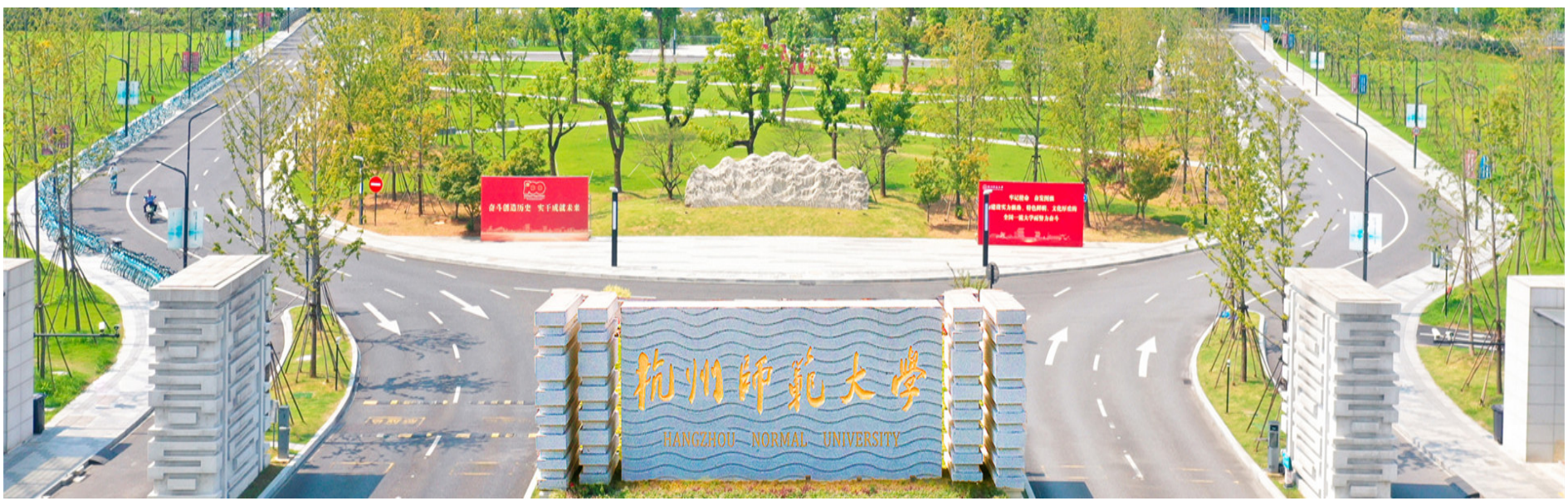
第二讲 - Python 序列

张建章

阿里巴巴商学院

杭州师范大学

2023-09



目录 I

目录 II

序列：列表 (可变)，元组 (不可变)，字符串 (不可变)，序列中的内容是有顺序的，其正向和反向查找索引如下图所示：

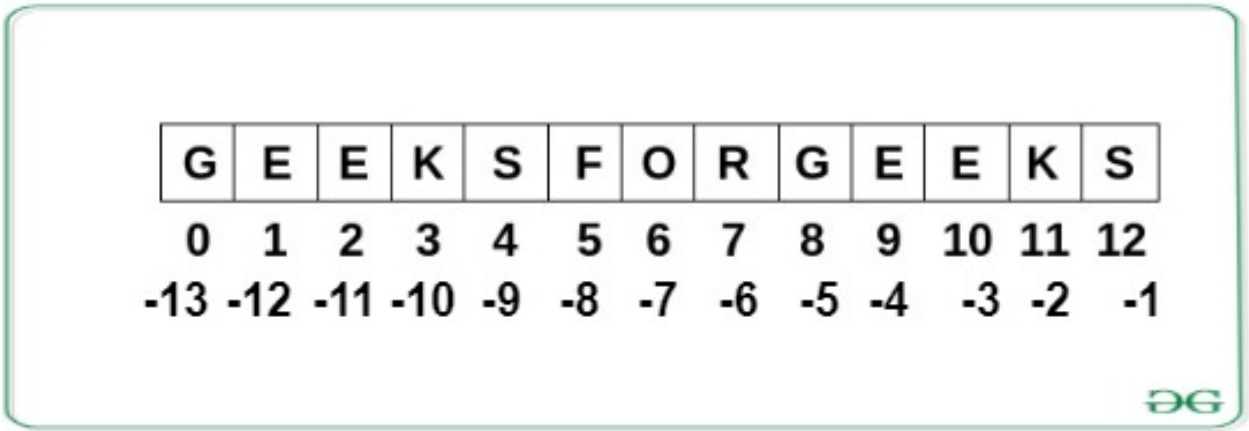


图 1: 序列的正向索引和反向索引

集合：无重复元素；

字典：存放具有映射关系的数据；

列表 (list)：一对`[]`中包含由`,`分割的一系列元素，一个列表中可以包含**任意多个 (>=0) 任意类型**的数据，每一个数据称为一个元素。

创建列表的两种常用方式：

① 最简单方法是将列表元素放在一对方括号`[]`内，各元素之间以逗号分隔，并用`=`将一个列表赋值给某个变量。如，`list1 = [1,3,5]`；

② 使用内置的`list`函数创建列表，如`list2 = list('Python')`，`list2`的内容就是列表`['P', 'y', 't', 'h', 'o', 'n']`。

注意：Python 允许同一个列表中各元素的数据类型不相同，可以是整数、字符串等基本类型，也可以是列表、集合及其他类型的对象，如`list3 = ['python', True, 99, ['good', 'boy']]`。

列表访问

可以通过索引访问列表 (长度为 n) 中的元素，索引分为**正向索引** (范围为：$0 \sim n-1$) 和**反向索引** (范围为：$-1 \sim -n$)，如下图所示：

