Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第五章

字符串与正则表达式

- 字符串是一个字符序列。
- 字符串常量是用单引号、双引号或三引号括起来的若干个字符。
- 字符串变量是用来存放字符串常量的变量。
- 正则表达式（regular expression）是用于字符串处理的强大工具，它使用预定义的特定模式去匹配一类具有共同特征的字符串，可以快速完成复杂字符串的查找、替换等处理。

5.1 字符串编码

- 1. Unicode码
- 统一码、万国码、单一码。
- 它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。
- 1994年正式公布。
- 它使用4字节的数字编码来表达每个符号。

- **2. UTF-8码**
- 是一种为Unicode字符设计的变长编码系统。
- 对于ASCII字符，UTF-8仅使用1个字节来编码。
- 一些复杂的字符则占用2、3或4个字节。
- Python支持UTF-8编码，中文字符、希腊字母均可以作为标识符使用。
- 例：
- >>>国家="中国"
- >>>国家
- '中国'

- 例：
- >>> 电子=2
- >>> 科技=3
- >>> 大学=电子+科技
- >>> 大学
- 5
- >>> type(大学)
- <class 'int'>

- Python提供了ord和chr两个内置函数，用于字符与其机器内部Unicode码编码值之间的转换。
- ord()函数将一个字符转化为相应的Unicode码编码值；chr()函数将一个整数转换成Unicode字符。
- 例：
- >>>print(ord('a'),ord('A'),ord('0'))
- 97 65 48
- >>>print(chr(97),chr(65),chr(48))
- a A 0

- 例：
- >>>print(ord('电'),ord('子'),ord('科'),ord('大'))
- 30005 23376 31185 22823
- >>>print(chr(20013),chr(22269))
- 中 国

- **3. Unicode码与UTF-8码的转换**
- **Unicode是一种类似于ASCII码的编码标准。**
- **UTF(Unicode Transformation Format)是指Unicode传送格式，即把Unicode文件转换成字节传送流。**
- **UTF-8是为传送Unicode字符而设计出来的“再编码”方法，它是互联网上使用最广的一种Unicode实现方式。**
- **UTF-8是以8个二进制位为单元对Unicode进行编码。**

Unicode 编码（十六进制）	UTF-8 字节流（二进制）
0000 - 007F	0xxxxxxx
0080 - 07FF	110xxxxx 10xxxxxx
0800 - FFFF	1110xxxx 10xxxxxx 10xxxxxx

- 一个Unicode码可能转成长度为1个字节，或2个、3个、4个字节的Unicode码，这取决于其Unicode码值。
- 英文字符的Unicode码值小于80H，只用一个字节的UTF-8传送，传送速度较快。

Unicode 编码（十六进制）	UTF-8 字节流（二进制）
0000 - 007F	0×××××××
0080 - 07FF	110×××× 10××××××
0800 - FFFF	1110××× 10×××××× 10××××××

- 例：“汉”字的Unicode编码是6C49H。
- 6C49在0800-FFFF之间，所以要用3字节模板：

  1110×××10××××××10××××××。
- 将6C49H写成二进制是：0110 1100 0100 1001。
- 用这个比特串依次代替模板中的×，得到：
- 1110 0110 1011 0001 1000 1001。
- 即“汉”的UTF-8编码是E6B189H。

- encode()方法：把Unicode码编码为指定编码方式。
- 例：
- s.encode("utf-8")：表示把字符串s从Unicode码编码为UTF-8码。这里要求s必须是Unicode编码方式，否则会出错。
- decode()方法：从指定编码方式解码为Unicode码。
- 例：
- s.decode("utf-8")：表示s从UTF-8编码方式解码为Unicode码。这里要求s必须是UTF-8编码方式，否则会出错。

- 例：
- >>>s="汉字ABC"
- >>>k=s.encode('utf-8')
- >>>k
- b'\xe6\xb1\x89\xe5\xad\x97ABC'
- >>>k.decode('utf-8')
- '汉字ABC'

- s="中国"
- k=s.encode('utf-8')
- print(k)
- print(k.decode('utf-8'))
- 运行结果：
- b'\xe4\xb8\xad\xe5\x9b\xbd'
- 中国

- 在Unicode字符串中，默认一个中文占一个字符；
- 编码格式为UTF-8时，一个中文占3个字符；
- 编码格式是GBK时，一个中文占2个字符。
- 例：
- >>>s='汉'　　#默认是Unicode码
- >>>len(s)
- 1

- >>>s='汉'　　#默认是Unicode码
- >>>s=s.encode('utf-8')　　#指定为UTF-8编码
- >>>len(s)
- 3
- >>>print(s[0],s[1],s[2])
- #输出“汉”的3个字节编码值(十进制)
- 230 177 137
- >>>s=s.decode('utf-8')　　#解码为Unicode码

- >>>s='汉'　　#默认是Unicode码
- >>>s=s.encode('gbk')　　#转换为GBK编码
- >>>len(s)
- 2
- >>>s=s.decode('gbk')　　#解码为Unicode码
- >>>ord(s)
- 27721
- >>>s[0]
- '汉'

