

关于 Python 的注意事项

X

2025 年 9 月 21 日

1 转义字符

表 1: Python 转义字符示例

符号	说明
\\	反斜杠\
\'	单引号'
\"	双引号"

2 原始字符串

利用 `r"text"` 可以使后文以文本直接显示，不进行转义操作

3 print 操作内部换行

使用 `\n` 再加上 `\` 换行，如果你嫌同一行太长，也可以使用 `\` 将同一行的字隔断，隔断后，下一行打印时自动接在上一行

4 长字符

利用 `“text”` 或 `”text”` 显示长字符串，长字符串段内换行有效且不会报错，转义会正常进行

5 算符顺序

表 2: 运算符优先级从低 (1) 到高 (17)

优先级	运算符	描述
1	lambda	Lambda 表达式
2	if - else	条件表达式
3	or	布尔“或”
4	and	布尔“与”
5	not x	布尔“非”
6	in, not in, is, is not, <, ≤, >, ≥, ≠, =	成员测试, 同一性测试, 比较
7		按位或
8	^	按位异或
9	&	按位与
10	<<, >>	移位
11	+, -	加法, 减法
12	*, @, /, //, %	乘法, 矩阵乘法, 除法, 地板除, 取余数
13	+x, -x, ~x	正号, 负号, 按位翻转
14	**	指数
15	await x	Await 表达式
16	x[index], x[index:index], x(arguments...), x.attribute	下标, 切片, 函数调用, 属性引用
17	(expressions...), [expressions...], {key:value...}, {expressions...}	绑定或元组显示, 列表显示, 字典显示, 集合显示

6 符号

<=号中<号和=号必须挨在一起

7 while 循环语句

break 操作¹可以直接结束 while 循环，或者 while 循环未达条件时也会自动退出

8 浮点数

8.1 浮点数的精确计算

利用 decimal 模块可以使浮点数精确，操作如下：

```
import decimal
x = decimal.Decimal('欲输入的数字')
```

之后以此类推，将所有浮点数这样表示即可进行精确运算

8.2 浮点数的科学表示

使用 aEb 表示 $a \cdot 10^b$

9 虚数

用 $x = a + bj$ 来表示，以浮点数形式储存，x.real 表示实部，x.imag 表示虚部

¹break 操作会结束离它最近的循环

10 运算

表 3: python 运算操作示例

操作	结果
<code>x % y</code>	取余
<code>x // y</code>	向下取整
<code>abs(x)</code>	绝对值或复数的模
<code>int(x)</code>	将 x 转换为整数
<code>float(x)</code>	将 x 转换为浮点数
<code>complex(re, im)</code>	返回一个复数，re 是其实部，im 是其虚部
<code>c.conjugate()</code>	返回 c 的共轭复数
<code>divmod(x, y)</code>	返回 x 除以 y 的整除结果以及余数
<code>pow(x, y)</code> 或 <code>x**y</code>	x 的 y 次方

注意：以上类型可以对字符串进行操作，但是字符串表示一个复数时，中间不能有空格，否则会报错

11 bool 类型

11.1 输出为 False 的内容

False²，None，值等于 0 的数字，空序列及集合

11.2 not 逻辑运算符

not 逻辑运算符会输出与 not 后内容相反的 bool 类型的值

²False 类型输出值可能为下文所述的 False，None，各种 0，空阔号 `○`，空字符串、序列或集合，但值都为 0

12 短路逻辑，针对 and 和 or 算符

12.1 and

and 运算符打印最前面的能决定 bool 类型 True 或 False 的变量

例：bool 类型为 True，打印最后一个变量；bool 类型为 False，打印 False

12.2 or

or 运算符打印第一个非 False 类型的变量，若全为 False 则打印 False

13 分支结构 (三元运算符)

