字符串基本处理和操作 (续)

- 在字符串上循环 (略)
- str 是不变序列类型，支持所有不变序列类型的公共操作 (详见后)
  - 字符串的下标操作得到单字字符串
  - 运算符 in 和 not in：判断一个字符串是否是另一字符串的子串，返回逻辑值 True / False
  - max(s), min(s)：分别得到字符串 s 中编码值最大和最小的字符做成的单字串
  - s.index(t)：得到字符串 t 首次在字符串 s 里出现的位置；如果 s 里没有 t，则报 ValueError
  - s.count(t)：统计字符串 t 在 s 里出现的次数 (从左至右，非重叠的出现)

```
>>> "abababababa".count("aba")
2
```

字符串比较

- 可用 ==, != 比较两个字符串的内容相等或者不等
  - 注意：Python 区分大小写字母
- 根据字符串上定义的字典序，两个字符串可比较大小关系
  - 字典序：基于字符集合上的序关系，即字符的编码 ord(c)
  - 字符串 s < t 的条件：顺序比较串 s、t 中各对应位置的字符
    - 设发现第一对不同字符在 i 且 ord(s[i]) < ord(t[i])；或者
    - 两个串在能比较范围内的字符都相同，但 s 较短
  - 两个串相等：两个串长度相同，对应的各字符分别相同
- 所有比较运算符 (==, !=, <, <=, >, >=) 均可用于字符串
  - 两个字符串之间比较，<, ==, > 三者之一必定成立

字符串操作 (1)

- Python 为字符串对象定义了一批特殊功能函数
  - 这些函数称为“方法”(method)，要通过点号记法使用
    - 点号记法：在字符串 (或者值为字符串的变量) 后写一个圆点符号，而后写函数名及实际参数表
  - 字符串是不变对象，部分字符串方法生成新字符串
- 生成当前串 s 的大写或小写拷贝

s.lower()	做 s 的全小写拷贝
s.upper()	做 s 的全大写拷贝
s.capitalize()	做 s 的首字符大写其余小写的拷贝
s.swapcase()	做 s 的大小写调换的拷贝

字符串操作 (2)

- 常用字符串的分类谓词 (完成判断，满足条件时返回 True)

s.isupper()	s 不空且其中所有存在大小写的字符都是大写
s.islower()	s 不空且其中所有存在大小写的字符都是小写
s.isdigit()	s 不空且其中所有字符都是数字
s.isalpha()	s 不空且其中所有字符都是字母
s.isidentifier()	s 不空且其形式可以作为标识符
s.isspace()	s 不空且其中全是空白字符 (空格、制表符、换行符)

- 其他字符串操作

s.find(sub)	参数 sub 是另一个串，查找并返回 sub 在 s 里第一次出现的位置，如果没有出现，函数返回 -1
s.find(sub, start, end)	在 s 中由 start 和 end 的指定范围里查找子串

字符串操作 (3)

s.count(sub)	统计 sub 在 s 里互不重叠的出现的次数， s.count(sub, start, end) 在指定范围内统计出现次数
s.replace(old, new)	建立字符串 s 的一个拷贝，其中把 s 里子串 old 的所有出现都替换成另一个串 new s.replace(old, new, count) 只做前 count 个替换
s.strip()	删去 s 两端的空白字符 (如果有) s.strip(chars) 删去两端出现的属于 chars 的所有字符
s.lstrip()	删去 s 左端的 (开头的) 空白字符 s.lstrip(chars) 删去左端属于 chars 的所有字符
s.rstrip()	删去 s 右端的空白字符 s.rstrip(chars) 删去右端属于 chars 的所有字符

- 注意：replace, strip, lstrip, rstrip 都是基于给定字符串 s 建立新字符串
- 更多字符串方法，参见标准库手册 4.7 节

字符串和表 (1)