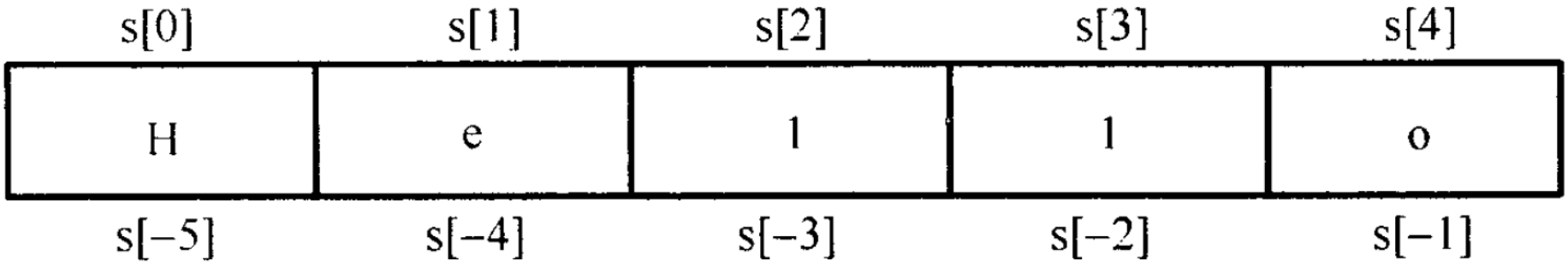
5.2 字符串的索引与分片

- **Python**字符串是一种元素为字符的序列类型。

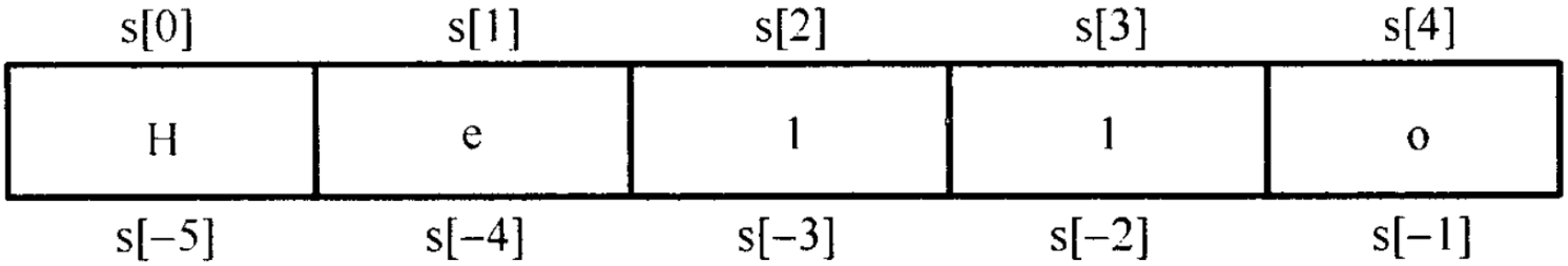
5.2.1 字符串的索引

- 在字符串中，最左边字符的编号为0，最右边字符的编号比字符串的长度小1。
- 在字符串中也可以使用负数从右向左进行编号，最右边的字符（即倒数第1个字符）的编号为-1。
- 字符串变量名后接用中括号括起来的编号即可实现字符串的索引。

- 例：
- >>>s="Hello"
- >>>print(s[0],s[-4])
- H e



- 注意：索引编号必须为整数，且不能越界，否则都会出现错误。
- 例:
- **>>>s="Hello"**
- **>>>s[5]**
- 索引编号越界，出现错误信息“**IndexError: string index out of range**”，意思是字符串索引超出范围。



- 例：将一个字符串中的字符按逆序打印出来。
- **s=input("Please enter a string:")**
- **for i in range(0,len(s)):**
- **　print(s[len(s)-1-i],end='')**
- 运行结果：
- **Please enter a string:ABCDEF**
- **FEDCBA**

5.2.2 字符串的分片（切片）

- 字符串的分片：从给定字符串中分离出部分字符。
- 字符串索引编号：i:j:k
- i是索引起始位置，j是索引结束位置（但不包括j位置上的字符），索引编号每次增加的步长为k。
- 例：
- >>>s="abcdefg"
- >>>print(s[0:5:2])
- ace

- 注意：
- 字符串中首字符的索引编号是0。
- 字符串分片时，不包括索引结束位置上的字符。
- 例：
- >>>s="abcdefg"
- >>>print(s[1:4:1])
- bcd
- >>>print(0:4:1])
- abcd

- 索引又分为正索引和负索引。
- 正索引是以字符串的开始（首字符，其索引编号是0）为起点。
- 负索引是以字符串的结束（末字符，其索引编号是-1）为起点。
- 例：
- **>>>s="abcdef"**
- **>>>print(s[-1:-4:-1])**
- **fed**

- 例：
- >>> s="0123456789"
- >>> print(s[0:10:1])
- 0123456789
- >>> print(s[-1:-11:-1])
- 9876543210
- >>> print(s[0:10:2])
- 02468

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

- 例：
- >>> s="0123456789"
- >>> print(s[3:6:1])
- 345
- >>> print(s[-3:-6:-1])
- 765
- >>> print(s[9:-11:-1])
- 9876543210
- >>> print(s[-10:10:1])
- 0123456789

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- 例：
- >>> s="0123456789"
- >>> s[3:-3:1]
- '3456'
- >>> s[6:-9:-1]
- '65432'

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- 例：
- >>> s="0123456789"
- >>> s[len(s):0:-1]
- '987654321'
- >>> s[-len(s):-1:1]
- '012345678'

>>> len(s)

10

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- 字符串分片的索引编号中，索引起始位置i、索引结束位置j和步长k均可省略。
- 省略i时，从0或-1开始；
- 省略j时，到最后一个字符结束；
- 省略k时，步长为1。
- 例：
- >>> s="0123456789"
- >>> s[:]
- '0123456789'

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- 例：
- >>> s="0123456789"
- >>> s[::2]
- '02468'
- >>> s[:2]
- '01'
- >>> s[::-1]
- '9876543210'
- >>> s[:-1]
- '012345678'

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

结论

- 正序s[i]→负序s[i-len(s)]
- 负序s[-j]→正序s[len(s)-j]
- 分片切片(step可以不等于1)≠求子串(step=1)

- 例：
- **s[2:-3:1]**
  - 其正序表达式为**s[2:len(s)-3:1]**
  - 其负序表达式为**s[2-len(s):-3:1]**

s[start:stop:step]的默认值

- s[start:stop:step]的默认值：
  - 当step>0时： start=0， stop=len(s)
  - 当step<0时： start=-1， stop=-1-len(s)

区别：

- range(start=0,stop,step=1)
  - （1）标点符号不同，分别是冒号和逗号；
  - （2）range中的stop无默认值。

简化的等效写法

- s[2::]===>s[2:len(s):1]
- s[2:]===>s[2::]===>s[2:len(s):1]
- s[2]===>s[2:3:1]
- s[:2]===>s[:2:]===>s[0:2:1]
- s[::2]===>s[0:len(s):2]
- s[::]===>s[:]===> s[0:len(s):1]

- 分片的操作很灵活，开始和结束的索引值可以超过字符串的长度。
- 例：
- **>>> s="0123456789"**
- **>>> s[-100:100]**
- **'0123456789'**

正索引编号	0	1	2	3	4	5	6	7	8	9
字符串	0	1	2	3	4	5	6	7	8	9
负索引编号	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- 例：利用字符串分片方法将一个字符串中的字符按逆序打印出来。
- **s1=input("Please enter a string:")**
- **s2=s1[::-1]**
- **print(s2)**

```
Please enter a string:abcdefg
gfedcba
```

5.3 字符串的操作

- 在Python中，字符串可以进行连接操作、逻辑操作，还有字符串处理函数。

5.3.1 字符串连接操作

- 1. 基本连接操作
- 字符串连接运算符为“+”，表示将两个字符串数据连接起来，成为一个新的字符串数据。
- 例：
- >>>"Sub"+"string"
- 'Substring'