形如以下的分支：

变量 = 值1 if 条件 else 值2

即为三元运算符。

注意值 1 与值 2 可以是任意可行的变量类型，可以理解为 = 号后是一个整体，先进行这个整体的判断，得到结果（值 1 或值 2），再将结果赋值给变量。

14 input 语句注意事项

input("text") 语句输入的内容默认是字符串，要进行运算，必须先转换为 int，float 等数字类型

15 换行

15.1 默认换行

输出换行符为 (\n)，print 函数会在输出末尾自动添加 (\n)，导致后续输出换行，另外 print() 操作可以理解为不输入任何内容的换行操作

15.2 修改换行命令

通过end参数修改结尾字符（默认值为“\n”），实现自定义分隔或不换行，比如：

```
print('text', end = '')
```

实现不换行的命令；或者

```
print('text', end = '1')
```

使得在每一个输出后添一个 1，但不进行换行操作；又或者

```
print('text', end = '1 \n')
```

使得在每一个输出后添一个 1，并进行换行操作

16 range 生成序列

注意 range 是左包含右不包含的，且输出为 int 类型，有以下三种类型：

表 4: range 操作

写法	意义
range(a)	[0,a) 间隔 1 取一个
range(a,b)	[a,b) 间隔 1 取一个
range(a,b,x)	[a,b) 间隔 x 取一个

需要注意的是：a, b, x 必须为 int 类型

17 循环中的 else

值得注意的是，这里的 else 是与 for 或 while 并列，只有在循环正常结束（没有 break 打断）时执行（从某种意义上讲，else 中的语句在循环寿终正寝，不被打断才执行）

17.1 for-else 循环

语法结构及意义如下：

```
for 变量 in 可迭代对象:  # 循环体（处理每个元素）
    if 条件:
        statement
        break  # 打断循环（此时else不执行）
    else:
        statement  # 循环正常结束（无break）时执行
```

17.2 while-else 循环

语法结构及意义如下：

```
while 条件:  # 循环体
    if 条件:
        statement
        break  # 打断循环（此时else不执行）
    else:
        statement  # 循环条件不成立（无break）时执行
```

18 列表

18.1 列表的构建以及元素的获取

18.1.1 单个元素

我们构建如下列表：

```
list = [element_0, element_1, element_2, ....., element_n-1]
```

当提取该序列（列表）的某个元素时，我们用

```
list[x]
```

来提取序列的第 $x+1$ 个元素³（即 list 中的element_x），需要注意的是，list 中的序列从element_0开始，而不是element_1

³类似的，所有可迭代序列都可以用这种方法得到序列中特定的元素

18.1.2 元素位置的获取

我们用

```
list.index(element, start, end)
```

来获得从 start 到 end 的第一个 element 元素的下标索引

18.1.3 切片

我们可以用

```
list[x: y: a]
```

来获取 list 中从第 x+1 个元素（包含）到第 y+1 个元素（不包含）每 a 个元素取一个所构成的子序列（又名“切片”）

需要注意的是：

- 1.x 的默认值为 0，y 的默认值为序列的元素数，a 的默认值为 1，最后的:a 可以不写
2. 若 a, x, y 同为负数就按照相应的规则倒序获取即可

18.1.4 列表修改 1——增

我们用

```
list.append(element)
```

在列表末尾添加一个元素⁴；或者用

```
list.extend(iterable)
```

在列表末尾添加一组可迭代对象⁵；也可以用

```
list.insert(location, element)
```

来对某个位置插入一个元素

⁴注意这里仅能添加一个元素

⁵iterable 的意思是可迭代对象，例如数组，列表，字符串等

18.1.5 列表修改 2——删

我们用

```
list.remove(element)
```

来删除某个元素 (element)，注意，这里只会删除列表中的第一个 element 元素；或者用

```
list.pop(number)
```

来删除序号为 number 的元素；也可以用

```
list.clear()
```

来直接清空 list 列表

18.1.6 列表修改 3——改

我们用

```
list[start: end: step] = iterable
```

修改列表元素，注意这里的替换包含了 start，不包含 end，当 $start = end$ 时，相当于在 start 位置前添加可迭代对象（也可以理解为对 list 的切片进行赋值），或者用

```
list[number] = element
```

来将 list 中下标为 number 的元素更改为元素 element，结合上方 index 指令的内容，我们可以用

```
list[list.index(element)] = element*
```

将第一个 element 元素换成 element* 元素

18.2 列表的“运算”

18.2.1 “加法”

