

# Python Day 5

Mittwoch, 16. September 2020 01:11

## 字符串

### 1. 字符串的定义

- Python 中字符串被定义为引号之间的字符集合。
- Python 支持使用成对的 单引号 或 双引号。

【例子】

```
In [38]: t1 = 'i love Python!'
print(t1, type(t1))
# i love Python! <class 'str'>

t2 = "I love Python!"
print(t2, type(t2))
# I love Python! <class 'str'>

print(5 + 8) # 13
print('5' + '8') # 58

i love Python! <class 'str'>
I love Python! <class 'str'>
13
58
```

- Python 的常用转义字符

转义字符	描述
\\	反斜杠符号
\'	单引号
\"	双引号
\n	换行
\t	横向制表符(TAB)
\r	回车

【例子】如果字符串中需要出现单引号或双引号，可以使用转义符号 \ 对字符串中的符号进行转义。

```
In [39]: print('let\'s go') # let's go
print("let's go") # let's go
print('C:\\now') # C:\\now
print("C:\\Program Files\\Intel\\Wifi\\Help")
# C:\\Program Files\\Intel\\Wifi\\Help

let's go
let's go
C:\\now
C:\\Program Files\\Intel\\Wifi\\Help
```

【例子】原始字符串只需要在字符串前边加一个英文字母 r 即可。

```
In [40]: print(r'C:\\Program Files\\Intel\\Wifi\\Help')
# C:\\Program Files\\Intel\\Wifi\\Help

C:\\Program Files\\Intel\\Wifi\\Help
```

【例子】三引号允许一个字符串跨多行，字符串中可以包含换行符、制表符以及其他特殊字符。



```
In [41]: para_str = """这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
"""

print(para_str)
# 这是一个多行字符串的实例
# 多行字符串可以使用制表符
# TAB ( \t )。
# 也可以使用换行符 [
# ]。

para_str = '''这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
'''

print(para_str)
# 这是一个多行字符串的实例
# 多行字符串可以使用制表符
# TAB ( \t )。
# 也可以使用换行符 [
# ]。


这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [
]。


这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [
]。
```

## 2. 字符串的切片与拼接

- 类似于元组具有不可修改性
- 从 0 开始 (和 Java 一样)
- 切片通常写成 `start:end` 这种形式，包括「`start` 索引」对应的元素，不包括「`end` 索引」对应的元素。
- 索引值可正可负，正索引从 0 开始，从左往右；负索引从 -1 开始，从右往左。使用负数索引时，会从最后一个元素开始计数。最后一个元素的位置编号是 -1。

【例子】

```
In [42]: str1 = 'I Love LsgoGroup'
print(str1[:6]) # I Love
print(str1[5]) # e
print(str1[:6] + " 插入的字符串 " + str1[6:])
# I Love 插入的字符串 LsgoGroup

s = 'Python'
print(s) # Python
print(s[2:4]) # th
print(s[-5:-2]) # yth
print(s[2]) # t
print(s[-1]) # n

I Love
e
I Love 插入的字符串 LsgoGroup
Python
th
yth
t
n
```

## 3. 字符串的常用内置方法

- `capitalize()` 将字符串的第一个字符转换为大写。

【例子】



```
In [43]: str2 = 'xiaoxie'  
print(str2.capitalize()) # Xiaoxie  
  
Xiaoxie  
  
• lower() 转换字符串中所有大写字符为小写。  
• upper() 转换字符串中的小写字母为大写。  
• swapcase() 将字符串中大写转换为小写，小写转换为大写。
```

【例子】

```
In [44]: str2 = "DAXIExiaoxie"  
print(str2.lower()) # daxiexiaoxie  
print(str2.upper()) # DAXIEXIAOXIE  
print(str2.swapcase()) # daxieXIAOXIE  
  
daxiexiaoxie  
DAXIEXIAOXIE  
daxieXIAOXIE
```

- count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数。

【例子】

```
In [45]: str2 = "DAXIExiaoxie"  
print(str2.count('xi')) # 2  
2
```

- endswith(suffix, beg=0, end=len(string)) 检查字符串是否以指定子字符串 suffix 结束，如果是，返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查。
- startswith(substr, beg=0,end=len(string)) 检查字符串是否以指定子字符串 substr 开头，如果是，返回 True，否则返回 False。如果 beg 和 end 指定值，则在指定范围内检查。