- 将字符串和数值数据进行连接时，需要将数值数据用str()函数或repr()函数转换成字符串，然后再进行连接。
- 例：
- >>> "x="+str(3.14159)
- 'x=3.14159'

- 注：str() 与 repr()的区别
- str() 的输出追求可读性，输出格式要便于理解，适合用于输出内容到用户终端。
- repr() 的输出追求明确性，除了对象内容，还需要展示出对象的数据类型信息，适合开发调试阶段使用。
- 例：
- >>>print(str("abc"))
- abc　　#对用户友好
- >>>print(repr("abc"))
- 'abc'　　#对Python友好

- Python的字符串是不可变类型，只能通过新建一个字符串去改变一个字符串的元素。
- 例：
- >>>s="abcd"
- >>>s[1]="k"　#错误操作
- 该操作可以换一种方式实现：
- >>>s="abcd"
- >>>s=s[0]+"k"+s[2:]
- >>>s
- 'akcd'

- **2. 重复连接操作**
- 使用乘法运算符“*”，构建一个由其自身字符串重复连接而成的字符串。
- 格式：s * n 或 n * s
- s是一个字符串；n是一个正整数，代表重复的次数。

- 例：
- >>> "abc"*2
- 'abcabc'
- >>> 3*"abc"
- 'abcabcabc'
- >>> 2*"abc"*3
- 'abcabcabcabcabcabc'
- >>> 0*"abc"
- ''

- 连接运算符“+”和重复操作符“*”也支持复合赋值运算。
- 例：
- >>> s="abc"　　>>> s="abc"
- >>> s*=2　　>>> s+="def"
- >>> s　　>>> s
- 'abcabc'　　'abcdef'

- 例：从键盘输入3个字符串，将它们连接成一个字符串后输出。
- s=''
- for i in range(0,3):
- 　　c=input("Please enter a string:")
- 　　s+=c
- print(s)

```
Please enter a string:aa
Please enter a string:bb
Please enter a string:cc
aabbcc
```

- Python中字符串是不可变的类型，使用“+”连接两个字符串时会生成一个新的字符串，生成新的字符串就需要重新申请内存，当连续相加的字符串很多时，效率就很低。
- 对于这种连加操作，可以用格式化操作符或join()函数取代，这样只会有一次内存的申请，以提高运算效率。

- 例：
- >>> '{:s}{:s}{:s}'.format('aa','bb','cc')
- 'aabbcc'
- >>> ''.join(['aa','bb','cc'])
- 'aabbcc'
- >>> '**'.join(['aa','bb','cc'])
- 'aa**bb**cc'
- s.join()函数使用s作为分隔符，把列表里的元素连接成一个字符串。

5.3.2 字符串逻辑操作

- 字符串的逻辑操作是指字符串参与逻辑比较，其操作的结果是一个逻辑量，通常用于表达字符处理的条件。

1. 关系操作

- 字符比较是按其计算机内部字符编码值的大小进行比较，西文字符按ASCII码值大小进行比较。
- 比较的基本规则是，空格字符最小，数字比字母小，大写字母比小写字母小（对应字母相差32）。
- 0----48----30H
- A----65----41H
- a----97----61H

- 字符串比较的规则:
- （1）单个字符比较，按字符ASCII码值大小进行比较。
- 例：
- >>> "A">"a"
- False
- >>> "A">"5"
- True

- （2）两个相同长度的字符串的比较是将字符串中的字符从左向右逐个比较，如果所有字符都相等，则两个字符串相等；
- 如果两个字符串中有不同的字符，以最左边的第1对不同字符的比较结果为准。
- 例：
- **>>> "abcd">"ab2d"**
- **True**

- （3）若两个字符串中字符个数不等时，则将较短的字符串后面补足空格后再比较。
- 例：
- >>> ord(' ')
- 32
- >>> "abcd">"abc"
- True
- >>> "aBcd">"abc"
- False

- 例：从键盘输入10个英文单词，输出其中以元音字母开头的单词。

```
ss='AEIOUaeiou'
for i in range(0,10):
    s=input("Please enter a word:")
    for c in ss:
        if s[0]==c:
            print(s)
            break
```

2. 成员关系操作

- 字符串的成员关系操作包括 in 和 not in 操作.
- 格式：字符串1 [not] in 字符串2
- 该操作用于判断字符串1是否属于字符串2。
- 例：
- >>> 'c' not in "uestc"
- False
- >>> "st" in "uestc"
- True

5.3.3 字符串的常用方法

- 字符串对象是不可改变的。
- 创建一个字符串后，不能改变这个字符中的某一部分。
- 任何字符串方法改变了字符串后，都会返回一个新的字符串，原字符串并没有变。

- 经常用到函数（function）和方法（method）两个概念。
- 例如：
- ord()和chr()是两个内置函数。
- upper()和lower()是字符串的两个方法。
- 它们是同一个概念，即具有独立功能、由若干语句组成的一个可执行程序段。
- 但它们又是有区别的。

- 函数是面向过程程序设计的概念，方法是面向对象程序设计的概念。
- 在面向对象程序设计中，类的成员函数称为方法，所以方法本质上还是函数，只不过是写在类里面的函数。

- 方法依附于对象，没有独立于对象的方法。
- 面向过程程序设计中的函数是独立的程序段。
- 函数可以通过函数名直接调用，例如：ord('A')。
- 而对象中的方法则要通过对象名和方法名来调用，一般形式为：
- 对象名.方法名(参数)

- 在Python中，字符串类型（string）可以看成一个类（class），而一个具体的字符串可以看成一个对象，该对象具有很多方法，这些方法是通过类的成员函数来实现的。
- 例：
- >>> "abc123def".upper()
- 'ABC123DEF'

1. 字母大小写转换

- **s.upper()：全部转换为大写字母。**
- **s.lower()：全部转换为小写字母。**
- **s.swapcase()：字母大小写互换。**
- **s.capitalize()：首字母大写，其余小写。**
- **s.title()：每个单词首字母大写。**

- 例：
- s='Python Program'
- print('{:s} lower={:s}'.format(s,s.lower()))
- print('{:s} upper={:s}'.format(s,s.upper()))
- print('{:s} swapcase={:s}'.format(s,s.swapcase()))
- print('{:s} capitalize={:s}'.format(s,s.capitalize()))
- print('{:s} title={:s}'.format(s,s.title()))

```
Python Program lower=python program
Python Program upper=PYTHON PROGRAM
Python Program swapcase=pYTHON pROGRAM
Python Program capitalize=Python program
Python Program title=Python Program
```