“加法”操作：

```
list1 + list2
```

将 list2 的序列接到 list1 后，形成新列表

18.2.2 “乘法”

“乘法”操作：

```
list * number
```

是将 list 中元素重复 number 次，形成新列表

18.2.3 拷贝

我们用（浅拷贝）

```
list1 = list.copy()
```

或者

```
list1 = list[:]
```

又或者

```
import copy
```

```
list = [.....]
```

```
list1 = copy.copy(list)
```

来实现对整个列表外围对象的拷贝（外围内存位置改变，但是内层如果有嵌套对象，内层列表的对象会指向相同的内存位置）⁶

如果要对列表完全的拷贝（修改原对象不会对之后的列表造成影响），我们可以用 `copy` 模块的 `deepcopy` 函数，操作如下：

```
import copy

matrix = [.....]
matrix1 = copy.deepcopy(matrix)
```

18.2.4 列表推导式

基本语法为：

```
list = [expression for target in iterable if condition]
```

将其转换为 `for` 循环语句为：

```
list = []
for target in iterable:
    if condition:
        list.append(expression)
```

当然，如果是直接将原列表改为列表推导式的列表，那就需要将原列表再赋值为 `list`；我们可以表示嵌套的列表推导式如下：

```
list = [expression for target1 in iterable1 if condition1
        for target2 in iterable2 if condition2
        ...
        for targetN in iterableN if conditionN]
```

用 `for` 循环语句来解释就是：

⁶使用 `list1 = list` 只实现了对内存位置的拷贝

```

list = []
for target1 in iterable1:
    if condition1:
        for target2 in iterable2:
            if condition2:
                ...
                for targetN in iterableN:
                    if conditionN:
                        list.append(expression)

```

18.3 嵌套列表

18.3.1 嵌套列表的构建

以如下方式构建一个嵌套列表：

```
matrix = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

当然，也可以写作：

```
matrix = [[0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]
```

或者，我们用

```
A = [0] * 3
```

```
for i in range(3):
    A[i] = [0, 0, 0]
```

来构建一个初始列表

18.3.2 嵌套列表的读取

我们用：

```
for i in matrix:  
    for each in i:  
        print(each)
```

来读取列表中的所有元素，或者用

```
matrix[i]
```

来读取第 $i+1$ 行的列表，或者用

```
matrix[i][j]
```

来读取第 $i+1$ 行第 $j+1$ 列的元素

18.4 其他注意

18.4.1 列表长度的获取

利用`len(list)`函数可以得到列表长度

18.4.2 倒数元素的获取

可以用

```
list[-x]
```

来得到列表的倒数第 x 个元素

18.4.3 列表元素的排序与转置

使用

```
list.sort()
```

指令使列表数字从小到大排序，同时，

```
list.reverse()
```

可以使列表转置，两者结合，可以写作：

```
list.sort(reverse = True)
```

不难看出，sort 操作中 reverse 指令默认为 False

19 元组

元组⁷表示为：

```
tuple = (element_0, element_1, element_2, ....., element_n)
```

除了不能修改外，其他诸如查找、切片功能都可以使用⁸⁹

20 获取字符的 Unicode 编码

使用 ord(某单个字符) 来获得该字符的 Unicode 编码

21 序列类型的解包

我们可以这样操作¹⁰：

```
x_0, x_1, ..., x_n-1 = iterable
```

来让x_0到x_n-1分别对应 iterable 中的 n 个元素¹¹

⁷注意，一元元组必须表示成 tuple = (elementsingle,)，否则会被认成数字或字符串

⁸需要注意的是，列表推导式是坚决不能用来生成元组的，不然为什么叫“列表”推导式呢

⁹另外还要注意的，元组只是不能修改其指向的内存位置，但如果其指向的内存改变了，元组也实际上发生了一定的改变

¹⁰注意这里 iterable 是 n 元的

¹¹若 iterable 的元素大于 n 个，我们可以在 x_n-1 左上角加上 * 号，形如：*x_n-1，来表示后面所有元素组成的一个列表

22 保留标识符

保留标识符（关键字）是语言预先定义的具有特殊含义的单词，不能用作变量名、函数名或其他标识符，比如：if, and, for...，可以使用 keyword 模块的 iskeyword 函数来判断，操作如下：