- **s.split(*sep=None, maxsplit=-1*)**
  - 得到一个字符串表，其元素是切分字符串 s 后所得的一些子串
  - 形参 **sep**：指定切分方式 (分割符)
    - 如果不给 **sep** 实参，默认是由连续空白字符 (空格/换行/制表符) 切分的子串，同时丢掉开头或结尾的全部空白字符
  - 形参 **maxsplit**：指定 (从左向右处理时) 最大切分项数
    - 到达指定项数后剩下的串作为一个子串放在结果表的最后
    - 默认值 (或值为 –1)，表示要求做完整个串的切分

```
>>> "1 2 3".split()
['1', '2', '3']
>>> "  1   2   3  ".split()
['1', '2', '3']
```

```
>>> "1,2,3".split(",")
['1', '2', '3']
>>> "1,2,3".split(",", maxsplit=1)
['1', '2,3']
>>> "1,2,,3".split(",")
['1', '2', '', '3']
```

字符串和表 (2)

- **s.rsplit(*sep=None, maxsplit=-1*)** 按从右到左的顺序切分
  - 只在指定切分项数的情况下才有用 (否则操作效果同 **split**)
- **s.splitlines([*keepends*])** 得到一个字符串表，其元素是 **s** 中的正文行
  - 也就是，用换行符作为切分符，对串 **s** 进行切分
  - 如果 **keepends** 没有实参，换行符号本身不包含在切分得到子串里
  - 如果给 **keepends** 提供实参 **True**，则每行最后的换行符保留
- **sep.join(lst)**：相当于 **split** 的逆操作，用串 **sep** 作为分隔符把字符串表 **lst** 中的元素拼成一个字符串
  - **join** 方法常用来构造长串，效率比串拼接等方法更高

```
>>> ", ".join(["break", "continue", "return"])
'break, continue, return'
```

生成字符串

Ref: https://stackoverflow.com/questions/1436703/what-is-the-difference-between-str-and-repr

- 之前程序的输出都采用 **print (屏幕输出)** 产生的“自然形式”，如果需要也可以控制输出的形式
- 生成文本形式输出，第一步是把数据对象字符串化
  - 使用函数 **str** 或 **repr** 可以生成与参数对应的字符串
  - **str 函数**：用来生成参数的易读 **(readable)** 文本表示形式
    - 对于任何标准类型的对象，**str** 都能生成一个字符串表示
  - **repr 函数 (**的目标**)**：所生成的文本表示还可以重新输入，让解释器恢复原来的对象
    - 如果对象 **x** 没有可以重新输入的字符串形式，**repr(x)** 将报 **SyntaxError**
  - **print** 函数对非字符串的实参自动调用 **str**，之后加上分隔符和结束符再输出

```
>>> print(repr.__doc__)
Return the canonical string representation of the object.

For many object types, including most builtins, eval(repr(obj)) == obj.
```

简单格式化功能

- 字符串的格式化：对字符串的形式进行处理
- str 类型提供了简单的格式化方法
  - 设 s 是字符串， n 为自然数

s.center(n)	得到将 s 串居中的长度为 n 的字符串
s.ljust(n)	得到将 s 串居左的长度为 n 的字符串
s.rjust(n)	得到将 s 串居右的长度为 n 的字符串

  - 也可通过参数指定填充字符 (默认为空格)
    - 例如 s.rjust(6, '0') 要求用 '0' 填充
  - 方法的参数可以是任意表达式，从而可方便地通过变量控制格式

```
>>> for i in range(30, 80, 7):
        print(str(i).rjust(5), str(i**2).rjust(7))
```

```
   30     900
   37    1369
   44    1936
   51    2601
   58    3364
   65    4225
   72    5184
   79    6241
```

格式化操作 (1)

- 利用 str 的 format 方法可生成复杂格式的字符串
- 形式：s.format(*args, **kwargs)　　# 这里设 s 是字符串
  - s 称为格式串，是描述格式化方式 (结果串的框架/模板) 的字符串，其中可以有任意多个用 {...} 表示的位置 (称为替换域)
  - *args：表示 format 方法可以接受任意多个实参表达式
  - **kwargs：表示可以接受任意多个关键字实参
  - 返回值：按照格式串 s 处理参数后生成的字符串
    - 生成结果串时，分别用各实参的值被格式化后产生的字符串，来替代 s 中的替换域

```
>>> "The {} of 2 + 5 is {}".format("sum", 1+2)
'The sum of 2 + 5 is 3'
>>> # 生成输出时不处理数学上的正确性，只是串拼接
```

格式化操作 (2)