2. 字符串对齐处理

- **s.ljust(width,[fillchar])：输出width个字符，s左对齐，右边不足部分用fillchar填充，默认用空格填充。**
- **s.rjust(width,[fillchar])：输出width个字符，s右对齐，左边不足部分用fillchar填充，默认用空格填充。**
- **s.center(width,[fillchar])：输出width个字符，s中间对齐，两边不足部分用fillchar填充，默认用空格填充。**
- **s.zfill(width)：把s变成width长，并且右对齐，左边不足部分用0补齐。**

- 例：
- s='Python Program'
- print('{:s} ljust={:s}'.format(s,s.ljust(20)))
- print('{:s} rjust={:s}'.format(s,s.rjust(20)))
- print('{:s} center={:s}'.format(s,s.center(20)))
- print('{:s} zfill={:s}'.format(s,s.zfill(20)))

```
Python Program ljust=Python Program
Python Program rjust=      Python Program
Python Program center=   Python Program
Python Program zfill=000000Python Program
```

3. 字符串搜索

- s.find(substr[,start,[end]])：返回s中出现substr的第1个字符的编号，如果s中没有substr则返回-1。start和end作用就相当于在s[start:end]中搜索。
- s.index(substr[,start,[end]])：与find()相同，只是在s中没有substr时，会返回一个运行时错误。
- s.rfind(substr[,start,[end]])：返回s中最后出现的substr的第1个字符的编号，如果s中没有substr则返回-1，即从右边算起的第1次出现的substr的首字符编号。
- s.rindex(substr[,start,[end]])：与rfind()相同，只是在s中没有substr时，会返回一个运行时错误。

- s.count(substr[,start,[end]])：计算substr在s中出现的次数。
- s.startswith(prefix[,start,[end]])：是否以prefix开头，若是返回True，否则返回False。
- s.endswith(suffix[,start,[end]])：是否以suffix结尾，若是返回True，否则返回False。

- 例： s='Python Program'
- print('{:s} find nono={:d}'.format(s,s.find('nono')))
- print('{:s} find t={:d}'.format(s,s.find('t')))
- print('{:s} find t from {:d}={:d}'.format(s,1,s.find('t',1)))
- print('{:s} find t from {:d} to

  {:d}={:d}'.format(s,1,2,s.find('t',1,2)))
- print('{:s} rfind t={:d}'.format(s,s.rfind('t')))
- print('{:s} count t={:d}'.format(s,s.count('t')))

```
Python Program find nono=-1
Python Program find t=2
Python Program find t from 1=2
Python Program find t from 1 to 2=-1
Python Program rfind t=2
Python Program count t=1
```

4. 字符串替换

- s.replace(oldstr,newstr[,count])：把s中的oldstr替换为newstr，count为替换次数。这是替换的通用形式，还有一些函数进行特殊字符的替换。
- s.strip([chars])：把s中前后chars中有的字符全部去掉，可以理解为把s前后chars替换为None。默认去掉前后空格。

- **s.lstrip([chars])：把s左边chars中有的字符全部去掉。默认去掉左边空格。**
- **s.rstrip([chars])：把s右边chars中有的字符全部去掉。默认去掉右边空格。**
- **s.expandtabs([tabsize])：把s中的tab字符替换为空格，每个tab替换为tabsize个空格，默认是8个。**

- 例：
- s='Python Program'
- print('{:s} replace t to *={:s}'.format(s,s.replace('t','*')))
- print('{:s} replace t to *={:s}'.format(s,s.replace('t','*',1)))
- print('{:s} strip={:s}'.format(s,s.strip()))
- print('{:s} strip={:s}'.format(s,s.strip('Pm')))

```
Python Program replace t to *=Py*hon Program
Python Program replace t to *=Py*hon Program
Python Program strip=Python Program
Python Program strip=ython Progra
```

5. 字符串的拆分与组合

- **s.split([sep[,maxsplit]])：以sep为分隔符，把字符串s拆分成一个列表。默认的分隔符为空格。maxsplit表示拆分的次数，默认取-1，表示无限制拆分。**
- **s.rsplit([sep[,maxsplit]])：从右侧把字符串s拆分成一个列表。**
- **s.splitlines([keepends])：把s按行拆分为一个列表。keepends是一个逻辑值，如果为True，则每行拆分后会保留行分隔符。**

- 例：
- >>> s="""fdfd
- fdfdfd
- gf
- 3443
- """
- >>> s.splitlines()
- ['fdfd', 'fdfdfd', 'gf', '3443']
- >>> s.splitlines(True)
- ['fdfd\n', 'fdfdfd\n', 'gf\n', '3443\n']

- s.partition(sub)：从sub出现的第1个位置起，把字符串s拆分成一个3元素的元组（sub左边字符，sub，sub右边字符）。如果s中不包含sub，则返回(s，",")。
- s.rpartition(sub)：从右侧开始，把字符串s拆分成一个3元素的元组（sub左边字符，sub，sub右边字符）。如果s中不包含sub则返回(","s)。

- s.join(seq)：把seq代表的序列组合成字符串，用s将序列各元素连接起来。
- 字符串中的字符是不能修改的。
- 如果要修改，通常的方法是，用list()函数把字符串s变为以单个字符为成员的列表（使用语句s=list(s)），再使用给列表成员赋值的方式改变值（如s[3]='a'），最后再使用语句“s="".join(s)”还原成字符串。

- 例：
- s='a b c de'
- print('{:s} split={}'.format(s,s.split()))
- s='a-b-c-de'
- print('{:s} split={}'.format(s,s.split('-')))
- print('{:s} partition={}'.format(s,s.partition('-')))

```
a b c de split=['a', 'b', 'c', 'de']
a-b-c-de split=['a', 'b', 'c', 'de']
a-b-c-de partition=('a', '-', 'b-c-de')
```

6. 字符串类型测试

- 字符串类型测试函数返回的都是逻辑值。
- **s.isalnum()**：是否全是字母和数字，并至少有一个字符。
- **s.isalpha()**：是否全是字母，并至少有一个字符。
- **s.isdigit()**：是否全是数字，并至少有一个字符。
- **s.isspace()**：是否全是空格，并至少有一个字符。
- **s.islower()**：s中的字母是否全是小写。
- **s.isupper()**：s中的字母是否全是大写。
- **s.istitle()**：s是否为首字母大写。