```
import keyword

keyword.iskeyword(str)
```

23 字符串

23.1 换大小写字母

表 5: 字符串大小写转换操作及功能

操作	功能描述
capitalize()	将字符串的首字母大写，其余字母小写
casefold()	将字符串转换为小写形式（支持特殊字符转换和多语言场景）
title()	将字符串中每个单词的首字母大写
swapcase()	将字符串中的大小写互换（大写变小写，小写变大写）
upper()	将字符串所有字符转换为大写形式
lower()	将字符串所有字符转换为小写形式

23.2 左中右对齐

表 6: 字符串对齐与填充操作功能

操作	功能描述
center(width, fillchar=' ')	将字符串居中，使用指定字符（fillchar 的内容，默认是空格，下同）填充两侧至目标宽度
ljust(width, fillchar=' ')	将字符串左对齐，使用指定字符填充右侧至目标宽度
rjust(width, fillchar=' ')	将字符串右对齐，使用指定字符填充左侧至目标宽度
zfill(width)	在字符串左侧填充零至目标宽度（负数符号保留）

23.3 查找

表 7: python 中的查找方法

方法	功能	未找到行为	方向
count()	统计出现次数	返回 0	全局
find()	从左到右查找某个元素, 返回首次找到的索引值	返回 -1	左 → 右
rfind()	从右到左查找某个元素, 返回首次找到的索引值	返回 -1	右 → 左
index()	从左到右查找某个元素, 返回首次找到的索引值	抛出异常	左 → 右
rindex()	从右到左查找某个元素, 返回首次找到的索引值	抛出异常	右 → 左

注意,可以在括号里加入 start,end 来规定范围(依旧是左闭右开),比如:count(element, start, end)

23.4 替换

表 8: 替换

方法名	作用描述
expandtabs([tabsize=8])	将字符串中的制表符 (\t) 扩展为一定数量的空格, 默认制表符大小为 8 个空格, 可通过参数指定。用于文本格式化和处理。
replace(old, new, count=-1)	在字符串中替换指定的子字符串, 'old' 为要被替换的子字符串, 'new' 为替换的新字符串, 'count' 指定替换次数, 默认-1 表示替换所有出现的 'old' 子字符串。用于修改和清理字符串内容。
translate(table)	根据给定的转换表 ('table') 对字符串进行字符替换, 转换表通常为字典等数据结构, 指定字符间的映射关系。常用于批量字符替换或字符编码转换。

其中, 对 translate 举例:

```
x = x.translate(str.maketrans(str1, str2, str3))
```

这个操作的意义是: 先删除 x 中形如 str3 的字符, 再将其他字符根据 str1 与 str2 的对应法则进行更换

23.5 判断（返回布尔类型的值）

表 9: Python 字符串判断方法汇总

方法	功能说明
前缀/后缀检查	
startswith(prefix, start, end)	检查字符串是否以指定前缀开头，可选参数限定检测范围
endswith(suffix, start, end)	检查字符串是否以指定后缀结尾，可选参数限定检测范围
字符类型判断	
isupper()	检测所有字符是否均为大写字母
islower()	检测所有字符是否均为小写字母
istitle()	检测是否符合标题格式（每个单词首字母大写）
isalpha()	检测是否仅包含字母字符
isascii()	检测是否均为 ASCII 字符
isspace()	检测是否均为空白字符（空格/制表符/换行等）
isprintable()	检测是否均为可打印字符
isdecimal()	检测是否均为十进制数字（0-9）
isdigit()	检测是否均为数字（包括 Unicode 数字）
isnumeric()	检测是否均为数值字符（含分数/罗马数字/中文数字等）
isalnum()	检测是否均为字母或数字组合
isidentifier()	检测是否符合变量命名规则（即将此作为变量名是否合法）

注意：

1. 前两个方法可以在括号内输入元组，这样的话只要有一个成立，布尔类型就为真
2. 需要留意 `isalpha()` 方法中，空格不是字母字符，因此有空格会返回 `False` 的值
3. `isspace()` 方法中，只要能打印且打印出来是空的，均为空白字符，如：制表符，换行符等
4. `isprintable()` 方法中，不可打印字符是指在显示或打印时不会产生可见输出（如文本或符号）的字符，如：换行符、制表符等，而字母、数字、标点符号、空格被认为可打印
5. `isdigit()` 方法当且仅当字符串非空且所有字符都是数字时返回 `True`，其他情况一律返回 `False`，比如：'2²' 会返回 `True`
6. `isnumeric()` 方法只要是数字就行（英文 one, two, three 等不算数字）

23.6 截取

表 10: Python 字符串处理函数

函数	功能
strip(str)	移除字符串 两端 空白/指定字符
lstrip(str)	移除字符串 左端 空白/指定字符
rstrip(str)	移除字符串 右端 空白/指定字符
removeprefix(prefix)	移除字符串 开头 的指定前缀
removesuffix(suffix)	移除字符串 结尾 的指定后缀

需要注意的是，前三个方法是从某个方向开始对字符串的每个元素一一比对是否有 str 中的字符，若有，就删，然后进入下一个字符，直到没有就停止；而后两个是直接比对字符串前后位置是否与指定前后缀相同，若相同，则执行删除操作，若不符，则不进行操作

23.7 拆分

表 11: Python 字符串分割方法核心特性对比

方法	分割次数	方向	可选参数	注意事项
partition(sep)	1 次	→	无	<ul style="list-style-type: none">• 未找到分隔符时返回 (原字符串, "", "")• 保留分隔符在结果中• 输出三元组
rpartition(sep)	1 次	←	无	
split(sep, maxsplit)	多次	→	sep=None maxsplit=-1	<ul style="list-style-type: none">• sep=None 时按空白 (即空格键) 分割 (智能去除首尾空白)• maxsplit 控制最大分割次数• 不保留分隔符在结果中• 输出表格

还有 `splitlines()` 的方法, 将文字以行划分, 输出表格, 可选参数为 `keepend = True/ False`, 决定是否保留换行符 (默认为 `False`)

23.8 拼接

我们用 `join()` 方法将字符串拼接, 操作如下:

```
str_result = str.join(iterable)
```

输出结果是将 `iterable` 里的元素¹²依次排列, 并在每两个中间插入字符串 `str`

¹²注意, 这里 `iterable` 里面的元素必须是字符串, 如果有其他类型的变量 (如: `int`, `float` 等) 必须转化为字符串类型, 可选方法有二: 一是用列表推导式或者 `list()` 函数将 `iterable` 转化成列表, 二是用 `map(str, iterable)` 直接变成 `str` 类型

23.9 格式化字符串

23.9.1 初等

使用 `format()` 方法可以将字符串中花括号内空缺内容填充¹³，形式如下：

```
str (中间含{}) .format(tuple)
```

这里必须满足 `{}` 数不得多于 `tuple` 的元素数，否则会报错；另外，要想输出 `{str1}` 可以用关键字参数 `+'{str1}'` 或者直接在 `str` 中使用 `{{str1}}`

23.9.2 进阶

`format()` 方法有很多可选参数，可以表示成如下的“式子”：