【例子】

```
In [46]: str2 = "DAXIExiaoxie"  
print(str2.endswith('ie')) # True  
print(str2.endswith('xi')) # False  
print(str2.startswith('Da')) # False  
print(str2.startswith('DA')) # True  
  
True  
False  
False  
True
```

- find(str, beg=0, end=len(string)) 检测 str 是否包含在字符串中，如果指定范围 beg 和 end，则检查是否包含在指定范围内，如果包含，返回开始的索引值，否则返回 -1。
- rfind(str, beg=0,end=len(string)) 类似于 find() 函数，不过是从右边开始查找。

【例子】

```
In [47]: str2 = "DAXIExiaoxie"  
print(str2.find('xi')) # 5  
print(str2.find('ix')) # -1  
print(str2.rfind('xi')) # 9  
  
5  
-1  
9
```

- isnumeric() 如果字符串中只包含数字字符，则返回 True，否则返回 False。

【例子】

```
In [48]: str3 = '12345'  
print(str3.isnumeric()) # True  
str3 += 'a'  
print(str3.isnumeric()) # False  
  
True  
False
```

- ljust(width[, fillchar]) 返回一个原字符串左对齐，并使用 fillchar（默认空格）填充至长度 width 的新字符串。
- rjust(width[, fillchar]) 返回一个原字符串右对齐，并使用 fillchar（默认空格）填充至长度 width 的新字符串。

【例子】

```
In [49]: str4 = '1101'  
print(str4.ljust(8, '0')) # 11010000  
print(str4.rjust(8, '0')) # 00001101  
  
11010000  
00001101
```



- `lstrip([chars])` 截掉字符串左边的空格或指定字符。
- `rstrip([chars])` 删除字符串末尾的空格或指定字符。
- `strip([chars])` 在字符串上执行 `lstrip()` 和 `rstrip()`。

【例子】

```
In [50]: str5 = ' I Love LsgoGroup '
print(str5.lstrip()) # 'I Love LsgoGroup '
print(str5.lstrip().strip('I')) # ' Love LsgoGroup '
print(str5.rstrip()) # ' I Love LsgoGroup'
print(str5.strip()) # 'I Love LsgoGroup'
print(str5.strip().strip('p')) # 'I Love LsgoGrou'
```

```
I Love LsgoGroup
Love LsgoGroup
I Love LsgoGroup
I Love LsgoGroup
I Love LsgoGrou
```

- `partition(sub)` 找到子字符串`sub`, 把字符串分为一个三元组 (`pre_sub, sub, fol_sub`) , 如果字符串中不包含`sub`则返回 ('原字符串', '', '')。
- `rpartition(sub)` 类似于 `partition()` 方法, 不过是从右边开始查找。

【例子】

```
In [51]: str5 = ' I Love LsgoGroup '
print(str5.strip().partition('o')) # ('I L', 'o', 've LsgoGroup')
print(str5.strip().partition('m')) # ('I Love LsgoGroup', ' ', '')
print(str5.strip().rpartition('o')) # ('I Love LsgoGr', 'o', 'up')
```

```
('I L', 'o', 've LsgoGroup')
('I Love LsgoGroup', ' ', '')
('I Love LsgoGr', 'o', 'up')
```

- `replace(old, new [, max])` 把将字符串中的 `old` 替换成 `new` , 如果 `max` 指定, 则替换不超过 `max` 次。

【例子】

```
In [52]: str5 = ' I Love LsgoGroup '
print(str5.strip().replace('I', 'We')) # We Love LsgoGroup
```

```
We Love LsgoGroup
```

- `split(str="", num)` 不带参数默认是以空格为分隔符切片字符串, 如果 `num` 参数有设置, 则仅分隔 `num` 个子字符串, 返回切片后的子字符串拼接的列表。

【例子】

```
In [53]: str5 = ' I Love LsgoGroup '
print(str5.strip().split()) # ['I', 'Love', 'LsgoGroup']
print(str5.strip().split('o')) # ['I L', 've Lsg', 'Gr', 'up']

['I', 'Love', 'LsgoGroup']
['I L', 've Lsg', 'Gr', 'up']
```