- 例：
- s='Python Program'
- print('{:s} isalnum={}'.format(s,s.isalnum()))
- print('{:s} isalpha={}'.format(s,s.isalpha()))
- print('{:s} isupper={}'.format(s,s.isupper()))
- print('{:s} islower={}'.format(s,s.islower()))
- print('{:s} isdigit={}'.format(s,s.isdigit()))
- s='3423'
- print('{:s} isdigit={}'.format(s,s.isdigit()))

```
Python Program isalnum=False
Python Program isalpha=False
Python Program isupper=False
Python Program islower=False
Python Program isdigit=False
3423 isdigit=True
```

5.4 字节类型

- 在Python中，字节类型和字符串不同。
- 字符串是由Unicode字符组成的序列，用str类型符表示。
- 字节类型是由编码介于0~255之间的字符组成的序列，分为不可变字节类型和可变字节类型，分别用bytes类型符和bytearray类型符表示。

- 在字符串前面加“b”可以定义bytes对象。
- bytes对象中的每一个字符可以是一个ASCII字符或\x00-\xff的十六进制数。
- 例：
- >>> s=b'abcd\x65'
- >>> s
- b'abcde'
- >>> type(s)
- <class 'bytes'>

- 和字符串一样，可以使用内置的len()函数求bytes对象的长度，也可以使用“+”运算符连接两个bytes对象，其操作结果是一个新的bytes对象。
- 例：
- >>> s=b'abcd\x65'
- >>> len(s)
- 5
- >>> s+=b'\xff'
- >>> s
- b'abcde\xff'
- >>> len(s)
- 6

- 可以使用索引来访问bytes对象中的某一个字符。
- 对字符串做这种操作获得的元素仍为字符串。
- 而对bytes对象做这种操作的返回值则为整数。是0~255之间的整数。
- bytes对象是不可改变的，不能对其赋值。
- 例：
- >>> s=b"abcd"
- >>> s[0]
- 97
- >>> s[0]=100　# 错误操作

- 如果需要改变某个字节，可以组合使用字符串的分片和连接操作（效果跟字符串是一样的），也可以将**bytes**对象转换为**bytearray**对象，**bytearray**对象是可以被修改的。

- 例：
- >>> s=b"abcd\x65"
- >>> sarr=bytearray(s)
- >>> sarr
- bytearray(b'abcde')
- >>> len(sarr)
- 5
- >>> sarr[0]=102
- >>> sarr
- bytearray(b'fbcde')

- 使用内置函数bytearray()来完成从bytes对象到可变的bytearray对象的转换，所有对bytes对象的操作也可以用在bytearray对象上。
- 区别是：可以使用编号给bytearray对象的某个字节赋值，并且这个值必须是0~255之间的一个整数。

- bytes对象和字符串是不可以混在一起的。
- 例如，将bytes对象和字符串做连接运算就会出现错误，因为它们是两种不同的数据类型。
- 也不允许针对bytes对象的出现次数进行计数，因为字符串里面根本没有字节字符。

- 但bytes对象和字符串并不是毫无关系。
- 字符串由一系列字符组成。
- 如果想要先把这些字节序列通过某种编码方式进行解码获得字符串，然后对该字符串进行计数，则需要显式地指明它。
- Python不会隐含地将bytes对象转换成字符串，或将字符串转换成bytes对象。

- bytes对象有一个decode()方法，它使用某种字符编码作为参数，然后依照这种编码方式将bytes对象转换为字符串。
- 对应地，字符串有一个encode()方法，它也使用某种字符编码作为参数，然后依照它将字符串转换为bytes对象。

- 例：
- >>> s="我爱uestc"　#s是一个字符串
- >>> len(s)
- 7
- >>> t=s.encode('utf-8')　#t是一个bytes对象
- >>> t
- b'\xe6\x88\x91\xe7\x88\xb1uestc'
- >>> len(t)
- 11

- 例：
- >>> s="我爱uestc"　# s是一个字符串
- >>> t=s.encode('gb18030')　# t是一个bytes对象
- >>> t
- b'\xce\xd2\xb0\xaeuestc'
- >>> len(t)
- 9
- >>> m=t.decode('gb18030')　#m是一个字符串
- >>> m
- '我爱uestc'

5.5 正则表达式

- 正则表达式（**Regular Expression**），又称规则表达式。
- 正则表达式通常被用来检索、替换那些符合某个模式（规则）的文本。
- 正则表达式是一种文本模式，模式描述在搜索文本时要匹配的一个或多个字符串。

- 正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符（元字符）、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

5.5.1 正则表达式元字符

- 正则表达式由普通字符和元字符组成。
- 普通字符是正常的文本字符，具有字符的本来含义。
- 元字符（metacharacter）具有特定的含义，它使正则表达式具有通用的匹配能力。

正则表达式常用元字符

元字符	功 能 描 述
.	匹配除换行符之外的任何单个字符
*	匹配位于“*”之前的0个或多个字符
+	匹配位于“+”之前的一个或多个字符
|	匹配位于“|”之前或之后的字符
^	匹配以“^”后面的字符开头的字符串
$	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
\	表示一个转义字符
[]	匹配位于“[]”中的任意一个字符

正则表达式常用元字符

元字符	功 能 描 述
.	匹配指定范围内的任意字符
()	将位于“()”内的内容作为一个整体来看待
{}	按“{}”中的次数进行匹配
\b	匹配单词头或单词尾
\B	与“\b”含义相反
\d	匹配一个数字字符，等价于[0-9]
\D	与“\d”含义相反
\s	匹配任何空白字符，包括空格、制表符、换页符等
\S	与“\s”含义相反
\w	匹配任何字母、数字及下画线，等价于[A-Za-z0-9_]
\W	与“\w”含义相反

元字符说明

（1）排除型字符串

- 若用[^……]取代[……]，则这个字符串就会匹配任何未列出的字符。
- 例：
- r[^abc]r：将匹配除rar、rbr、rcr之外的任意r*r文本。

- 注意：在[]中，元字符不起作用，都代表字符本身的含义。
- 例：
- [akm$]将匹配“a”、“k”、“m”、“$”中的任意一个字符，在这里，元字符“$” 就是一个普通字符。

（2）规定重现次数的范围

- {n,m}大括号内的数字用于表示某字符允许出现的次数区间。
- {}里面的参数不一定要写全两个，也可以仅仅写一个，这样代表仅仅匹配指定的字符数。

- 例：

- \b{6}\b：匹配刚刚6个字母的单词。
- {n,}：匹配n次或更多次，例：{5,}匹配5次及5次以上。

（3）正则表达式举例

- 在具体应用时，元字符可以单独使用，也可以组合使用。
- 例：
- ① 匹配账号是否合法（设账号以字母开头，允许字母、数字及下画线，包括5~16个字符）：
- **^[a-zA-Z][a-zA-Z0-9_]{4,15}$**。
- ② 匹配国内电话号码：**\d{3}-\d{8}|\d{4}-\d{7}**，例：**021-87888822** 或 **0746-4405222**。