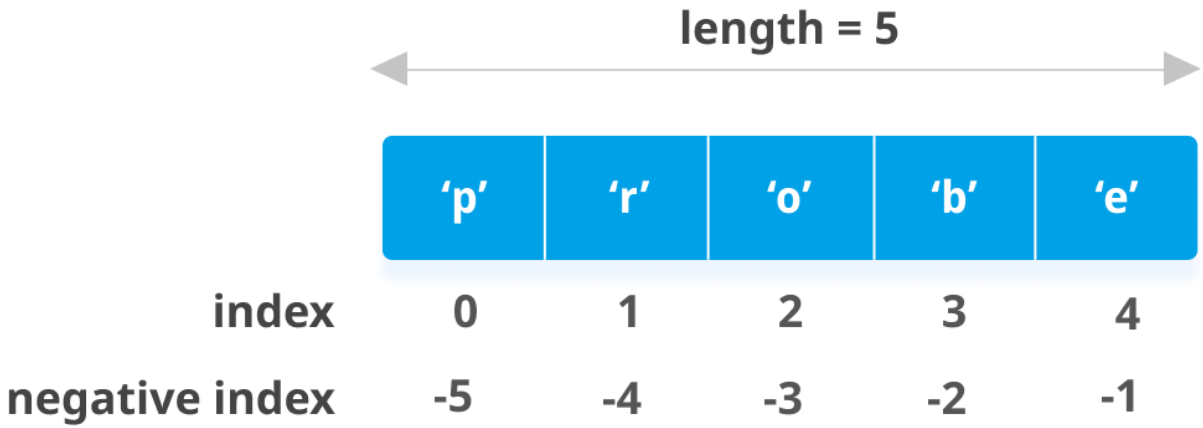


图 2: 列表的正向索引和反向索引

列表元素访问语法为`list[index]`，称为下标变量，可用于读取 (读取列表中索引 $index$ 对应的元素值) 或写入 (更改列表中索引 $index$ 对应的元素值) 列表内容。

成员判断和切片

使用 `in` 和 `not in` 运算符可以判断一个元素是否在列表中。

列表切片：使用语法 `list[start:end]` 返回列表 `list` 的一个片段，**其结果是** `list` 中索引 `start` 到 `end-1` 对应的元素所构成的**一个新列表**。

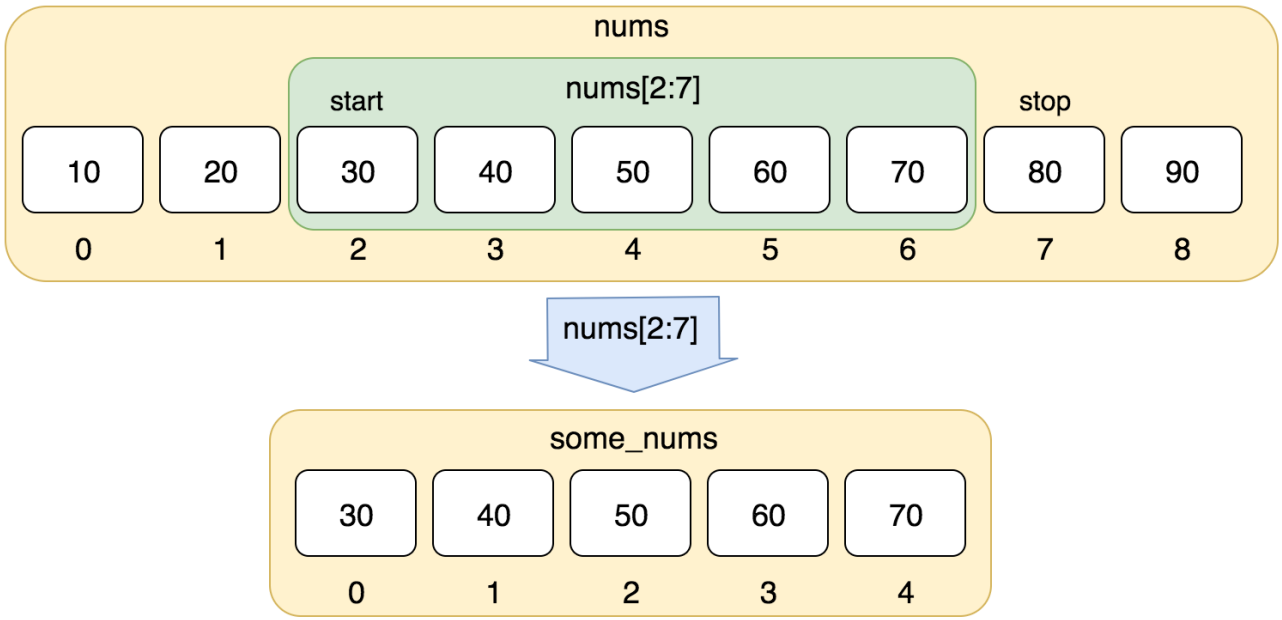


图 3: 列表的切片操作

在切片操作中，起始下标 `start` 和结束下标 `end` 可以省略：① 如省略 `start`，则起始下标默认为 0，即从列表第一个元素开始截取；② 如省略 `end`，则结束下标默认为列表长度 n，即截取到列表最后一个元素。下图中第三个参数 `-2` 表示**切片步长**，步长为正，要求从左往右切列表，步长为负，要求从右往左切列表。`list[::-1]` 可以实现列表快速翻转。