【例子】

```
In [54]: u = "www.baidu.com.cn"
# 使用默认分隔符
print(u.split()) # ['www.baidu.com.cn']

# 以"."为分隔符
print((u.split('.'))) # ['www', 'baidu', 'com', 'cn']

# 分割0次
print((u.split(".", 0))) # ['www.baidu.com.cn']

# 分割一次
print((u.split(".", 1))) # ['www', 'baidu.com.cn']

# 分割两次
print(u.split(".", 2)) # ['www', 'baidu', 'com.cn']

# 分割两次, 并取序列为1的项
print((u.split(".", 2)[1])) # baidu

# 分割两次, 并把分割后的三个部分保存到三个变量
u1, u2, u3 = u.split(".", 2)
print(u1) # www
print(u2) # baidu
print(u3) # com.cn
```

```
'www baidu com cn'1
```



```
'www.baidu.com.cn'
['www', 'baidu', 'com', 'cn']
['www.baidu.com.cn']
['www', 'baidu.com.cn']
['www', 'baidu', 'com.cn']
baidu
www
baidu
com.cn
```

【例子】去掉换行符

```
In [55]: c = '''say
hello
baby'''

print(c)
# say
# hello
# baby

print(c.split('\n')) # ['say', 'hello', 'baby']

say
hello
baby
['say', 'hello', 'baby']
```

【例子】

```
In [56]: string = "hello boy<[www.baidu.com]>byebye"
print(string.split('[')[1].split(']')[0]) # www.baidu.com
print(string.split('[')[1].split(']')[0].split('.')) # ['www', 'baidu', 'com']

www.baidu.com
['www', 'baidu', 'com']
```

- `splitlines([keepends])` 按照行(\r, \n, \n)分隔, 返回一个包含各行作为元素的列表, 如果参数 `keepends` 为 False, 不包含换行符, 如果为 True, 则保留换行符。

【例子】

```
In [57]: str6 = 'I \n Love \n LsgoGroup'
print(str6.splitlines()) # ['I ', ' Love ', ' LsgoGroup']
print(str6.splitlines(True)) # ['I \n', ' Love \n', ' LsgoGroup']

['I ', ' Love ', ' LsgoGroup']
['I \n', ' Love \n', ' LsgoGroup']
```

- `maketrans(intab, outtab)` 创建字符映射的转换表, 第一个参数是字符串, 表示需要转换的字符, 第二个参数也是字符串表示转换的目标。
- `translate(table, deletechars="")` 根据参数 `table` 给出的表, 转换字符串的字符, 要过滤掉的字符放到 `deletechars` 参数中。

【例子】

```
In [58]: str7 = 'this is string example....wow!!!'
intab = 'aeiou'
outtab = '12345'
trantab = str7.maketrans(intab, outtab)
print(trantab) # {97: 49, 111: 52, 117: 53, 101: 50, 105: 51}
print(str7.translate(trantab)) # th3s 3s str3ng 2x1mpl2....w4w!!!

{97: 49, 101: 50, 105: 51, 111: 52, 117: 53}
th3s 3s str3ng 2x1mpl2....w4w!!!
```

## 4. 字符串格式化

- `format` 格式化函数

```
In [59]: str8 = "{0} Love {1}".format('I', 'Lsgogroup') # 位置参数
print(str8) # I Love Lsgogroup

str8 = "{a} Love {b}".format(a='I', b='Lsgogroup') # 关键字参数
print(str8) # I Love Lsgogroup

str8 = "{0} Love {b}".format('I', b='Lsgogroup') # 位置参数要在关键字参数之前
print(str8) # I Love Lsgogroup

str8 = '{0:.2f}{1}'.format(27.658, 'GB') # 保留小数点后两位
print(str8) # 27.66GB
```

```
I Love Lsgogroup
I Love Lsgogroup
I Love Lsgogroup
27.66GB
```



- Python 字符串格式化符号

符 号	描 述
%c	格式化字符及其ASCII码
%s	格式化字符串, 用str()方法处理对象
%r	格式化字符串, 用repr()方法处理对象
%d	格式化整数
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数 (大写)
%f	格式化浮点数字, 可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e, 用科学计数法格式化浮点数
%g	根据值的大小决定使用%f或%e
%G	作用同%g, 根据值的大小决定使用%f或%E

