从右边开始查找

```
word = 'python'
print(word.rfind('thon'))  # 2
```

为什么需要从右边开始查找呢，考虑一下这种情况，目标字符串中包含多个子串，如果使用find方法，那么就只能找到从左侧数第一个子串的位置，如果想找最后一个，就需要使用rfind方法

index

python字符串的index方法功能跟find()方法一样，只是如果sub不在字符串中会报一个异常

方法定义如下

```
def index(self, sub, start=None, end=None):
    pass
```

示例代码

```
word = 'hello world'
print(word.index('world'))  # 6
```

如果要查询的子串不存在，方法会报错

```
word = 'hello world'
print(word.index('worldww'))
```

报错内容为

```
Traceback (most recent call last):
  File "/Users/kwsy/kwsy/coolpython/demo.py", line 2, in <module>
    print(word.index('worldww'))  # 6
ValueError: substring not found
```

子串没有找到

rindex

python字符串的rindex方法功能类似于index，不同之处在于rindex是从右侧开始查找

```
word = 'python'
print(word.rindex('thon'))  # 2
```

3. 验证类方法

编号	方法名称	功能描述
1	startswith(prefix[, start[, end]])	检查字符串是否是以指定子字符串 prefix 开头
2	endswith	检查字符串是否以 suffix 结束
3	isalnum	如果字符串至少有一个字符并且所有字符都是字母或数字则返 回 True,否则返回 False
4	isalpha	如果字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False
5	isdigit	如果字符串只包含数字则返回 True 否则返回 False
6	isnumeric	如果字符串中只包含数字字符，则返回 True，否则返回 False
7	isspace()	如果字符串中只包含空白，则返回 True，否则返回 False.
8	isdecimal()	检查字符串是否只包含十进制字符，如果是返回 true，否则返回 false
9	istitle()	如果字符串是标题化的(见 title())则返回 True，否则返回 False
9	isupper()	如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
10	islower	如果字符串中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False

startswith

python字符串的startswith方法检查字符串是否是以指定子字符串 prefix 开头，方法定义如下

```
def startswith(self, prefix, start=None, end=None):
```

示例代码

```
word = 'python'
print(word.startswith('py'))  # True
```

startswith方法允许你指定起始索引和结束索引

```
word = 'python'
print(word.startswith('th', 2))  # True
```

endswith

python字符串的endswith方法判断字符串是否以suffix结尾， 可以通过beg和 end来指定范围

方法定义如下

```
def endswith(self, suffix, start=None, end=None)
```

示例代码如下

```
word = 'hello world'
print(word.endswith('world'))  # True
```

尽管可以在方法调用时指定开始索引和结束索引，但通常实践中不会用到，下面的例子向你展示如何使用start和end参数

```
word = 'hello world'

print(word.endswith('world', 0, -2))  # False
print(word.endswith('worl', 0, -1))   # True
```

isalnum

python字符串isalnum方法可以判断字符串是否只包含字母和数字，如果字符串不是空串，且只包含字母和数字，该方法返回True,反之返回False

```
word = 'hello23'
print(word.isalnum())  # True

word = 'hello word'
print(word.isalnum())  # False
```

isalpha

python字符串的isalpha方法用于判断字符串是否只包含字母，如果字符串不是空串且都是字母，方法返回True，反之返回False

```
word = 'hello'
print(word.isalpha())  # True

word = 'hello world'  # 包含一个空格
print(word.isalpha())  # False

word = 'hello100'  # 包含数字
print(word.isalpha())  # False
```

isdigit

python字符串的isdigit方法用于判断字符串是否只包含数字

```
word = '123'
print(word.isdigit())  # True
```

isnumeric

python字符串的isnumeric方法可用于判断字符串是否是数字，数字包括Unicode数字，byte数字（单字节），全角数字（双字节），罗马数字

```
print('23'.isnumeric())     # True
print('五十五'.isnumeric())    # True
print('Ⅵ'.isnumeric())    # True
```

isspace

python字符串的isspace方法用于判断字符串是否只包含空白符，空白符包括空格、回车符(\r)、换行符(\n)、水平制表符(\t)、垂直制表符(\v)、换页符(\f)

```
word = ' \t\n'
print(word.isspace())   # True
```

isdecimal

python字符串的isdecimal方法检查字符串是否只包含十进制字符(Unicode数字，，全角数字（双字节）)，如果是返回 true，否则返回 false

```
print('1233'.isdecimal())
```

istitle

python字符串的istitle方法用于判断字符串是否是标题化的字符串,即字符串里每个英文单词的首字母都是大写

```
word = 'Hello Word'
print(word.istitle())  # True
```

isupper

python字符串的isupper方法用于判断字符串里的字母是否都是大写

```
word = 'ABC'
print(word.isupper())  # True
```