```
formatted_string = '{:{fill}{align}{sign}({#})(0){width}{grouping_option}.{precision}{type}}'.format(str, fill
```

其中，可选字符的效果如下：

表 12: `format` 方法可选参数详解

参数	描述
fill	指定填充字符（默认空格），如 <code>:a<5</code> \rightarrow <code>"hello"</code> \rightarrow <code>"helloa"</code>
align	对齐方式： <code><</code> （左）、 <code>></code> （右）、 <code>^</code> （中）
sign	符号显示： <code>+</code> （总显示）、 <code>-</code> （仅负）、空格（正数留空）
#	添加进制前缀（ <code>0x/0o/0b</code> ），如 <code>{:x}</code> \rightarrow <code>0xff</code>
0	用 <code>0</code> 替代空格填充，如 <code>:05d</code> \rightarrow <code>42</code> \rightarrow <code>"00042"</code>
width	最小输出宽度，如 <code>:10</code> \rightarrow <code>"hello"</code> \rightarrow <code>" hello"</code>
grouping_option	数字分组（如千位分隔，可选，和 <code>_</code> ）， <code>:.2f</code> \rightarrow <code>1234.56</code> \rightarrow <code>"1,234.56"</code>
precision	浮点数精度，如 <code>:.3f</code> \rightarrow <code>3.14159</code> \rightarrow <code>"3.142"</code>
type	数据类型： <code>s</code> （字符串）、 <code>d</code> （整）、 <code>f</code> （浮点）、 <code>b/o/x</code> （进制）

¹³注意，这里面花括号仅能选择一个 `number` 参数（用以指示使用 `tuple` 里的哪个元素）或者一个关键字参数（比如 `name`，然后在 `tuple` 里写上 `name = x`，用来专门指代），且不能留有任何空格；另外，若两种参数同时出现，那么位置参数必须在关键字参数之前，否则就会出错

注意：

1. fill 的内容必须为单位长度的字符串或 0123456789 中的一个数字

2. 为 align, sign 赋值的内容必须是字符串

当然，也可以使用 f-string 的方法来简化代码：

```
formatted_string = f'{{str:{{fill}}{{align}}{{sign}}(##)(0){{width}}{{grouping_option}}{{precision}}{{type}}.{{precision}}{}}
```

注意这也要有一个 {}

24 序列

24.1 运算符

24.1.1 is 运算符

is 运算符用以指向两个变量是否指向同一个内存位置¹⁴（is not 与之相反）

同一个字符串只在一个内存中储存，一样的一张列表会储存在不同的内存位置

而对于直接将某个列表乘以 n 的情况，实际上是复制其内存位置；而用列表推导式则不会出现这个问题

24.1.2 in 运算符

判断某个元素是否包含于另一个序列

24.1.3 del 运算符

del 运算符可以用来执行删除操作，可以直接删除变量，如：del x就直接删除了变量 x，也可以用于删除列表中的元素，写法和切片类似（效果就是把切片赋值为空）

24.2 函数

24.2.1 min(),max()

字如其名，可以直接输入参数，或者比较 iterable 的元素大小，比如：min(iterable) 注意字符串是比较其编码值

当然，有时无法返回结果，我们可以写 min(iterable, default = str)，这样，无法输出对象时，就会输出 default 里面的值

¹⁴想调用内存位置可以用函数 id(target)

24.2.2 `sum(iterable, start = number)`

`start` 意为从 `number` 开始加

24.2.3 `sorted(iterable, key = function, reverse = True/False)`

注意，`list.sort()` 会直接改变 `list`，而 `sorted(list)` 会形成一个新的列表；`sort()` 只能处理 `list`，`sorted()` 可以处理一切 `iterable`，并传出 `list`

24.2.4 `reversed(iterable)`

注意这个会传出一个迭代器，要想查看需将这个转化成 `list`, `tuple`, `string` 等可视化对象

24.2.5 `all()`,`any()`

用来判断 `iterable` 里面的元素是否全为真/是否存在真

24.2.6 `enumerate()`

操作：

```
enumerate(iterable, start = number)
```

结果（迭代器¹⁵，需要转换成 `list`, `tuple`, `string` 的变量来查看）：

```
(number, element_1), (number + 1, element_2), ... （以此类推，这些都是二元元组）
```

24.2.7 `zip()`

`zip()` 函数将几个可迭代对象的第 `i` 个元素打包成第 `i` 个元组传出（迭代器，需要转换成 `list`, `tuple`, `string` 的变量来查看），比如：

```
x = [1, 2, 3, 4, 5]
y = [1, 1, 4, 5, 1]
z = [3, 7, 8, 1, 0]
```

```
zipped = list(zip(x, y, z))
```

¹⁵迭代器只能使用一次，用过即消失，想让可迭代对象变成迭代器，可以使用 `iter(iterable)` 函数，将其转化为迭代器

会输出：

```
[(1, 1, 3), (2, 1, 7), (3, 4, 8), (4, 5, 1), (5, 1, 0)]
```

注意，如果几个 iterable 的元素数目不同，则在最短序列时停止并输出结果，要是想用那些被忽略的值，可以使用 itertools 模块的 `zip_longest()` 函数，操作如下：