【例子】

```
In [60]: print('%c' % 97) # a
print('%c %c %c' % (97, 98, 99)) # a b c
print('%d + %d = %d' % (4, 5, 9)) # 4 + 5 = 9
print("我叫 %s 今年 %d 岁!" % ('小明', 10)) # 我叫 小明 今年 10 岁!
print('%o' % 10) # 12
print('%x' % 10) # a
print('%X' % 10) # A
print('%f' % 27.658) # 27.658000
print('%e' % 27.658) # 2.765800e+01
print('%E' % 27.658) # 2.765800E+01
print('%g' % 27.658) # 27.658
text = "I am %d years old." % 22
print("I said: %s." % text) # I said: I am 22 years old..
print("I said: %r." % text) # I said: 'I am 22 years old.'
```

```
a
a b c
4 + 5 = 9
我叫 小明 今年 10 岁!
12
a
A
27.658000
2.765800e+01
2.765800E+01
27.658
I said: I am 22 years old..
I said: 'I am 22 years old.'
```

- 格式化操作符辅助指令

符 号	功 能
m.n	m 是显示的最小总宽度,n 是小数点后的位数 (如果可用的话)
-	用作左对齐
+	在正数前面显示加号(+)
#	在八进制数前面显示零('0'), 在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格

```
In [60]: print('%c' % 97) # a
print('%c %c %c' % (97, 98, 99)) # a b c
print('%d + %d = %d' % (4, 5, 9)) # 4 + 5 = 9
print("我叫 %s 今年 %d 岁!" % ('小明', 10)) # 我叫 小明 今年 10 岁!
print('%o' % 10) # 12
print('%x' % 10) # a
print('%X' % 10) # A
print('%f' % 27.658) # 27.658000
print('%e' % 27.658) # 2.765800e+01
print('%E' % 27.658) # 2.765800E+01
print('%g' % 27.658) # 27.658
text = "I am %d years old." % 22
print("I said: %s." % text) # I said: I am 22 years old..
print("I said: %r." % text) # I said: 'I am 22 years old.'
```

```
a
a b c
4 + 5 = 9
我叫 小明 今年 10 岁!
12
a
^
```



```
A  
27.658000  
2.765800e+01  
2.765800E+01  
27.658  
I said: I am 22 years old..  
I said: 'I am 22 years old.'.
```

- 格式化操作符辅助指令

符号	功能
m.n	m 是显示的最小总宽度,n 是小数点后的位数 (如果可用的话)
-	用作左对齐
+	在正数前面显示加号( + )
#	在八进制数前面显示零('0'), 在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格

#### 【例子】

```
In [61]: print('%5.1f' % 27.658) # '27.7'  
print('%.2e' % 27.658) # 2.77e+01  
print('%10d' % 10) # ' 10'  
print('%-10d' % 10) # '10'  
print('%+d' % 10) # +10  
print('%#0' % 10) # 0o12  
print('%#x' % 108) # 0x6c  
print('%010d' % 5) # 0000000005
```

  

```
27.7  
2.77e+01  
10  
10  
+10  
0o12  
0x6c  
0000000005
```

# 字典

## 1. 可变类型与不可变类型

- 序列是以连续的整数为索引，与此不同的是，字典以"关键字"为索引，关键字可以是任意不可变类型，通常用字符串或数值。
- 字典是 Python 唯一的一个 映射类型，字符串、元组、列表属于序列类型。

那么如何快速判断一个数据类型 X 是不是可变类型的呢？两种方法：

- 麻烦方法：用 `id(X)` 函数，对 X 进行某种操作，比较操作前后的 `id`，如果不一样，则 X 不可变，如果一样，则 X 可变。
- 便捷方法：用 `hash(X)`，只要不报错，证明 X 可被哈希，即不可变，反过来不可被哈希，即可变。

#### 【例子】

```
In [62]: i = 1  
print(id(i)) # 140732167000896  
i = i + 2  
print(id(i)) # 140732167000960  
  
l = [1, 2]  
print(id(l)) # 4300825160  
l.append('Python')
```



```
print(id(l)) # 4300825100
```

```
140731832701760  
140731832701824  
2131670369800  
2131670369800
```