- ③匹配QQ号（设QQ号从10000开始）：[1-9][0-9]{4,}。
- ④匹配身份证（设身份证为15位或18位）：\d{15}|\d{18}。
- ⑤匹配特定数字。
  - ^[1-9]\d*$：匹配正整数。
  - ^-[1-9]\d*$：匹配负整数。
  - ^-?[1-9]\d*$：匹配整数。

- ⑥匹配特定字符串。
  - **^[A-Za-z]+$**：匹配由26个英文字母组成的字符串。
  - **^[A-Z]+$**：匹配由26个大写英文字母组成的字符串。
  - **^[a-z]+$**：匹配由26个小写英文字母组成的字符串。
  - **^[A-Za-z0-9]+$**：匹配由数字和26个英文字母组成的字符串。
  - **^\w+$**：匹配由数字、26个英文字母或者下画线组成的字符串。

5.5.2 正则表达式模块

- 在Python中，正则表达式的功能通过正则表达式模块re来实现。
- re模块提供各种正则表达式的匹配操作，在文本解析、复杂字符串分析和信息提取时是一个非常有用的工具。
- 使用re模块时，首先要导入re模块。

- 例：查询re模块的功能信息
  - >>>import re
  - >>>print(re.__doc__)
- 例：查看re模块方法
  - >>>dir(re)

1. 生成正则表达式对象

- 使用re模块的一般步骤：
- 先使用compile()函数将正则表达式的字符串形式编译为正则表达式对象，然后使用正则表达式对象提供的方法进行字符串处理，这样可以提高字符串处理的效率。

- compile()函数的调用格式：
- re.compile(pattern[,flag])
  - 参数pattern是代表匹配模式的正则表达式。
  - flag是匹配选项标志，可取的值如下：
  - re.I、re.IGNORECASE：忽略大小写。使字符串匹配时忽略字母的大小写。
- 例：正则表达式“[A-Z]”也可以匹配小写字母，“Spooc”可以匹配“Spooc”、“spooc”或“spOOC”。

- re.M、re.MULTILINE：多行匹配模式。改变元字符“^”和“$”的行为，使“^”和“$”除了匹配字符串开始和结尾外，也匹配每行的开始和结尾（换行符之后或之前）。
- re.S、re.DOTALL：匹配包括换行符在内的任意字符。改变元字符“.”的行为。使元字符“.”完全匹配任何字符，包括换行符。如没有这个标志，“.”匹配除换行符外的任何字符。

- **re.L、re.LOCALE：使预定义字符集\w、\W、\b、\B、\s、\S由当前区域设置决定。**
- **例：如果系统配置设置为法语，那么可以用预定义字符集来匹配法文文本。**
- **re.U、re.UNICODE：使预定义字符集\w、\W、\b、\B、\s、\S、\d、\D由Unicode字符集决定。**

- **re.X、re.VERBOSE：** 忽略模式字符串中的空格字符，除非被转义的空格或空格位于字符集合内（中括号内）。该方式允许用#字符添加注释直至行尾。
- 匹配模式的取值可以使用运算符“|”表示同时生效。例：**re.I|re.M**。

2. 字符匹配和搜索

- re模块提供了许多用于字符串处理的函数，这些函数有两种格式：
- （1）直接使用re模块格式
- （2）正则表达式对象格式
- 两种格式的函数参数是不同的。

- 具体使用时，既可以直接使用这些函数来进行字符串处理，也可以将正则表达式编译生成正则表达式对象，然后使用正则表达式对象的方法来进行字符串处理。
- 正则表达式对象的函数功能更为强大，是更常用的方法。

- （1）match()函数
- 如果没有生成正则表达式对象，使用match()函数可以直接进行正则表达式的匹配。
- match()函数的调用格式：
- re.match(pattern,string[,flag])
- 参数pattern是代表匹配模式的正则表达式，string是要匹配的字符串，flag是匹配选项标志，可取的值与compile()函数的匹配选项标志相同。

- match()函数从字符串的开始位置尝试匹配正则表达式。若匹配成功，则返回match对象；否则返回None。
- 可以使用match对象的group()或group(n)方法输出所有匹配的字符串（n代表数字顺序，从1开始）。
- 在对匹配完的结果进行操作之前，需要先判断是否匹配成功。

- 例：
- >>> import re
- >>> m=re.match('^[\w]{3}','abcdefg')
- >>> m
- <_sre.SRE_Match object; span=(0, 3), match='abc'>
- >>> if m:
- 　　m.group()
- 'abc'

- 还可以用简单的写法来获得所有匹配的字符串。
- 例：
- **>>> re.match('foo','food').group()**
- **'foo'**

- 如果生成了正则表达式对象，则该函数的另一种调用格式是：
- match(string[,pos[,endpos]])
- 这是正则表达式对象的方法，将从string的pos下标处尝试匹配正则表达式；
- 如果正则表达式结束时仍可匹配，则返回一个match对象；
- 如果匹配过程中无法匹配正则表达式，或者匹配未结束就已到达endpos，则返回None。

- **pos和endpos的默认值分别为0和len(string)。**
- **re.match()函数中无法指定这两个参数，参数flag用于编译正则表达式时指定匹配模式。**

- 例：
- import re
- line="Cats are smarter than dogs"
- pattern=re.compile(r'(.*) are (.*?) .*',re.M|re.I)
- matchObj=pattern.match(line)
- if matchObj:
- 　print("matchObj.group():",matchObj.group())
- 　print("matchObj.group(1):",matchObj.group(1))
- 　print("matchObj.group(2):",matchObj.group(2))
- else:
- 　print("No match!")

```
matchObj.group(): Cats are smarter than dogs
matchObj.group(1): Cats
matchObj.group(2): smarter
```

- 在**compile()**函数的正则表达式中，**r**是**raw**（原始）的意思。因为在表示字符串中有一些转义符，例如表示回车用“**\n**”。如果要表示“****”，则需要写为“****”。如果需要表示一个“****”加“**n**”，则不使用**r**方式时要写为“**\\n**”，若使用**r**方式则为“**r'\n'**”，这样清晰多了。
- 例：
- **>>> print("abc\ndef")**
- **abc**
- **def**
- **>>> print(r"abc\ndef")**
- **abc\ndef**

- 注意：
- match()方法并不是完全匹配。当正则表达式结束时，若string还有剩余字符，仍然视为成功。
- 想要完全匹配，可以在表达式末尾加上边界匹配符'$'。