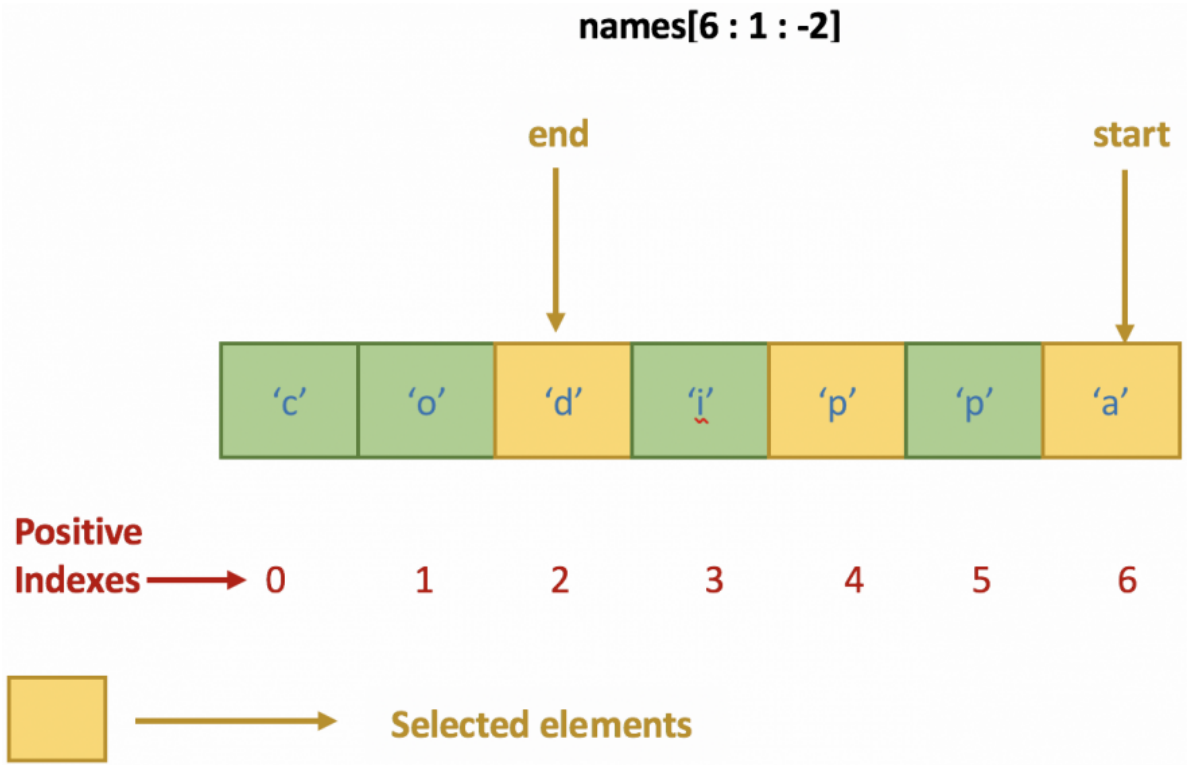


图 4: 带步长的列表的切片操作

修改列表元素值

可以通过**列表元素访问**和**切片操作**修改列表单个元素的值和同时修改多个元素的值，如下：

```
x = [1,2,3,4,5,6]

# 通过索引访问列表元素，修改单个元素值
x[2] = 100 # [1,2,100,4,5,6]

# 通过列表切片，同时修改多个元素值
x[-1:-3] = [200, 300] # [1, 2, 100, 4, 300, 200]
```

因为列表切片的结果是**一个列表**，所以上面切片赋值操作中等号右侧的内容一定是一个列表。

通过切片扩展、删减和复制列表

```
x = [1, 'a', 'b', 'c', 5, 6]

# 扩展列表
x[1:1] = [7, 8, 9] # [1, 7, 8, 9, 'a', 'b', 'c', 5, 6]

# 删减列表
x[1:4] = [] # [1, 'a', 'b', 'c', 5, 6]

# 复制列表
y = x[:] # [1, 'a', 'b', 'c', 5, 6]

# 引用赋值-(不推荐使用)
z = x # [1, 'a', 'b', 'c', 5, 6]

# 判断三个变量的值以及指向的内存空间是否相同
x == y == z # True
x is y # False
x is z # True
```

列表的拼接和复制

Python 中，使用 `+` 号拼接两个列表，并返回一个新列表，使用 `*` 复制一个列表若干次，返回一个新列表，如下：

```
x = [1, 2, 3]

y = x*3 # [1, 2, 3, 1, 2, 3, 1, 2, 3]

x[0] = 0 # 复制返回的是新列表，故y中的元素保持不变

z = x*2 + y # [0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

# 复制和拼接都是返回新列表，故z中的元素保持不变
x[0] = 111
y[0] = 222

# [0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
print(z)
```

列表的比较

使用关系运算符 `<, >, ==, <=, >=, !=` 对比较两个列表，规则为：

① 比较两个列表的第 0 个元素，如果两个元素相同，则继续比较下面两个元素；

② 如果两个元素不同，则返回两个元素的比较结果；

③ 一直重复这个过程直到有不同的元素或比较完所有的元素为止(长的列表大于短的列表，如下面的 `x < z`)。

```
x = [1, 2, 3, 4, 5]
y = [1, 2, 3, 100]
z = [1, 2 ,3 ,4 ,5, 6]
x < y # True
x < z # True
# 比较两个字符的Unicode码值，ord('阿') > ord('a')，因此结果为True
xstr = ['阿', 'z']
ystr = ['a', '里']
xstr > ystr # True
```

列表推导式

列表推导式是一个生成**新列表**的简洁方法。一个列表推导式由**方括号**括起来，方括号内包含**一个表达式**和**至少一个 for 循环语句**，之后可以**接 0 到多个 for 或 if 子句**。列表推导式可以产生一个由表达式求值结果作为元素的**新列表**。