如果某个字符本身并不区分大小写，则不影响这个方法的返回结果

islower

python字符串的islower方法可以判断字符串里的字母是否都为小写字母

```
word = 'hello world'
print(word.islower())  # True
```

需要注意的是，如果一个字符本身并不区分大小写，比如上面示例中的空格，那么这个字符不参与比较，不影响方法的返回结果

字符串与int,float,bool相互转换

在前面学习int,float,bool类型数据时，你已经掌握了这3种数据之间的相互转换。现在，你可以使用内置函数str将这3种类型的数据转换为字符串，同样的，也可以用之前学习过的int()，float()，bool()函数将字符串转换为对应的数据类型。

在交互式解释器里跟随我的代码进行操作

1. 字符串与int相互转换

```
>>> a = 100
>>> str_a = str(a)
>>> str_a
'100'
>>> int_a = int(str_a)
>>> int_a
100
>>> int('32.5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '32.5'
```

如果你尝试将一个带小数点的字符串转换成int类型数据，就会报错。

2. 字符串与float相互转换

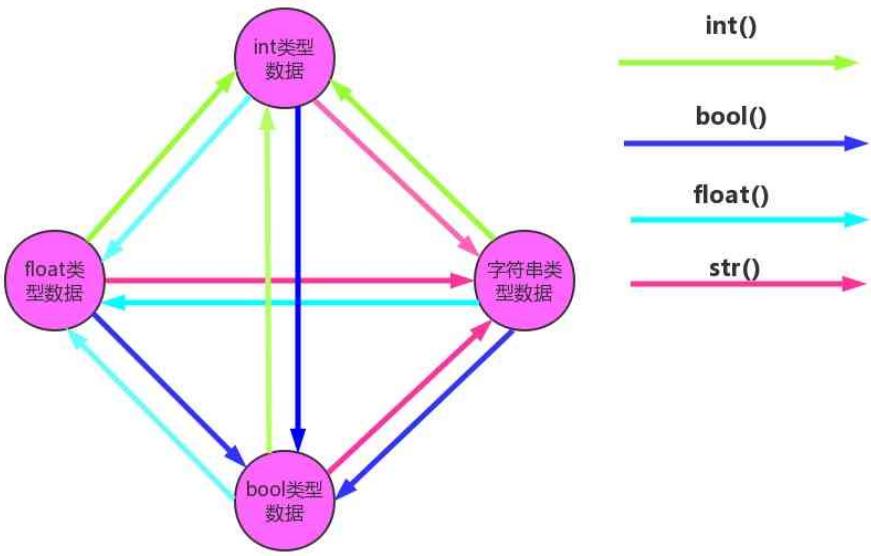
```
>>> float('324')
324.0
>>> float('34.2')
34.2
>>> int(float('55.6'))
55
>>> str(43.5)
'43.5'
```

3. 字符串与bool相互转换

```
>>> str(True)
'True'
>>> str('False')
'False'
>>> bool('True')
True
>>> bool('False')
True
>>> bool('')
False
```

bool()函数并不能够将字符串'False'转换为bool类型数据False，只要当字符串是空字符串时，bool()函数的返回结果才是False，其他情况均返回True

4. 类型转换总结



四种数据类型之间可以互相转换，一种颜色的线条代表一个内置函数。

ascii码表

ASCII 字符代码表 一

高四位 / 低四位		ASCII非打印控制字符 0000 0					ASCII非打印控制字符 0001 1					ASCII 打印字符 0010 2		ASCII 打印字符 0011 3		ASCII 打印字符 0100 4		ASCII 打印字符 0101 5		ASCII 打印字符 0110 6		ASCII 打印字符 0111 7		
		十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	◆	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	▬	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	^Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

ascii表是一套电脑编码系统，例如一个字符a,你看到的是a,但在计算机中，一切都是以二进制存储的，ascii规定了每一个字符在计算机的二进制存储方式，a在计算机中的二进制存储方式是"01100001"，转成成10进制数就是97。

