Python实验报告

实验四 字符串处理

学号：2016326603046 姓名：余泓锷

实验目的

了解python中字符串的定义
掌握python中字符串格式化处理
掌握python中字符串分隔、连接、输出的基本操作及其对应方法
了解python中正则表达式的规则并能进行字符串匹配

实验内容

实验题1：
验证这些字符串方法的作用
isalnum()、isalpha()、isdigit()、isdecimal()、isnumeric()、isspace()、isupper()、islower()
zfill()、center()、ljust()、rjust()、s.startswith()、s.endswith()
解答：

In [1]:

```
# isalnum() 字符串是否由字母和数字组成
a = 'abcd1234'
b = 'How are you?'
print('a.isalnum() =',a.isalnum())
print('b.isalnum() =',b.isalnum())
```

```
a.isalnum() = True
b.isalnum() = False
```

In [2]:

```
# isalpha() 检测字符串是否只由字母组成
a = 'abcd'
b = 'abcd1234'
print("a.isalpha() = ",a.isalpha())
print("b.isalpha() = ",b.isalpha())
```

```
a.isalpha() =  True
b.isalpha() =  False
```

In [3]:

```
# isdigit() isdecimal() isnumeric() 均为检测字符串是否只由数字组成
num = u"1"  #unicode
print('num.isdight() =',num.isdigit())   # True
print('num.isdecimal() =',num.isdecimal()) # True
print('num.isnumeric() =',num.isnumeric()) # True
print()
num = "1" # 全角
print('num.isdight() =',num.isdigit())   # True
print('num.isdecimal() =',num.isdecimal()) # True
print('num.isnumeric() =',num.isnumeric()) # True
print()
num = b"1" # byte
print('num.isdight() =',num.isdigit())   # True
#print('num.isdecimal() =',num.isdecimal()) # AttributeError 'bytes' object has no attribute 'isdeci
#nprint('num.isnumeric() =',num.isnumeric()) # AttributeError 'bytes' object has no attribute 'isnum
print()
num = "Ⅲ" # 罗马数字
print('num.isdight() =',num.isdigit())   # False
print('num.isdecimal() =',num.isdecimal()) # False
print('num.isnumeric() =',num.isnumeric()) # True
print()
num = "四" # 汉字
print('num.isdight() =',num.isdigit())   # False
print('num.isdecimal() =',num.isdecimal()) # False
print('num.isnumeric() =',num.isnumeric()) # True
```

```
num.isdight() = True
num.isdecimal() = True
num.isnumeric() = True

num.isdight() = True
num.isdecimal() = True
num.isnumeric() = True

num.isdight() = True

num.isdight() = False
num.isdecimal() = False
num.isnumeric() = True

num.isdight() = False
num.isdecimal() = False
num.isnumeric() = True
```

In [4]:

```
# isupper() 检测字符串中所有的字母是否都为大写
# islower() 检测字符串是否由小写字母组成
a = 'ABCD'
b = 'abcd'
c = 'aCcD'
print('a.isupper() ={0}\ta.isupper() ={1}'.format(a.isupper(),a.islower()))
print('b.isupper() ={0}\tb.isupper() ={1}'.format(b.isupper(),b.islower()))
print('c.isupper() ={0}\tc.isupper() ={1}'.format(c.isupper(),c.islower()))
```

```
a.isupper() =True       a.isupper() =False
b.isupper() =False      b.isupper() =True
c.isupper() =False      c.isupper() =False
```

In [5]:

```
# zfill(width) 返回指定长度的字符串，原字符串右对齐，前面填充0
a = 'abcd'
print('a.zfill(3) = ',a.zfill(3))
print('a.zfill(4) = ',a.zfill(4))
print('a.zfill(5) = ',a.zfill(5))
```

```
a.zfill(3) =  abcd
a.zfill(4) =  abcd
a.zfill(5) =  0abcd
```

In [6]:

```
# center(width,fillchar) 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
a = 'abcd'
print('a.center(3) = ',a.center(3))
print('a.center(5) = ',a.center(5))
print('a.center(5,\'0\') = ',a.center(5,'0'))
print('a.center(6) = ',a.center(6))
print('a.center(6,\'0\') = ',a.center(6,'0'))
```

```
a.center(3) =  abcd
a.center(5) =   abcd
a.center(5,'0') =  0abcd
a.center(6) =   abcd
a.center(6,'0') =  0abcd0
```