- 整数 `i` 在加 1 之后的 `id` 和之前不一样，因此加完之后的这个 `i` (虽然名字没变)，但不是加之前的那个 `i` 了，因此整数是不可变类型。
- 列表 `l` 在附加 'Python' 之后的 `id` 和之前一样，因此列表是可变类型。

【例子】

```
In [1]: print(hash('Name')) # 7047218704141848153  
print(hash((1, 2, 'Python'))) # 1704535747474881831  
print(hash([1, 2, 'Python']))  
# TypeError: unhashable type: 'list'
```

```
-6668157630988609386  
-1857436431894091236
```

```
TypeError Traceback (most recent call last)  
<ipython-input-1-6416367464f8> in <module>()  
      3 print(hash((1, 2, 'Python'))) # 1704535747474881831  
      4  
----> 5 print(hash([1, 2, 'Python']))  
      6 # TypeError: unhashable type: 'list'  
  
TypeError: unhashable type: 'list'
```

```
In [1]: print(hash({1, 2, 3}))  
# TypeError: unhashable type: 'set'
```

- 数值、字符和元组 都能被哈希，因此它们是不可变类型。

- 数值、字符和元组 都能被哈希，因此它们是不可变类型。
- 列表、集合、字典不能被哈希，因此它是可变类型。

## 2. 字典的定义

字典 是无序的 键:值 (`key:value`) 对集合，键必须是互不相同的 (在同一个字典之内)。

- `dict` 内部存放的顺序和 `key` 放入的顺序是没有关系的。
- `dict` 查找和插入的速度极快，不会随着 `key` 的增加而增加，但是需要占用大量的内存。

字典 定义语法为 {元素1, 元素2, ..., 元素n}

- 其中每一个元素是一个「键值对」-- 键:值 (`key:value`)
- 关键点是「大括号 {}」，「逗号 ,」和「冒号 :」
- 大括号 -- 把所有元素绑在一起
- 逗号 -- 将每个键值对分开
- 冒号 -- 将键和值分开

## 3. 创建和访问字典

【例子】

```
In [88]: brand = ['李宁', '耐克', '阿迪达斯']  
slogan = ['一切皆有可能', 'Just do it', 'Impossible is nothing']  
print('耐克的口号是:', slogan[brand.index('耐克')])  
# 耐克的口号是: Just do it  
  
dic = {'李宁': '一切皆有可能', '耐克': 'Just do it', '阿迪达斯': 'Impossible is nothing'}  
print('耐克的口号是:', dic['耐克'])  
# 耐克的口号是: Just do it
```

```
耐克的口号是: Just do it  
耐克的口号是: Just do it
```

【例子】通过字符串或数值作为 key 来创建字典。



```
In [2]: dic1 = {1: 'one', 2: 'two', 3: 'three'}
print(dic1) # {1: 'one', 2: 'two', 3: 'three'}
print(dic1[1]) # one
print(dic1[4]) # KeyError: 4

{'one': 1, 'two': 2, 'three': 3}
one

KeyError Traceback (most recent call last)
<ipython-input-2-bb8d02bd63a3> in <module>()
      2 print(dic1) # {1: 'one', 2: 'two', 3: 'three'}
      3 print(dic1[1]) # one
----> 4 print(dic1[4]) # KeyError: 4

KeyError: 4
```

```
In [3]: dic2 = {'rice': 35, 'wheat': 101, 'corn': 67}
print(dic2) # {'wheat': 101, 'corn': 67, 'rice': 35}
print(dic2['rice']) # 35

{'rice': 35, 'wheat': 101, 'corn': 67}
35
```

注意：如果我们取的键在字典中不存在，会直接报错 `KeyError`。

【例子】通过元组作为 key 来创建字典，但一般不这样使用。

```
In [91]: dic = {(1, 2, 3): "Tom", "Age": 12, 3: [3, 5, 7]}
print(dic) # {(1, 2, 3): 'Tom', 'Age': 12, 3: [3, 5, 7]}
print(type(dic)) # <class 'dict'>

{(1, 2, 3): 'Tom', 'Age': 12, 3: [3, 5, 7]}
<class 'dict'>
```