你可能已经注意到，这个表里只有英文字母，没有咱们的汉字，我们中国也需要一套电脑编码系统，于是有了GBK，世界上这么多国家，都有各自的文字，都各自搞一套，不利于交流，于是有了unicode编码，包含了世界上几乎所有的文字，关于这方面，推荐你阅读我的文章 https://zhuanlan.zhihu.com/p/63547961

关于ASCII表，你需要掌握的是数字和英文字母的编码范围

- 数字0~9在ascii表里的编码范围是48~57
- 大写字母在ascii表里的编码范围是65~90
- 小写字母在ascii表里的编码范围是97~122

了解这些内容，你就可以自己完成字符串大小写的转换了。

一文读懂ascii，unicode, utf-8，彻底解决UnicodeEncodeError的问题

1. ASCII

在计算机里，一切都是用二进制存储的，比如 a 这个字母，在计算机里，用 0110 0001 这个8个bit来表示，8个bit就是一个字节。所谓ascii，就是一个字符编码，它规定了英文中的各种字符在计算机里表示形式。

ascii码作为一种字符编码，可以表示128个字符与二进制之间的关系，字符a的二进制编码是“0110 0001”,把这个二进制转成10进制就是97，下面的代码可以处理这种关系的转换

```
en_str = 'a'
en_ascii = ord(en_str)
print(en_ascii, type(en_ascii))

print(chr(97))
```

输出结果

```
97 <class 'int'>
a
```

2. unicode

2.1 大一统

只要稍微一思考，就会发现一个严重的问题，ascii码只是对英文的字符进行编码，可是这个世界上的语言文字又不仅仅只有英文，我们常用的汉字就有几千个，可ascii码只能对128个字符进行编码，这让我们中国人咋办

于是乎，我们中国人就搞出了GB2312，GBK这两个字符集，ascii用一个字节进行编码，我们汉字太多，因此我们用多个字节进行编码。

中国人搞一套，法国人搞一套，俄罗斯人又搞了一套，渐渐的，就乱套了。

干脆，搞一个大点的字符集，把这个世界上所有的字符都进行编码，然后大家就用这套编码来处理文本，这就是unicode字符集。

2.2 大一统的问题

unicode只是一个字符集，它规定了不同的字符在二进制上的表示形式，比如“升”这个汉字，它的unicode编码是 \u5347，5347是16进制，转换成成10进制是21319，转成二进制是101 0011 0100 0111，这一个汉字，至少需要2个字节来表示。

下面的代码，演示了获取一个汉字的编码内容

```
# 1 转成unicode
ch = '升'
ch_unicode = ch.encode('unicode_escape')
print(ch_unicode)

# 2 转成16进制形式
ch_hex = "0x" + str(ch_unicode,encoding='utf-8')[2:]
print(ch_hex)

# 3 转成10进制
ch_int = eval(ch_hex)
print(ch_int)

# 4 转成二进制
print(bin(ch_int))
```

程序运行结果

```
b'\\u5347'
0x5347
21319
0b101001101000111
```

unicode并没有规定这些字符所对应的二进制代码，但是并没有规定这些二进制代码该如何存储。这个汉字两个字节就能存储，但有些字符需要三个字节，像a这种字符，以前大家用ascii码的时候，用一个字节就能表示，在unicode里如果用两个或者更多字节表示，那么不是很浪费么，而且也与之前的ascii不兼容。

3. utf-8

utf-8解决了unicode的编码问题，它是一种变长的编码方式，ascii码表里的字符仍然用一个字节来存储，一个汉字用三个字节来存储

```
ascii_a = 'a'
ascii_a_utf8 = ascii_a.encode(encoding='utf-8')
print(ascii_a_utf8, len(ascii_a_utf8))

ch = '升'
ch_utf8 = ch.encode(encoding='utf-8')
print(ch_utf8, len(ch_utf8))
```

程序运行结果

```
b'a' 1
b'\xe5\x8d\x87' 3
```

4. python3里的unicode

在python3中，字符串是以unicode编码的，当你想把一个字符串写入到磁盘上时，就必须指定用哪种编码方式进行存储，否则，就容易出错，比如下面的这段代码

```
with open('city', 'w') as f:
    f.write('北京')
```

报的错误是

```
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1:
ordinal not in range(128)
```