```
import itertools
```

```
itertools.zip_longest(x, y, z, ..., fillvalue = number/str)
```

会将空缺处以 `fillvalue` 填充，默认值是 `bool` 类型的 `None`

24.2.8 map(function, iterable)

对所有 iterable 的元素依次进行 function 操作，然后形成一个可迭代对象（迭代器，需要转换成 list, tuple, string 的变量来查看）

24.2.9 filter(function, iterable)

与 `map()` 类似，但只返回结果为真的变量，其他删了不要

24.2.10 next(iterator, str)

将迭代器中的元素一一拿出来，注意，每次只返回一个变量，下次返回下一个，不输入 `str` 时，没有变量时会出现异常，若输入可控变量 `str`，则会返回 `str`

24.3 将可迭代对象转化为某几种序列的函数

可以使用 `list()`, `tuple()`, `str()` 函数将可迭代对象分别转化为列表，元组和字符串

注意：前两个互换可以理解为改变括号，前两个变成 `str` 是直接两边加引号，而 `str` 变成前两个是将每个字符取出来作为 list 或者 tuple 里的元素

25 字典

25.1 概述

字典是一种键值对存储结构，形如：

```
dictionary = {key1: value1, key2: value2, ...}
```

需要注意，key 的值互不相同；可以通过 key 的值获得相应的 value：

```
dictionary[keyN] = valueN
```

这个操作同样可用于修改 key 所对应的 value 值，若 keyN 不存在，则会将该键值对写入

25.2 字典的构建

25.2.1 直接构建

```
dictionary = {key1: value1, key2: value2, ...}
```

这里 value 的数据类型没有限制，而 key 的类型是不可变类型（字符串、数字、元组等）

25.2.2 利用 dict() 函数来构建

第一种方法和上面一样：

第二种（有点像赋值）是：

```
dictionary = dict(key1 = value1, key2 = value2, ...)
```

特别注意 key 的内容必须为能在加引号后变为字符串的一串字符且不能为数字（函数知识）；第三种（类似于传入元组）是：

```
dictionary = dict((key1, value1), (key2, value2), ...)
```

然后是先用 zip() 函数构建成打包的元组再传入：

```
dictionary = dict(zip(iterable1, iterable2))
```

25.3 操作

25.3.1 del(), clear(), len(), in, list(), iter()

与之前相同；注意，list(), iter() 只会对 key 进行操作

25.3.2 几个方法

表 13: Python 字典核心方法

方法	功能
<code>fromkeys(keys, value)</code>	创建新字典，所有 key 的值对应为 value
<code>pop(key, default)</code>	删除指定键，若没有，返回 default
<code>popitem()</code>	删除最后插入的键值对
<code>update(dictionary)</code>	批量更新字典
<code>get(key, default)</code>	查找 key 对应的 value 值，没有就传出 default
<code>setdefault(key, default)</code>	若字典中有 key，不响应；若无 key，形成 (key, default) 的键值对
<code>keys()</code>	返回字典所有键的动态视图对象，实时反映字典变化
<code>values()</code>	返回字典所有值的动态视图对象，实时反映字典变化
<code>items()</code>	返回字典所有键值对（元组形式）的动态视图对象，实时反映字典变化

25.4 字典的嵌套

将 value 赋值成一个字典即可，查看操作与列表相同

25.5 字典推导式

```
dictionary = {key: value for element in iterable if condition}
```

或者：

```
dictionary_ = {key_: value_ for key, value in dictionary.items() if condition}
```

26 集合

26.1 集合的构建