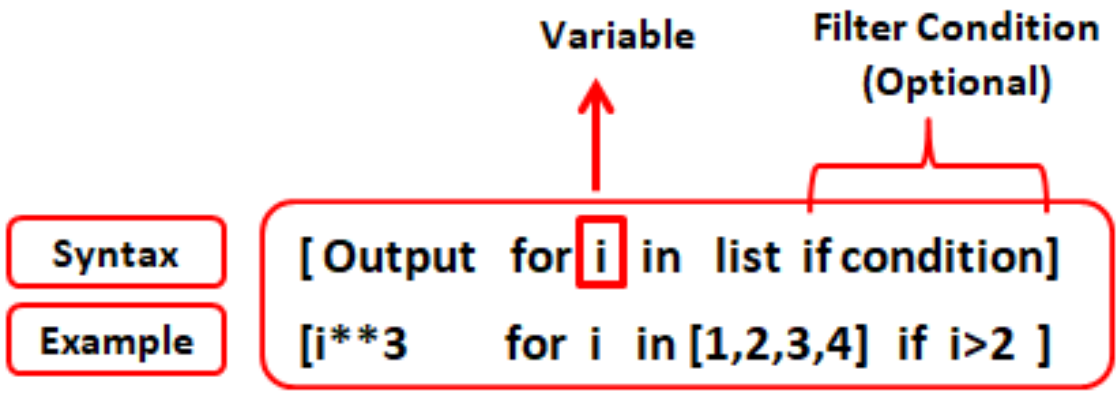


图 5: 列表推导式

- `all(iterable)`，全部元素为 True，返回 True；
- `any(iterable)`，有一个元素为 True，返回 True；
- `list(s)`，将可迭代对象 s 变为列表；
- `len(s)`，计算列表的长度；
- `max(iterable)`，返回列表中最大的元素；
- `min(iterable)`，返回列表中最小的元素；
- `sorted(iterable[, cmp[, key[, reverse]]])`，对列表进行排序；
- `sum(iterable[, start])`，对列表进行求和；
- `reversed(iterable)`，翻转列表。

上述内置函数不仅可以用于操作列表，还可用于操作其他可迭代对象，使用 `help` 函数可查看其详细用法，如 `help(all)`。

- `list.append(x)`，在列表末尾增加一个元素 x；
- `list.extend(L)`，在列表末尾再拼接一个列表 L；
- `list.insert(i, x)`，在索引为 i 的位置插入一个元素 x；
- `list.remove(x)`，从列表中移除元素 x；
- `list.pop(i)`，删除索引 i 对应的元素；
- `list.index(x)`，返回元素 x 对应的索引值；
- `list.count(x)`，计数元素 x 在列表中出现的次数；
- `list.sort(key=None, reverse=False)`，对列表进行排序；
- `list.reverse()`，将列表翻转。

上述常用的列表方法**只能用于**操作列表，使用 `help` 函数可查看其详细用法，如 `help(list.append)`，`help(list)` 查看列表的全部方法。

- **基本用法：** 操作列表的内置函数的用法为`function_name(list)`，列表方法的用法为`list.method_name(parameters)`；
- **结果:**操作列表的内置函数不改变列表的内容,如`reversed(list)`其返回结果是一个新的列表，而列表`list`不发生变化，列表方法会直接改变表列表的内容，如`list.reverse()`执行后，列表`list`中的元素将会翻转，并不会返回一个新列表。

注意： 当元素 x 在列表`list`中多次出现时，`list.remove(x)`只能移除第一个 x(索引最小的那个 x)，`list.index(x)`只能返回第一个 x 的索引值 (x 对应的最小索引值)。

多维列表：列表中嵌套列表，比较常用的是二维列表 (矩阵) 和三维列表 (张量)。数值 (矩阵、张量) 运算中通常使用科学计算包 `numpy` 中的 `array` 数据类型来表示多维数值列表。

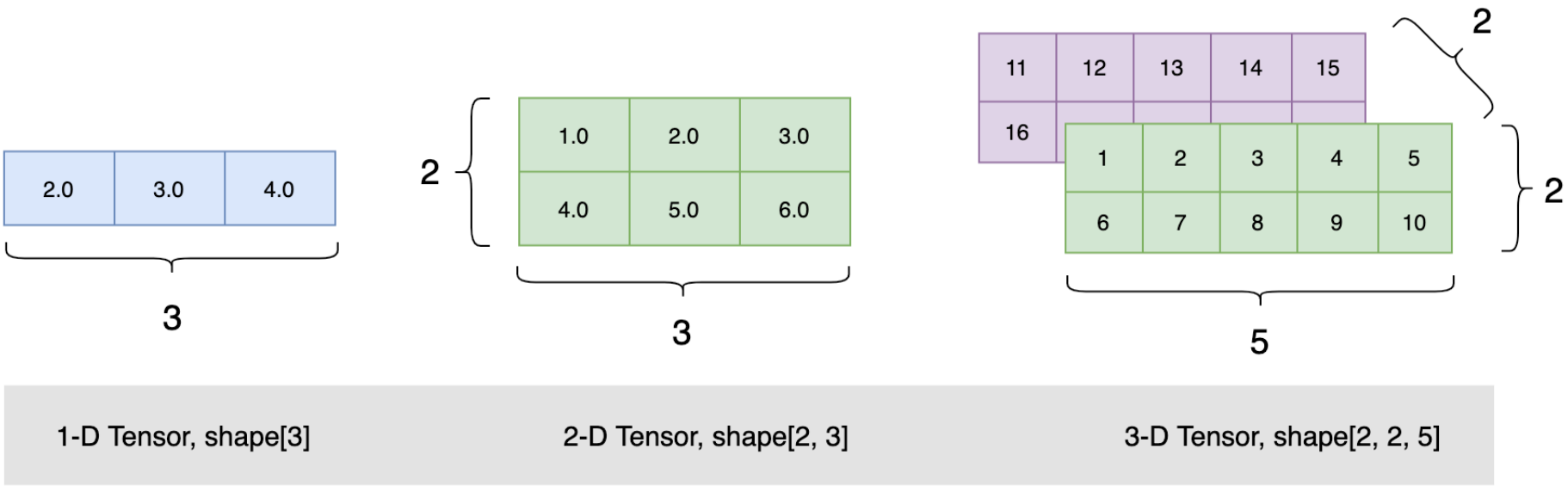


图 6: 多维数值列表 (张量)