- （2）search()函数
- match()函数只是在字符串的左端位置尝试匹配正则表达式，也就是只报告从位置0开始的匹配情况。
- 如果想要搜索整个字符串来寻找匹配，应当用search()函数。

- search()函数也有两种调用格式：
- re.search(pattern,string[,flag])
- search(string[,pos[,endpos]])
- 参数的含义与match()函数相同。
- 在字符串中查找匹配正则表达式模式的位置，返回match对象。
- 如果没有找到匹配的位置，则返回None。

- match()函数是只在字符串左端匹配检查的匹配，而search()函数对于一个匹配字符串中的任何位置搜索检查。

- 例：
- import re
- line="Cats are smarter than dogs"
- matchObj=re.match('dogs',line,re.M|re.I)
- if matchObj:
- print("match--
  >matchObj.group():",matchObj.group())
- else:
- print("No match!")
- searchObj=re.search('dogs',line,re.M|re.I)
- if searchObj:
- print("search--
  >searchObj.group():",searchObj.group())
- else:
- print("Nothing found!")

```
No match!
search-->searchObj.group(): dogs
```

- （3）findall()函数
- findall()函数搜索字符串，以列表形式返回全部能匹配正则表达式的子串。
- 该函数也有两种调用格式：
- re.findall(pattern,string[,flag])
- findall(string[,pos[,endpos]])
- 参数的含义与match()函数相同。

- 例：
- import re
- patt=re.compile('w{3}\.([a-zA-Z]+\.)+com')
- print(re.findall(patt,'www.baidu.com'))
- print(re.findall(patt,'www.baidu.com

  www.baidu.edu.com'))

```
['baidu.']
['baidu.', 'edu.']
```

- （4）finditer()函数
- 与findall()函数类似，在字符串中找到正则表达式所匹配的所有子串，并组成一个迭代器返回。
- 该函数也有两种调用格式：
- re.finditer(pattern,string[,flag])
- finditer(string[,pos[,endpos]])
- 参数的含义与match()函数相同。

- 例：
- import re
- p=re.compile('\d+')
- for m in p.finditer('one1 two2 three3 four4'):
- 　print(m.group())

```
1
2
3
4
```

3. 字符替换

- re模块的sub()函数、subn()函数，或正则表达式对象的同名方法，使用正则表达式匹配字符串，用指定内容替换结果，并返回替换后的字符串。

- sub()函数有两种调用格式：
- re.sub(pattern,repl,string[,count,flag])
- sub(repl,string[,count=0])
- 该函数在字符串string中找到匹配正则表达式pattern的所有子串，用另一个字符串repl进行替换。
- 如果没有找到匹配pattern的串，则返回未被修改的string。
- repl既可以是字符串，也可以是一个函数。
- count用于指定最多替换次数，不指定时全部替换。

- 例：
- import re
- p=re.compile('(one|two|three)')
- print(p.sub('num','one word two words three words',2))

```
num word num words three words
```

- subn()函数的功能和sub()函数相同，但返回新的字符串以及替换的次数组成的元组。
- subn()函数有两种调用格式：
- re.subn(pattern,repl,string[,count,flag])
- subn(repl,string[,count=0])

- 例：
- import re
- p=re.compile('(one|two|three)')
- print(p.subn('num','one word two words three words',2))

('num word num words three words', 2)

4. 字符分割

- re模块的split()函数或正则表达式对象的同名方法，使用正则表达式匹配字符串，并分割字符串，返回分割后的字符串列表。
- split()函数有两种调用格式：
- re.split(pattern,string[,maxsplit,flag])
- split(string[,maxsplit])
- 参数maxsplit用于指定最大分割次数，不指定时将全部分割。

- 例：
- import re
- r1=re.compile('\w+')
- print(r1.split('192.168.1.1'))
- print(re.split('(\w+)','192.168.1.1'))
- print(re.split('(\w+)','192.168.1.1',1))

```
['', '', '', '', '', '', '', '']
['', '192', '.', '168', '.', '1', '.', '1', '']
['', '192', '.168.1.1']
```

5. escape()函数

- escape(string)函数用于将string中的正则表达式特殊字符之前加上转义符再返回。
- 如果字符串很长且包含很多特殊字符，为避免输入大量反斜杠，可以使用这个函数。
- 例：
- >>> import re
- >>> re.escape('www.python.org')
- 'www\\.python\\.org'

5.6 字符串应用举例

- 例：输入一个字符串，每次去掉最后面的字符并输出。
- s=input()
- for i in range(-1,-len(s),-1):
- print(s[:i])

```
uestc
uest
ues
ue
u
```

- 例：翻译密码。为了保密，常不采用明码电文，而用密码电文，按事先约定的规律将一个字符转换为另一个字符，收报人则按相反的规律转换得到原来的字符。例如，将字母“A”→“F”，“B”→“G”，“C”→“H”，即将一个字母变成其后第5个字母。例如，“He is in Beijing.”应转换为“Mj nx ns Gjnonsl.”。

- 分析：
- 依次取电文中的字符，对其中的字母进行处理，对字母之外的字符维持原样。
- 取字母的ASCII代码，加上5，再按其ASCII码转换为另一个字母。
- 还有一个问题要处理，当字母为“V”时，加5就超过了“Z”，故应使之转换为“A”，同理，“W”→“B”，“X”→“C”，“Y”→“D”，“Z”→“E”。

```
line1=input()
line2=""
for c1 in line1:
    if c1.isalpha():
        i=ord(c1)
        j=i+5
        if (j>ord("z") or (j>ord("Z") and j<ord("Z")+6)):j-=26
        c2=chr(j)
        line2+=c2
else:
        line2+=c1
print(line2)
```

Windows!

Bnsitbx!