```
set = {element1, element2, ...}
```

或者用 `set()` 函数：

```
set = set(iterable)
```

这里会构建一个 iterable 所有元素的集合（会自动去除重复的元素）。注意，集合有无序性，因此不能用下标索引。

这里的集合是可变对象，要想构建一个不可变集合，可以使用 frozenset() 方法，用法与 set() 相同。

注意，集合也有推导式

26.2 判断

表 14: Python 集合核心方法

方法	核心功能描述
isdisjoint(other)	检查两集合是否无交集（无共同元素）
issubset(other)	检查当前集合是否为另一集合的子集
issuperset(other)	检查当前集合是否为另一集合的超集
union(*others)	返回当前集合与其他集合的并集
intersection(*others)	返回当前集合与其他集合的交集
difference(*others)	返回当前集合与其他集合的差集
symmetric_difference(other)	返回当前集合与其他集合的对称差集

其它快捷方法：

表 15: 快捷方案

方法	核心功能描述
<, >, =	检查两集合包含关系
setA setB	取并集
setA & setB	取交集
setA - setB	取差集
setA ^ setB	取对称差集

26.3 其他操作

26.3.1 update() 操作

有 `intersection_update()`, `difference_update()`, `symmetric_difference_update()` 方法，相当于将原来的集合赋值给判断后输出的集合

26.3.2 add(element), remove(element), discard(element), pop()

`set` 中没有 `element` 时，`remove()` 会报错，而 `discard()` 不会；`pop()` 会随机删一个元素

27 函数

27.1 写法

```
def function(parameter1, parameter2, ...):  
    content
```

调用时就写：

```
function(parameter1, parameter2, ...)
```

注意，以上 `parameterN` 可以写作 `parameterN = ...`，这也叫做关键字参数。写函数时，关键字参数必须要出现在位置参数后面（可以用 `*` 来隔断）；当然，也可以通过这个给函数提供默认值

27.2 不确定元素个数的输入方式

```
def function(*tuple_name, parameter1, parameter2, ...):  
    content
```

注意，这里 `tuple_name` 的意思是传入的参数以元组的形式储存，输入时仍正常输入即可；后面的参数必须用关键字参数

27.3 函数中字典的构建

```
def function(**keywords):  
    content
```

注意，这里要想调用，必须写作：

```
function(key1 = value1, key2 = value2, ...)
```

这里 key 参数若为字符串，不能加引号

27.4 return

作用：

1. 返回结果：将函数内部计算或处理的结果传递给外部调用者
2. 终止函数执行：一旦执行 return，函数会立即结束运行，后续代码不再执行；若无 return，代码自动返回 None

27.5 作用域

在函数内部定义的变量仅能在该函数内部作用，不能在函数外部出现；而在函数外部出现的变量没有限制

需要注意的是，如果一个变量同时在函数内外出现，则内外互不影响（若内部未进行赋值等操作，内部就调用外部的值；若进行操作，就使用内部操作的结果，函数运行完毕后，变量的值变回原来的值）

但是如果想在函数里面定义或者改变一个全局变量，可以使用 global 语句：

```
def function():  
    global variation  
    content
```

但不提倡这样做

27.6 嵌套函数

```
def functionA():  
    content  
    def functionB():  
        content  
    content
```

注意,不能在外部直接调用 functionB,要想使用 functionB 必须将调用语句写在 functionA 里面,嵌套函数中,变量升一级用 nonlocal 语句

27.7 LEGB 法则

作用 “‘效果’”: local > enclose > global > build-in, 意思是: 当前后可能冲突时, 会优先选择 “更大的” 来进行作用

27.8 闭包

通过闭包操作, 可以将一个变量单独的储存在某个包内, 避免外部参数对其的影响, 一般写法为:

```
def functionA(parameters):  
    content (包含定义局域参数parameter_x的语句)  
  
    def functionB(parameters):  
        content (包含: nonlocal parameter_x语句)  
        return ...  
  
    content  
    return functionB  
  
fun = functionA(parameters)  
  
fun(parameters)
```

这里，我们每次调用 `fun` 时，相当于是在使用 `functionB` 进行操作，而内置于 `functionB`（引用 `functionA` 的 `parameter_x`）的参数没有被删除，可以经过上一个值来进行重复使用；如果将 `functionA` 多次赋值给不同的参数，里面的 `parameter_x` 互不影响

28 * 的用法

28.1 解包

对 `list`, `tuple` 使用一个 `*` 号，对字典使用 2 个 `*` 号并且只会将 `key` 参数解包

28.2 表示不确定元素个数的输入方式

见函数章节