- 格式串本身是一个普通字符串，可以把格式串赋给变量，之后再通过变量使用
- 调用 **s.format(...)** 生成一个字符串时
  - 格式串 **s** 中，除替换域之外的字符顺序 (按原样) 拷贝到结果串，各个替换域用 **format** 的实参值生成的字符串替换
    - 格式串 **s** 描述结果的框架， **format** 的实参填充片段
  - 结果串里实际包含的花括号字符，在格式串里要用双写 **{{** 和 **}}**
- 理解格式化，需要弄清两件事：
  1. 如何用替换域描述对实际参数的格式化要求
  2. 实参与替换描述的匹配和代入关系
- 详情见标准库手册 **6.1.3** 节，有许多繁琐细节；下面只介绍格式化操作的基本规则和常用实例

格式化操作 (3)

- 实参与替换域的匹配方式

  1. 默认情况：按位置一一匹配

     "A {} is {} but {}.".format(arg_0, arg_1, arg_2)

  2. 在替换域里，用整数 (按一般的下标规则) 指定实参 (位置)

     "A {2} is {0} but {1}.".format(arg_0, arg_1, arg_2)

     - 实参顺序可以任意排列，替换域里根据需要指定实参，也可以重复使用某个 (某些) 实参

  3. 在替换域里，也可以用名字来指定实参 (即关键字实参)

     ```
     >>> "The {noun} is {adj} but also {adj2}.".format(
                 noun="pig", adj2="smart", adj="fat")
     ```

     # 产生串： "The pig is fat but also smart. "

- 默认情况下，对实参生成的字符串形式采用默认形式

格式化操作 (4)

- 控制替换字符串的生成形式：替换域中，(在整数或关键字之后) 可以有一个“:”，后跟一个转换描述，描述实参的转换方式 (注：类似于 f-string)
- 常用的转换描述项 (均可省略，如出现时须按下面的顺序)
  - 描述对齐方式的字符 <, >, 或 ^，分别表示该替换域内容采用居左, 居右或居中方式，可以在对齐字符前给一个填充字符 (默认空格)
    - 无对齐描述时，字符串采用居左对齐，数值采用居右对齐
  - 一个整数，表示本域的最小宽度，实际数据内容需要输出更多字符时可以输出得更宽；默认的输出宽度由实际数据内容确定
  - 一个字符，表示转换类型：s 表示字符串，d 表示整数用十进制方式输出，f 和 F 表示用浮点数形式输出，e 和 E 表示用科学记数法输出，g 和 G 根据情况自动采用浮点形式或科学形式
    - 默认是根据实际数据类型输出；整数可用浮点形式输出，但必须写出具体的输出形式 f/e/g 等
  - 对浮点数转换 f 和 F，可以有圆点和一个整数表示浮点数输出中小数部分的位数 (精度)，默认输出精度为 6 位

格式化操作 (5)

- 包含转换描述的替换域实例：
  - **{:<<10d}** 十进制整数形式，宽 10 字符，左对齐，填充 <
  - **{1:->10s}** 第1个实参，字符串形式，宽10，右对齐，填充 -
  - **{price:10.2f}** 域名 price，浮点形式，宽10，小数点后2位
- 注意：
  - 转换类型为 s 时，实参必须是字符串
  - 转换类型为 d 时，实参必须是整数
  - 转换类型是 f/F/e/E/g/G 时，实参可以是整数或浮点数
  - d 等整数转换类型不允许出现精度描述 (圆点加精度)
  - 非数值类型的转换中精度描述规定输出域的最大宽度
  - (更多转换描述说明参见标准库手册)

格式化操作实例

- 生成正弦和余弦函数表

```
from math import sin, cos

head = "{:^5}  {:^10s}  {:^10s}"          # 表头的输出格式
content = "{:5.3f}  {:10.8f}  {:10.8f}" # 每行的输出格式

def gen_table(start, end, step):
    print(head.format("x", "sin(x)", "cos(x)"))
    x = start
    while x < end:
        print(content.format(x, sin(x), cos(x)))
        x += step

gen_table(0.0, 0.65, 0.1)
```

```
  x       sin(x)       cos(x)
0.000  0.00000000  1.00000000
0.100  0.09983342  0.99500417
0.200  0.19866933  0.98006658
0.300  0.29552021  0.95533649
0.400  0.38941834  0.92106099
0.500  0.47942554  0.87758256
0.600  0.56464247  0.82533561
```