有了前面的内容做铺垫，你大概可以知道究竟发生了什么错误。

字符串采用的是unicode字符集，但是文件保存的时候，默认采用ascii编码，这就有问题了，ascii可以表示的范围太有限了，只有128个字符，可是汉字的unicode编码里很容就出现大于128的字节，这就是错误发生的原因，解决这个问题，可以采取下面两种方法

4.1 指定utf-8编码

```
with open('city', 'w', encoding='utf-8') as f:
    f.write('北京')
```

4.2 以二进制的形式写入文件

```
with open('city', 'wb') as f:
    f.write('北京'.encode('utf-8'))
```

这种方法虽然也行，但并不常用，因为这需要每次写入都对字符串进行utf-8编码，不如第一种方法简单高效。

split函数切分字符串时为什么会产生空字符串

群里的一个小伙伴提了一个问题，他使用split函数切分字符串得到的结果里有空字符串，他对空字符串的出现感到困惑不解。

```
text = '1aa2'
print(text.split('a'))      # ['1', '', '2']
```

这个空字符串是如何产生的呢？以a为分隔符切分时，我们认为两个a之间有一个空字符串，因此产生一个空字符串。

感觉很奇怪，但是很合理。如果不这样处理，那么结果就变成了 ['1', '2']，符合你的直觉，但这回导致一个尴尬的情况

```
text = '1aa2'
print(text.split('aa'))  # ['1',  '2']
```

字符串'1aa2' 分别以a和aa为分隔符切分，得到的结果竟然是一样的，这显然是不合逻辑的，这就好比两根油条，长度一样，你用菜刀分别以两个不同的宽度去切油条，最终两根油条切完后完全一样，这简直是匪夷所思。

split函数切分字符串时产生的字符串，是合情合理的，它避免了一个字符串以不同的分隔符切分却产生相同结果的情况发生。

字符串练习题

1. 字符串方法练习题

在交互式解释器中完成下列题目

```
1. 将字符串 "abcd" 转成大写
2. 计算字符串 "cd" 在 字符串 "abcd"中出现的位置
3. 字符串 "a,b,c,d" ，请用逗号分割字符串，分割后的结果是什么类型的？
4. "{name}喜欢{fruit}".format(name="李雷") 执行会出错，请修改代码让其正确执行
5. string = "Python is good", 请将字符串里的Python替换成 python,并输出替换后的结果
6. 有一个字符串 string =  "python修炼第一期.html"，请写程序从这个字符串里获得.html前面的部分，要用尽可能多的方式来做这个事情
7. 如何获取字符串的长度？
8. "this is a book",请将字符串里的book替换成apple
9. "this is a book", 请用程序判断该字符串是否以this开头
10. "this is a book", 请用程序判断该字符串是否以apple结尾
11. "This IS a book",  请将字符串里的大写字符转成小写字符
12. "This IS a book",  请将字符串里的小写字符，转成大写字符
13. "this is a book\n",  字符串的末尾有一个回车符，请将其删除
```

答案如下

```
。。。。。。
```

这里只对其中2个题目讲解

第4小题的程序直接运行会报错，因为字符串里面有两个需要替换的位置，而format方法里只传入了一个参数，显然是不够

第13小题，strip() 方法用于移除字符串头尾指定的字符（默认为空格或换行符）或字符序列

2. 逻辑推理练习

不用解释器执行代码，直接说出下面代码的执行结果

```
string = "Python is good"
1. string[1:20]
2. string[20]
3. string[3:-4]
4. string[-10:-3]
5. string.lower()
6. string.replace("o", "0")
7. string.startswith('python')
8. string.split()
9. len(string)
10. string[30]
11. string.replace(" ", '')
```

答案如下

```
。。。。。。
```

第2题和第10题都报错，是因为超出了索引范围，字符串长度为14，你去20和30的位置取值，当然会报错

关于切片操作，只需要知道从哪里开始到哪里结束就一定能推导出答案，以string[3:-4]为例，3是开始的位置，-4是结束的位置，但这个范围是左闭右开的，从3开始没错，但不会到-4，而是到-5，更前面的一个位置，python支持负数索引，或者说是反向索引，从右向左从-1开始逐渐减小。

第一题中，做切片的时候是从1开始，到20结束，即便是右开，直到19，也仍然超出了索引范围，为什么不报错呢，这就是语言设计者自己的想法了，切片时，不论是开始位置还是结束位置，超出索引范围都不会报错，我猜，大概是由于切片是一个范围操作，这个范围内有值就切出来，没值返回空字符串就好了。