访问多维列表中的元素：① 使用多重 `for` 循环遍历每个元素，② 使用多级索引访问某个元素，`tensor3[1][1][3]` 可访问上图最右侧三维列表中的元素 19。

- 多维列表本质也是一个列表，可通过 `type` 函数可查看其数据类型，其包含的元素也是列表；
- 前述的操作列表的内置函数和列表的方法也适用于多维列表；
- 多维列表**可以表示**数学上的张量，也可以表示其他数据，如二维表(数据库表，简单 Excel 表格)，其每一个维度值是灵活的，如下例：

```
# 下面的多维列表表示了下图中矩阵的左上角
left_top = [[1, 2, 3], [4, 5], [7]]
```

1	2	3
4	5	6
7	8	9

元组 (tuple)：一对 `()` 中包含由 `,` 分割的一系列元素，一个元组中可以包含**任意多个 (>=0) 任意类型**的数据，每一个数据称为一个元素，**元组中的元素不可变**。

创建元组的两种常用方式：

① 最简单方法是将元组元素放在一对圆括号 `()` 内，各元素之间以逗号分隔，并用 `=` 将一个元组赋值给某个变量。如，`tuple1 = (1,3,5)`；

② 使用内置的 `tuple` 函数创建元组，

如 `tuple2 = tuple('Python')`，`tuple2` 的内容就是元组 `('P', 'y', 't', 'h', 'o', 'n')`。

注意：Python 允许同一个元组中各元素的数据类型不相同，可以是整数、字符串等基本类型，也可以是列表、集合及其他类型的对象，如 `tuple3 = ('python', True, 99, ['good', 'boy'])`。

正确理解元组中的数据元素不可变

① 元组一旦创建，其长度不可变，不能删减、增加元素，不能重新赋值元素；

② **不能重新赋值元素**：元组中的元素是不可变数据类型时，该元素不可变 (重新赋值)，如 1-Introduction.pdf 中介绍过的基本数据类型 (整数、浮点数、字符串、布尔值、空值)；如果元素是可变数据类型，则可以改变该元素的值，但是不可以改变该元素的数据类型 (重新赋值)，具体看下面示例：

```
tuple4 = (1, 2, [1, 2, 3])

tuple4[2].append(4) # (1, 2, [1, 2, 3, 4])

# Error: 'tuple' object does not support item assignment
tuple4[2] = True

# Error: 'tuple' object does not support item assignment
tuple4[0] = 'good'
```

元组相关操作

可以使用 `tuple` 函数将列表、字符串等转换为元组；元组也是序列，一些用于列表的基本操作也可以用在元组上，如索引访问、成员判断、切片等。

```
# 使用tuple函数将列表转换为一个元组
list_1 = [1, 'Lisa', True, 6.6]
tuple_2 = tuple(list_1)

# 使用tuple函数将字符串转换为一个元组
string_1 = '阿里巴巴商学院'
tuple_3 = tuple(string_1)

# 判断元素是否在元组中
'巴' in tuple_3
# 使用索引访问元组中的元素
tuple_3[0]
# 对元组进行切片
tuple_3[1:3]
```

元组封装：将多个值自动封装到一个元组中；

序列拆封：将一个封装起来的序列自动拆分为若干个基本数据。

为多个变量同时赋值：结合了元组封装和序列拆封操作。

`a, b, c, d = 4, 'Tony', True, 6.6` 等价于 `my_tuple = 4, 'Tony', True, 6.6` 和 `a, b, c, d = my_tuple`

```
# 元组封装

tuple_4 = 4, 'Tony', True, 6.6 # (4, 'Tony', True, 6.6)

# 元组拆封
a, b, c, d = tuple_4

# 列表拆封
list_2 = [8, 'Dan', False, 8.8]
a, b, c, d = list_2
```

相同点：均为序列数据类型，除元素修改操作外，其他操作，如索引访问、计数、切片等，两者用法相同；

不同点：元组属于不可变序列，一旦创建，便不允许进行元素的增加、删除和重新赋值，因此，元组没有`append`、`extend`、`insert`方法；列表是可变序列，可以对其中元素进行增删改操作；

应用场景不同：元组的访问和处理速度比列表更快。因此，如果所定义的序列内容不会进行修改，最好使用元组而不是列表。另外，使用元组也可以使元素在实现上无法被修改，从而使代码更加安全。

标准的序列操作同样适用于字符串：

- 索引，切片
- 乘法，加法
- 成员资格判断
- 求长度，最大值，最小值

```
str1 = '喜欢看你紧紧皱眉 叫我胆小鬼'
str1[-3:] # 胆小鬼
'皱眉' in str1 # True
len(str1), max(str1), min(str1) # (14, '鬼', ' ')
'鲁班' + '大师' # 鲁班大师
'娜可' + '露'*2 # 娜可露露
```

字符串不可变，因此单独赋值或切片赋值是不合法的。

什么是字符串格式化

字符串格式化本质上就是一个字符串模板，用于**高效 (模板的功能性)**、**美观 (模板的装饰性)** 地生成可打印的字符串。

高效：在字符串格式化语句中，公共的、不变的部分只书写一次，可变的部分使用字符串格式化方法进行动态填充，例如下面使用字符串格式化打印青蛙跳水儿歌，可变部分是与青蛙数量相关的数量值，其他部分不变。

```
# num取值从1-9
for num in range(1,10):
  str_tmp = '{}只青蛙{}张嘴，{}只眼睛{}条腿，'.format(num, num,
  ↪  2*num, 4*num) + '扑通~'*num + '{}声跳下水'.format(num)
  print(str_tmp)
```