通过构造函数 `dict` 来创建字典。

- `dict()` 创建一个空的字典。

【例子】通过 `key` 直接把数据放入字典中，但一个 `key` 只能对应一个 `value`，多次对一个 `key` 放入 `value`，后面的值会把前面的值冲掉。

```
In [92]: dic = dict()
dic['a'] = 1
dic['b'] = 2
dic['c'] = 3

print(dic)
# {'a': 1, 'b': 2, 'c': 3}

dic['a'] = 11
print(dic)
# {'a': 11, 'b': 2, 'c': 3}

dic['d'] = 4
print(dic)
# {'a': 11, 'b': 2, 'c': 3, 'd': 4}

{'a': 1, 'b': 2, 'c': 3}
{'a': 11, 'b': 2, 'c': 3}
{'a': 11, 'b': 2, 'c': 3, 'd': 4}
```

- `dict(mapping)` new dictionary initialized from a mapping object's (key, value) pairs

【例子】

```
In [95]: dic1 = dict([('apple', 4139), ('peach', 4127), ('cherry', 4098)])
print(dic1) # {'cherry': 4098, 'apple': 4139, 'peach': 4127}

dic2 = dict((('apple', 4139), ('peach', 4127), ('cherry', 4098)))
print(dic2) # {'peach': 4127, 'cherry': 4098, 'apple': 4139}

{'apple': 4139, 'peach': 4127, 'cherry': 4098}
{'apple': 4139, 'peach': 4127, 'cherry': 4098}
```

- `dict(**kwargs)` -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: `dict(one=1, two=2)`

【例子】这种情况下，键只能为字符串类型，并且创建的时候字符串不能加引号，加上就会直接报语法错误。

```
In [96]: dic = dict(name='Tom', age=10)
```



```
print(dic) # {'name': 'Tom', 'age': 10}
print(type(dic)) # <class 'dict'>
{'name': 'Tom', 'age': 10}
<class 'dict'>
```

## 4. 字典的内置方法

- `dict.fromkeys(seq[, value])` 用于创建一个新字典，以序列 `seq` 中元素做字典的键，`value` 为字典所有键对应的初始值。

【例子】

```
In [97]: seq = ('name', 'age', 'sex')
dic1 = dict.fromkeys(seq)
print(dic1)
# {'name': None, 'age': None, 'sex': None}

dic2 = dict.fromkeys(seq, 10)
print(dic2)
# {'name': 10, 'age': 10, 'sex': 10}

dic3 = dict.fromkeys(seq, ('小马', '8', '男'))
print(dic3)
# {'name': ('小马', '8', '男'), 'age': ('小马', '8', '男'), 'sex': ('小马', '8', '男')}

{'name': None, 'age': None, 'sex': None}
{'name': 10, 'age': 10, 'sex': 10}
{'name': ('小马', '8', '男'), 'age': ('小马', '8', '男'), 'sex': ('小马', '8', '男')}
```

- `dict.keys()` 返回一个可迭代对象，可以使用 `list()` 来转换为列表，列表为字典中的所有键。

【例子】

```
In [98]: dic = {'Name': 'lsgogroup', 'Age': 7}
print(dic.keys()) # dict_keys(['Name', 'Age'])
lst = list(dic.keys()) # 转换为列表
print(lst) # ['Name', 'Age']

dict_keys(['Name', 'Age'])
['Name', 'Age']
```

- `dict.values()` 返回一个迭代器，可以使用 `list()` 来转换为列表，列表为字典中的所有值。

【例子】

```
In [100]: dic = {'Sex': 'female', 'Age': 7, 'Name': 'Zara'}
print(dic.values())
# dict_values(['female', 7, 'Zara'])

print(list(dic.values()))
# [7, 'female', 'Zara']

dict_values(['female', 7, 'Zara'])
['female', 7, 'Zara']
```