In [7]:

```
# ljust() 返回一个原字符串左对齐
#         并使用空格填充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
# rjust() 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
#         如果指定的长度小于字符串的长度则返回原字符串
a = 'abcd'
print('a.ljust(3) = {0}\ta.rjust(3) = {1}'.format(a.ljust(3),a.rjust(3)))
print('a.ljust(5) = {0}\ta.rjust(5) = {1}'.format(a.ljust(5),a.rjust(5)))
print('a.ljust(5,\'0\') = {0}\ta.rjust(5,\'0\') = {1}'.format(a.ljust(5,'0'),a.rjust(5,'0')))
print('a.ljust(6) = {0}\ta.rjust(6) = {1}'.format(a.ljust(6),a.rjust(6)))
print('a.ljust(6,\'0\') = {0}\ta.rjust(6,\'0\') = {1}'.format(a.ljust(6,'0'),a.rjust(6,'0')))
```

```
a.ljust(3) = abcd       a.rjust(3) = abcd
a.ljust(5) = abcd       a.rjust(5) =  abcd
a.ljust(5,'0') = abcd0  a.rjust(5,'0') = 0abcd
a.ljust(6) = abcd       a.rjust(6) =   abcd
a.ljust(6,'0') = abcd00 a.rjust(6,'0') = 00abcd
```

In [8]:

```
# startswith() 用于检查字符串是否是以指定子字符串开头
#              如果是则返回 True 否则返回 False
#              如果参数 beg 和 end 指定值 则在指定范围内检查
# endswith() 用于检查字符串是否是以指定子字符串结尾
#              如果是则返回 True 否则返回 False
#              如果参数 beg 和 end 指定值 则在指定范围内检查
a = 'abcdefghijklmnopqrstuvwxyz'
print('a.startswith(\'a\') = {0}\ta.endswith(\'z\') = {1}'.format(a.startswith('a'),a.endswith('z'))
print('a.startswith(\'ac\') = {0}\ta.endswith(\'xz\') = {1}'.format(a.startswith('ac'),a.endswith('x
print('a.startswith(\'df\',3) = {0}\ta.endswith(\'vw\',18,24) = {1}'.format(a.startswith('def',3),a.
```

```
a.startswith('a') = True        a.endswith('z') = True
a.startswith('ac') = False      a.endswith('xz') = False
a.startswith('df',3) = True     a.endswith('vw',18,24) = False
```

实验题2：
测试字符串连接是用+快，还是使用join快。
解答：

In [9]:

```
import time
m = 10000
print('为使结果更明显，对每次的字符串组两种连接方式各重复10000次\n再把时间差除以10000得到最终时间')
for n in range(1,21,2):
    print('字符串组有{0}个字符串的情况，结果如下：'.format(n))
    lj_str = ''
    begin = time.time()
    for j in range(m):
        for i in range(n):
            lj_str += str(i)
    end = time.time()
    print ("The  '+' time is:", (end-begin)/10000)
    li_str2 = ''
    begin = time.time()
    for j in range(m):
        li_str2 = ''.join([str(i) for i in range(n)])
    end = time.time()
    print ("The  'join' time is:", (end-begin)/10000)
```

为使结果更明显，对每次的字符串组两种连接方式各重复10000次
再把时间差除以10000得到最终时间
字符串组有1个字符串的情况，结果如下：
The '+' time is: 3.3001899719238283e-06
The 'join' time is: 3.7002086639404297e-06
字符串组有3个字符串的情况，结果如下：
The '+' time is: 6.2003374099731445e-06
The 'join' time is: 4.500269889831543e-06
字符串组有5个字符串的情况，结果如下：
The '+' time is: 9.700584411621093e-06
The 'join' time is: 5.900335311889649e-06
字符串组有7个字符串的情况，结果如下：
The '+' time is: 1.2800717353820801e-05
The 'join' time is: 7.200407981872559e-06
字符串组有9个字符串的情况，结果如下：
The '+' time is: 1.5600895881652832e-05
The 'join' time is: 8.70051383972168e-06
字符串组有11个字符串的情况，结果如下：
The '+' time is: 1.890108585357666e-05
The 'join' time is: 9.900569915771485e-06
字符串组有13个字符串的情况，结果如下：
The '+' time is: 2.1901249885559082e-05
The 'join' time is: 1.0800647735595703e-05
字符串组有15个字符串的情况，结果如下：
The '+' time is: 2.5101423263549805e-05
The 'join' time is: 1.2000679969787597e-05
字符串组有17个字符串的情况，结果如下：
The '+' time is: 2.8101587295532228e-05
The 'join' time is: 1.3400793075561524e-05
字符串组有19个字符串的情况，结果如下：
The '+' time is: 3.136167526245117e-05
The 'join' time is: 1.4900875091552735e-05