输出结果见下页，27-28 页上的例子都是在展示字符串格式化高效 (模板的功能性) 的方面，第 28 页打印歌手的字符串格式化的一种应用场景是“为音乐颁奖典礼制作介绍获奖者信息的台词”

什么是字符串格式化 (续)

1只青蛙1张嘴，2只眼睛4条腿，扑通~1声跳下水
2只青蛙2张嘴，4只眼睛8条腿，扑通~扑通~2声跳下水
3只青蛙3张嘴，6只眼睛12条腿，扑通~扑通~扑通~3声跳下水
4只青蛙4张嘴，8只眼睛16条腿，扑通~扑通~扑通~扑通~4声跳下水
5只青蛙5张嘴，10只眼睛20条腿，扑通~扑通~扑通~扑通~扑通~5声跳下水
6只青蛙6张嘴，12只眼睛24条腿，扑通~扑通~扑通~扑通~扑通~扑通~6声跳下水
7只青蛙7张嘴，14只眼睛28条腿，扑通~扑通~扑通~扑通~扑通~扑通~扑通~7声跳下水
8只青蛙8张嘴，16只眼睛32条腿，扑通~扑通~扑通~扑通~扑通~扑通~扑通~扑通~8声跳下水
9只青蛙9张嘴，18只眼睛36条腿，扑通~扑通~扑通~扑通~扑通~扑通~扑通~扑通~扑通~9声跳下水

图 7: 使用字符串格式化高效打印青蛙跳水儿歌

什么是字符串格式化 (续)

美观：字符串格式化使用一系列**格式说明符**指定所填充内容的显示格式，例如，下面的例子就用字符串格式化整齐美观地 (模板的装饰性) 打印出一批商品的价格表。

```
width = int(input('Please enter width: '))
price_width = 10
item_width  = width - price_width
header_fmt = '{{:{}}}{{:>{}}}'.format(item_width, price_width)
fmt        = '{{:{}}}{{:>{}.2f}}'.format(item_width, price_width)
print('=' * width)
print(header_fmt.format('Item', 'Price'))
print('-' * width)
print(fmt.format('Apples', 0.4))
print(fmt.format('Pears', 0.5))
print(fmt.format('Cantaloupes', 1.92))
print(fmt.format('Dried Apricots (16 oz.)', 8))
print(fmt.format('Prunes (4 lbs.)', 12))
print('=' * width) # 50
```

什么是字符串格式化 (续)

```
Please enter width:  50
==================================================
Item                                        Price
--------------------------------------------------
Apples                                       0.40
Pears                                        0.50
Cantaloupes                                  1.92
Dried Apricots (16 oz.)                      8.00
Prunes (4 lbs.)                             12.00
==================================================
```

图 8: 使用字符串格式化整齐美观地打印出价格表

总结：字符串格式化的语法包括两部分：① 占位符 (`%` 或 `{}`)，指定在哪里填充内容，不可省略；② 格式说明符，指定填充内容的显示格式，可以省略。如果只使用字符串格式化的功能性，可以不写格式符，只使用占位符即可，如果也使用字符串格式化的装饰性，那就需要根据需求指定格式说明符。

字符串格式化：用**元组内的元素**按照**预设格式**替代字符串中的内容。

两种方式：使用 `%`；使用 `format` 方法。

```
num = 3
str2 = '%s只青蛙%s张嘴，%s只眼睛%s条腿，' %(num, num, 2*num, 4*num)
↪  + '扑通~'*num + '%s声跳下水' %(num)
print(str2)
num = 5
str3 = '{}只青蛙{}张嘴，{}只眼睛{}条腿，'.format(num, num, 2*num,
↪  4*num) + '扑通~'*num + '{}声跳下水'.format(num)
print(str3)
```

`%` 为占位符，**标记了**需要插入值的**位置**。格式符号 `s` 表示插入的值会被格式化为字符串，如果不是字符串会自动转换为字符串 (如，上例中插入的 num 为整数)。

`format` 方法可以按照从左到右的顺序依次将元组中的值插入到占位符 `{}` 所在的位置。

Python2.6 开始，新增的字符串格式化方法 `format`，增强了字符串格式化的功能。

```
# 1. 按照索引进行替换(推荐)
singer_template = "歌手{3}，{0}年出生于{1}，代表作有《{2}》等。"
print(singer_template.format(1978, '新加坡', '胆小鬼', '孙燕姿'))
# 2. 按照字段名进行替换(推荐)
singer_template = "歌手{name}，{birth_year}年出生于{nation}，
↪  代表作有《{masterpiece}》等。"
print(singer_template.format(birth_year = 1963, nation =
↪  '中国天津市', masterpiece = '好汉歌', name = '刘欢'))
# 3. 字段名和索引可以混用，元组中位置参数(索引)在前，
↪  关键字参数(字段名)在后
singer_template = "歌手{name}，{0}年出生于{nation}，代表作有《{1}》
↪  等。"
print(singer_template.format(1991, '光年之外', nation =
↪  '中国上海市', name = '邓紫棋'))
```

`format` 方法中的占位符 `{}` 的完整形式为 **{字段名或索引: 字段格式}**，冒号前后的内容均可省略，若都省略，则将元组中的内容以字符串格式进行显示。

```
# /表示填充字符，如果不指定默认就是空格填充；
# ^表示居中显示
# +表示在正数前面加正号，负数前面加负号
# 10表示宽度为10
# .2f表示保留两位小数
str4 = "{:/^+10.2f}"
str4.format(3.1415926) # //+3.14///
```