- 例：Python的标识符以字母或下画线（_）开头，后接字母、数字或下画线组成，从键盘输入字符串，判断它是否为Python的标识符。

- 分析：
- 利用string模块中的常量，包括string.digits（数字0~9）、string.ascii_letters（所有大小写字母）、sttring.ascii_lowercase（所有小写字母）、string.ascii_uppercase（所有大写字母）。
- 先输入字符串，再分别判断首字符和中间字符，并给出提示。
- 判断中间字符利用for循环遍历字符串。

```
import string
alphas=string.ascii_letters+'_'
nums=string.digits
print('Welcome to the Identifier checker 1.0')
print('Testees must be at least 2 chars long')
myInput=input('Identifier to test?')
if (len(myInput)>1):
    if myInput[0] not in alphas:
        print('''invalid:
            first symbol must be alphabetic''')
    else:
        for otherchar in myInput[1:]:
            if otherchar not in alphas+nums:
                print('''invalid:
                    remaining symbols must be alphanumeric''')
                break
        else:
            print('Ok as an identifier')
```

- 也可以构造一个正则表达式来匹配所有合法的Python标识符，程序如下：
- import re
- pattern=re.compile(r'\b[a-zA-Z_](\w|_)*\b')
- myInput=input('Identifier to test?')
- m_Obj=pattern.match(myInput)
- if m_Obj:
- print('Ok as an identifier')
- else:
- print('No as an identifier')

- 例：从键盘输入几个数字，用逗号分隔，求这些数字之和。
- 分析：输入的数字作为一个字符串来处理，首先分离出数字串，再转换成数值，这样就能求和了。

```
s=input('请输入几个数字（用逗号分隔）')
d=s.split(',')
print(d)
sum=0
for x in d:
    sum+=float(x)
print('sum=',sum)
```

- 也可以使用正则表达式来实现，程序如下：
- import re
- s=input('请输入几个数字（用逗号分隔）')
- p=re.compile(',')
- d=p.split(s)
- print(d)
- sum=0
- for x in d:
- 　　sum+=float(x)
- print('sum=',sum)

自测题

一、选择题

- 1．访问字符串中的部分字符的操作称为（ A ）。
- A．分片　　B．合并
- C．索引　　D．赋值

- 2．下列关于字符串的描述错误的是（ B ）。
- A．字符串s的首字符是s[0]
- B．在字符串中，同一个字母的大小是等价的。
- C．字符串中的字符都是以某种二进制编码的方式进行存储和处理的
- D．字符串也能进行关系比较操作

- 3．执行下列语句后的显示结果是（ A ）。

```
world="world"
print("hello"+world)
```

- A． helloworld　　B． "hello"world
- C． hello world　　D． "hello"+world

- 4．下列表达式中，有3个表达式的值相同，另一个不相同，与其他3个表达式不同的是（ C ）。
- A．"ABC"+"DEF"　　B．''.join(("ABC","DEF"))
- C．"ABC"-"DEF"　　D．'ABCDEF'*1
- 5．设s="Python Programming"，那么print(s[-5:])的结果是（ A ）。
- A．mming　　B．Python
- C．mmin　　D．Pytho

- 6．设s="Happy New Year"，则s[3:8]的值为（ B ）。
- A．'ppy Ne'　　B．'py Ne'
- C．'ppy N'　　D．'py New'
- 7．将字符串中全部字母转换为大写字母的字符串方法是（ D ）。
- A．swapcase　　B．capitalize
- C．uppercase　　D．upper

- 8．下列表达式中，能用于判断字符串s1是否属于字符串s（即s1是否s的子串）的是（ D ）。
- ①s1 in s；②s.find(s1)>0；③s.index(s1)>0；④s.rfind(s1)；⑤s.rindex(s1)>0
- A．①　　B．①②
- C．①②③　　D．①②③④⑤

- 9． re.findall('to','Tom likes to play football too.',re.I)的值是(A)。
- A． ['To', 'to', 'to']　　B． ['to', 'to', 'to']
- C． ['To', 'to']　　D． ['to', 'to']

● 10．下列程序执行后，得到的输出结果是（C）。

```
import re
p=re.compile(r'\bb\w*\b')
str="Boys may be able to get a better idea."
print(p.sub('**',str,1))
```

- A．** may be able to get a better idea.
- B．Boys may be able to get a ** idea.
- C．Boys may ** able to get a better idea.
- D．Boys may ** able to get a ** idea.

二、填空题

- 1．"4"+"5"的值是（　）。'45'
- 2．字符串s中最后一个字符的位置是（　）。

  len(s)-1
- 3．设s='abcdefg'，则s[3]的值是（　），s[3:5]的值是（　），s[:5]的值是（　），s[3:]的值是（　），s[::2]的值是（　），s[::-1]的值是（　）,s[-2:-5]的值是（　）。
- 'd'，'de'，'abcde'，'defg'，'aceg'，'gfedcba'，''

- 4．'Python Program'.count('P')的值是（　）。2
- 5．'AsDf888'.isalpha()的值是（　）。False
- 6．下面语句的执行结果是（　）。['A', 'A', 'A']

```
s='A'
print(3*s.split())
```

- 7．已知s1='red hat'，print(s1.upper())的结果是（　），s1.swapcase()的结果是（　），s1.title()的结果是（　），s1.replace('hat','cat')的结果是（　）。
- RED HAT，'RED HAT'，'Red Hat'，'red cat'

- 8．设s='a,b,c'，s2=('x','y','z')，s3=':'，则s.split(',')的值为（　），s.rsplit(',',1)的值为（　），s.partition(',')的值为（　），s.rpartition(',')的值为（　），s3.join('abc')的值为（　），s3.join(s2)的值为（　）。
- ['a', 'b', 'c']，['a,b', 'c']，('a', ',', 'b,c')，('a,b', ',', 'c'),'a:b:c','x:y:z'

- 9．re.sub('hard','easy','Python is hard to learn.')的值是（　）。
- 'Python is easy to learn.'
- 10．下列程序执行后，得到的输出结果是（　）。

  ['An', 'elite', 'university']

```
import re
str="An elite university devoted to computer
software"
print(re.findall(r'\b[aeiouAEIOU]\w+?\b',str))
```

三、问答题

- 1．什么叫字符串？有哪些常用的字符编码方案？
- 2．数字字符和数字值（如'5'和5）有何不同？如何转换？
- 3．为什么print('I like Python'* 5)可以正常执行，而print('I like Python'+5)却运行时出错？

- 4．写出表达式。
- （1）利用各种方法判断字符变量c是否为字母(不区分大小写字母)。

c.isalpha()或c.lower()<='z' and c.lower()>='a'或

c.upper()<='Z' and c.upper()>='A'或c<='Z' and c >='A'or

c<='z'and c >='a'或'A'<=c<='Z' or 'a'<=c<='z'

- （2）利用各种方法判断字符变量c是否为大写字母。

c.isupper()或者c<='Z' and c>='A'或者'A'<=c<='Z'

- （3）利用各种方法判断字符变量c是否为小写字母。
- c.islower()或者c<='z' and c>='a'
- （4）利用各种方法判断字符变量c是否为数字字符。
- c.isdigit()或c<='9' and c>='0'或'0'<=c<='9'

- **5． re.match("back","text.back")与 re.search("back","text.back")的执行结果有何不同？**