结论：在连接的字符串比较少的时候，'+'的连接效率高；在连接的字符串比较多的时候，'join'的效率比较高

实验题3：

以下是一个html文本，请将其中所有的html标签去除
解答：

In [10]:

```
这是一个例子，正式的原文在下一个Cell
原文：
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta property="qc:admins" content="465267610762567726375" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Python3 正则表达式 | 菜鸟教程</title>
去除以后：
Python3 正则表达式 | 菜鸟教程
```

```
  File "<ipython-input-10-e90e840daf41>", line 1
    这是一个例子，正式的原文在下一个Cell
          ^
SyntaxError: invalid character in identifier
```

In [11]:

```
text = '''<div id="topics">
    <div class = "post">
        <h1 class = "postTitle">
            <a id="cb_post_title_url" class="postTitle2" href="https://www.cnblogs.com/cq90/p/695956
        </h1>
        <div class="clear"></div>
        <div class="postBody">
            <div id="cnblogs_post_body" class="blogpost-body"><p>u/U:表示unicode字符串 <br>不是
<p>r/R:非转义的原始字符串 <br>与普通字符相比，其他相对特殊的字符，其中可能包含转义字符，即那些，
<p>b:bytes <br>python3.x里默认的str是(py2.x里的)unicode, bytes是(py2.x)的str, b”“前缀代表的就
<p> </p>
<p>参考：http://blog.csdn.net/u010496169/article/details/70045895</p></div><div id="MySignature"></d
<div class="clear"></div>
<div id="blog_post_info_block">
<div id="BlogPostCategory"></div>
<div id="EntryTag"></div>
<div id="blog_post_info">
</div>
<div class="clear"></div>
<div id="post_next_prev"></div>
</div>'''
```

In [12]:

```
text
```

Out[12]:

' \n\t \n\t\t<h1 class = "postTitle">\n\t\t\tpython学习-字符串前面添加u,r,b的含义\n\t\t</h1>\n\t\t \n\t\t \n\t\t\t u/U:表示unicode字符串
不是仅仅是针对中文, 可以针对任何的字符串，代表是对字符串进行unicode编码。
一般英文字符在使用各种编码下, 基本都可以正常解析, 所以一般不带u；但是中文, 必须表明所需编码, 否则一旦编码转换就会出现乱码。
建议所有编码方式采用utf8 \n r/R:非转义的原始字符串
与普通字符相比，其他相对特殊的字符，其中可能包含转义字符，即那些，反斜杠加上对应字母，表示对应的特殊含义的，比如最常见的”\n”表示换行，”\t”表示Tab等。而如果是以r开头，那么说明后面的字符，都是普通的字符了，即如果是“\n”那么表示一个反斜杠字符，一个字母n，而不是表示换行了。
以r开头的字符，常用于正则表达式，对应着re模块。 \n b:bytes
python3.x里默认的str是(py2.x里的)unicode, bytes是(py2.x)的str, b”“前缀代表的就是bytes
python2.x里, b前缀没什么具体意义， 只是为了兼容python3.x的这种写法 \n   \n 参考：http://blog.csdn.net/u010496169/article/details/70045895 \n \n \n \n \n \n \n \n \n '

In [13]:

```
import re
witch = re.compile('>[\n|\t]+<')
text1 = witch.sub('><',text)
witch = re.compile('[\n]')
text1 = witch.sub('\\\\n',text1)
witch = re.compile('[\t]')
text1 = witch.sub('\\\\t',text1)
witch = re.compile('</')
text1 = witch.sub('\\n</',text1)
witch = re.compile(' ')
text1 = witch.sub(' ',text1)
huanhang = re.compile('<br>')
text1 = huanhang.sub('\\n',text1)

biaoqian = re.compile('<[^>]*>')
text2 = biaoqian.sub('',text1)
print(text2)
```

python学习-字符串前面添加u,r,b的含义

u/U:表示unicode字符串
不是仅仅是针对中文，可以针对任何的字符串，代表是对字符串进行unicode编码。
一般英文字符在使用各种编码下，基本都可以正常解析，所以一般不带u；但是中文，必须表明所需编码，否则一旦编码转换就会出现乱码。
建议所有编码方式采用utf8
r/R:非转义的原始字符串
与普通字符相比，其他相对特殊的字符，其中可能包含转义字符，即那些，反斜杠加上对应字母，表示对应的特殊含义的，比如最常见的”\n”表示换行，”\t”表示Tab等。而如果是以r开头，那么说明后面的字符，都是普通的字符了，即如果是“\n”那么表示一个反斜杠字符，一个字母n，而不是表示换行了。
以r开头的字符，常用于正则表达式，对应着re模块。
b:bytes
python3.x里默认的str是(py2.x里的)unicode，bytes是(py2.x)的str，b”“前缀代表的就是bytes
python2.x里，b前缀没什么具体意义， 只是为了兼容python3.x的这种写法

参考：http://blog.csdn.net/u010496169/article/details/70045895