标准格式说明符如下，`[]` 表示其中的参数是可选的:

```
[[fill]align][sign][#][0][width][grouping_option][.precision][type]
```

图 9: 类比游戏中的装备格理解标准格式说明符

① 游戏中装备格子有限，每个装备格子不一定非得买装备；格式说明符类型也有限 (数数上面有几对中括号)，每类格式说明符不一定被指定；

② 游戏中有出装顺序；标准格式说明符也有顺序呀 (上面从左到右就是顺序)；

③ 游戏中可以一件装备也不出 (没输出、不抗揍、跑得慢)；格式说明符也可以一个也不指定 (没什么装饰性了)；

④ 游戏中不带惩戒不能买打野刀；格式说明符里不指定 `align` (对齐方式) 就不能指定 `fill` (填充字符)。

align（对齐方式）	作用
<	左对齐（字符串默认对齐方式）
>	右对齐（数值默认对齐方式）
=	填充时强制在正负号与数字之间进行填充，只支持对数字的填充
^	居中

图 10: align 参数

除非定义了最小字段宽度，否则字段宽度将始终与填充它的数据大小相同，因此在这种情况下对齐选项没有意义。

`fill` 定义用于将字段填充到最小宽度的字符。填充字符 (如果存在) **必须后跟对齐标志**。

`sign` 参数仅对数字类型有效：

sign	作用
+	强制对数字使用正负号
-	仅对负数使用前导负号(默认)
空格	对正数使用一个' '作前导，负数仍以'-'为前导

图 11: sign 参数

如果存在 `#` 字符，则整数使用“替代形式”进行格式化。这意味着二进制、八进制和十六进制输出将分别以“0b”、“0o”和“0x”为前缀。

`width` 是定义最小字段宽度的十进制整数。如果未指定，则字段宽度将由内容决定。

如果 `width` 前面有一个零 `0` 字符，则启用零填充。这等效于对齐类型为“=”且填充字符为“0”。

`grouping_option` 用来对数字整数部分进行千分位分隔。

描述符	作用
,	使用,作为千位分隔符
_	使用_作为千位分隔符

图 12: grouping_option 参数

`.precision` 是一个十进制数，表示在浮点转换中小数点后应显示多少位。对于非数字类型，该字段指示最大字段大小 - 换句话说，从字段内容中将使用多少个字符。**整数不允许使用精度**。

`type` 决定了数据应该如何呈现，默认值为 `s`，以字符串格式显示。

对于 `type` 参数，可用的整数表示类型有：

- `b`，以二进制显示；
- `c`，显示数字对应的 unicode 字符；
- `d`，以十进制显示；
- `o`，以八进制显示；
- `x` 或 `X`，以十六进制显示。

对于 `type` 参数，可用的浮点数表示类型有：

- `e` 或 `E`，用科学计数法表示，使用 E 或 e 表示指数；
- `f` 或 `F`，用浮点数表示，对于 nan(not a number) 和 inf(无穷大) 用大 (小) 写表示；
- `g` 或 `G`，自动在科学计数法和浮点数表示中做出选择，指数 E(e) 大小写取决于 G(g)；
- `%`，对原数值乘以 100 后，加上 `%` 作后缀。

字符串格式化其他参考资料：资料 1，资料 2，资料 3，

```
str5 = "{:^10.5s}的成绩是{:<+10.2f}，班级排名第{:d}"
str5.format("张三", 91.5, 5) #     张三     的成绩是+91.50    ，
↪  班级排名第5
str6 = "{num:+e}的二进制形式为{num:+#b}，八进制为{num:+#o}，
↪  十六进制为{num:+#X}"
str6.format(num=456) # +4.560000e+02的二进制形式为+0b111001000，
↪  八进制为+0o710，十六进制为+0X1C8
str7 = "{:5.3s}的年收入为{:5d}元，月平均{:5.2f}元"
str7.format('zjzhang', 60023, 60023/12.0) # zjz
↪  的年收入为60023元，月平均5001.92元
str8 = "{:5.3s}的年收入为{:4d}元，月平均{:5.2f}元"
str8.format('zjzhang', 60023, 60023/12.0) # zjz
↪  的年收入为60023元，月平均5001.92元
str8.format('Tomy', 67, 67/12.0) # Tom  的年收入为  67元，月平均
↪  5.58元
```

① 字符串的精度是对字符串的直接切片 (看上例中的 str7)；

② 浮点数精度存在**四舍五入**，首先保证小数部分，当长度超过宽度时，最初设置的宽度将不起作用 (看上例中的 str7)；**整数不允许使用精度**；

③ 默认对齐方式，字符串左对齐，数值右对齐 (看上例中的 str8)。

内部自转义：`{{` 表示 `{`，`}}` 表示 `}`，`%%` 表示 `%`

f-string：字符串格式化的最新方法，在形式上以 f 或 F 修饰符引领的字符串 (f'xxx' 或 F'xxx')，以大括号 {} 标明被替换的字段，可在 {} 内调用变量、函数和表达式求值。

```
# {}你看，我左边是什么
'{{}}{text}'.format(text = '你看，我左边是什么')
# %是百分号
'%%%s'%('是百分号')
import math
radius = 3
# 有一个圆，半径为3.00，面积为28.27
f"有一个圆，半径为{radius:.2f}，面积为{math.pi*(radius**2):.2f}"

x = 80
# 角度值为80.00，弧度值为1.39626，其正弦值为0.98
F"角度值为{x:.2f}，弧度值为{math.radians(x):.5f},
↪  其正弦值为{math.sin(math.radians(x)):.2f}"
```

可以使用内置函数 `help` 查看字符串的全部方法，命令为 `help(str)`。

```
# find函数返回子串在字符串中第一次出现的索引位置，没找到则返回-1
str9 = "My heart will go on and on"
str9[14:20] # 'go on '
str9.find('on', 14, 20) # 17, 即使指定查找范围，
↪  返回的索引值仍然是在整个字符串中的索引值
# index函数返回子串在字符串中第一次出现的索引位置，
↪  没找到则代码报错substring not found
str9.index('on', 14, 20) # 17 即使指定查找范围，
↪  返回的索引值仍然是在整个字符串中的索引值
str9.index('on', 0, 8) # 没找到，报错
# startswith，endswith分别用于判断字符串是否以特定字符为开头和结尾
str9.startswith('My') # True
str9.endswith('and',0,-3) # True
# count函数计算子串在字符串中出现的次数
"My heart will go on and on".count('n',-9) # 3, satrt和end可以省略
```

`startswith`，`endswith`，`find`，`index` 均可以用索引值指定查找的范围。`find` 找不到返回-1，`index` 找不到则报错。

常见的 4 个空白字符有: `\t`， `\r`， `\n`，空格。

```
# center函数令字符串居中，并指定宽度和填充字符
"回到最初我们来的地方".center(20,'~') #
↪  ~~~~~回到最初我们来的地方~~~~~
# split函数将字符串分割为列表，默认分隔符为空白字符
# join函数将字符串列表组合为字符串，默认连接符为空字符，
↪  空字符就是两个引号之间啥也没有，不是空格，也不是空白字符
str9.split(' ') # ['My', 'heart', 'will', 'go', 'on', 'and',
↪  'on']
'~'.join(str9.split(' ')) # My~heart~will~go~on~and~on
# 再来看一个结合字符串格式化的例子
num_str = '1 2 \r 3\t4\n9'
num_list = num_str.split() # ['1', '2', '3', '4', '9']
'*'.join(num_list) # 1*2*3*4*9
print(f"{'*'.join(num_list)}的计算结果为: {1*2*3*4*9}") #
↪  1*2*3*4*9的计算结果为: 216
# strip方法去除字符串开头和结尾的指定字符，默认去除空白字符
"let's gooooooooo".strip('o') # let's g
" \tAlibaba\r \n".strip() # Alibaba
```

其他常用的字符串方法如下:

```
# lower和upper分别将字符串中的全部字符变为小写和大写
'My heart'.lower() # 'my heart'
'My heart'.upper() # 'MY HEART'
# replace将字符串中的特定字符替换为指定字符
my_str = 'so good'
my_str.replace('o','~') # s~ g~~d, my_str内容不变
'goooood'.replace('ooo','') # good
# translate方法根据定义的转换表转换字符串的字符
# 1. 定义转换表
intab = "aeiou"
outtab = "12345"
trantab = str.maketrans(intab, outtab)
# 2. 使用转换表进行批量替换
str10 = "this is string example....wow!!!"
# th3s 3s str3ng 2x1mpl2....w4w!!!
print(str10.translate(trantab))
```

使用字符串的下列方法**判断字符串是否满足特定的条件**：isalnum、isalpha、isdecimal、isdigit、isidentifier、islower、isnumeric、isprintable、isspace、istitle、isupper，请自行敲代码学习这些方法。

`string` 模块包含了许多常用的字符集：

```
import string
string.digits # 0123456789
string.punctuation # 打印出所有英文标点符号
```

```
whitespace -- 包含所有 ASCII 空格的字符串
ascii_lowercase -- 包含所有 ASCII 小写字母的字符串
ascii_uppercase -- 包含所有 ASCII 大写字母的字符串
ascii_letters -- 包含所有 ASCII 字母的字符串
数字 -- 包含所有 ASCII 十进制数字的字符串
hexdigits -- 包含所有 ASCII 十六进制数字的字符串
octdigits -- 包含所有 ASCII 八进制数字的字符串
punctuation -- 包含所有 ASCII 标点字符的字符串
printable -- 包含所有被认为可打印的 ASCII 字符的字符串
```

图 13: string 模块中常用的字符集变量

Python 的内置函数 `str` 将对象转化为人类可读的字符串表示，`repr` 将对象转化为 Python 解释器读取的字符串表示，两个函数返回的结果都是字符串 (string) 类型。

当字符串作为两者的参数时，repr 返回的结果会多一层引号。`eval` 函数接受一个字符串作为参数，其作用是把字符串中的内容作为 Python 命令运行，并返回运行结果 (即，剥离引号 + 运行命令)。

```
s = '1*2*3*4'
str(s) # '1*2*3*4'
repr(s) # "'1*2*3*4'"
eval(str(s)) # 24
eval(repr(s)) # '1*2*3*4'
```

要在字符串中插入“非法字符”，需使用转义字符。非法字符的一个示例是字符串中的双引号被双引号括起来 (看下面代码示例)，其他常用的转义字符主要是空白字符 (回车、换行、制表符)。在字符串的引号前面加上字符 `r` 可消除字符串中的转义 (如存在)。

```
str1 = "\"是一个双引号"
print(str1)
print('Hello\nProgrammers')
print(r'Hello\nProgrammers')
print("\"是一个双引号") # \"是一个双引号
```

转义字符	说明
\n	换行符，将光标位置移到下一行开头。
\r	回车符，将光标位置移到本行开头。
\t	水平制表符，也即 Tab 键，一般相当于四个空格。
\a	蜂鸣器响铃。注意不是喇叭发声，现在的计算机很多都不带蜂鸣器了，所以响铃不一定有效。
\b	退格（Backspace），将光标位置移到前一列。
\\	反斜线
\'	单引号
\"	双引号
\	在字符串行尾的续行符，即一行未完，转到下一行继续写。

THE END