- `dict.items()` 以列表返回可遍历的(键, 值)元组数组。

【例子】

```
In [101]: dic = {'Name': 'Lsgogroup', 'Age': 7}
print(dic.items())
# dict_items([('Name', 'Lsgogroup'), ('Age', 7)])

print(tuple(dic.items()))
# (('Name', 'Lsgogroup'), ('Age', 7))

print(list(dic.items()))
# [('Name', 'Lsgogroup'), ('Age', 7)]

dict_items([('Name', 'Lsgogroup'), ('Age', 7)])
((Name, 'Lsgogroup'), ('Age', 7))
[('Name', 'Lsgogroup'), ('Age', 7)]
```

- `dict.get(key, default=None)` 返回指定键的值，如果值不在字典中返回默认值。

【例子】



```
In [102]: dic = {'Name': 'Lsgogroup', 'Age': 27}
print("Age 值为 : %s" % dic.get('Age')) # Age 值为 : 27
print("Sex 值为 : %s" % dic.get('Sex', "NA")) # Sex 值为 : NA
print(dic) # {'Name': 'Lsgogroup', 'Age': 27}

Age 值为 : 27
Sex 值为 : NA
{'Name': 'Lsgogroup', 'Age': 27}
```

- `dict.setdefault(key, default=None)` 和 `get()` 方法类似, 如果键不存在于字典中, 将会添加键并将值设为默认值。

【例子】

```
In [103]: dic = {'Name': 'Lsgogroup', 'Age': 7}
print("Age 键的值为 : %s" % dic.setdefault('Age', None)) # Age 键的值为 : 7
print("Sex 键的值为 : %s" % dic.setdefault('Sex', None)) # Sex 键的值为 : None
print(dic)
# {'Age': 7, 'Name': 'Lsgogroup', 'Sex': None}

Age 键的值为 : 7
Sex 键的值为 : None
{'Name': 'Lsgogroup', 'Age': 7, 'Sex': None}
```

`key in dict` 操作符用于判断键是否存在于字典中, 如果键在字典 `dict` 里返回 `true`, 否则返回 `false`。而 `not in` 操作符刚好相反, 如果键在字典 `dict` 里返回 `false`, 否则返回 `true`。

```
In [104]: dic = {'Name': 'Lsgogroup', 'Age': 7}

# in 检测键 Age 是否存在
if 'Age' in dic:
    print("键 Age 存在")
else:
    print("键 Age 不存在")

# 检测键 Sex 是否存在
if 'Sex' in dic:
    print("键 Sex 存在")
else:
    print("键 Sex 不存在")

# not in 检测键 Age 是否存在
if 'Age' not in dic:
    print("键 Age 不存在")
else:
    print("键 Age 存在")

# 键 Age 存在
# 键 Sex 不存在
# 键 Age 存在
```

- `dict.pop(key[,default])` 删除字典给定键 `key` 所对应的值, 返回值为被删除的值。`key` 值必须给出。若 `key` 不存在, 则返回 `default` 值。
- `del dict[key]` 删除字典给定键 `key` 所对应的值。

【例子】

```
In [105]: dic1 = {1: "a", 2: [1, 2]}
print(dic1.pop(1), dic1) # a {2: [1, 2]}

# 设置默认值, 必须添加, 否则报错
print(dic1.pop(3, "nokey"), dic1) # nokey {2: [1, 2]}

del dic1[2]
print(dic1) # {}

a {2: [1, 2]}
nokey {2: [1, 2]}
{}
```

- `dict.popitem()` 随机返回并删除字典中的一对键和值, 如果字典已经为空, 却调用了此方法, 就报出 `KeyError` 异常。

【例子】

```
In [107]: dic1 = {1: "a", 2: [1, 2]}
print(dic1.popitem()) # {2: [1, 2]}
print(dic1) # (1, 'a')

(2, [1, 2])
```



```
{1: 'a'}
```

- `dict.clear()` 用于删除字典内所有元素。

【例子】

```
In [108]: dic = {'Name': 'Zara', 'Age': 7}
print("字典长度 : %d" % len(dic))  # 字典长度 : 2
dic.clear()
print("字典删除后长度 : %d" % len(dic))
# 字典删除后长度 : 0

字典长度 : 2
字典删除后长度 : 0
```

- `dict.copy()` 返回一个字典的浅复制。

【例子】

```
In [109]: dic1 = {'Name': 'Lsgogroup', 'Age': 7, 'Class': 'First'}
dic2 = dic1.copy()
print("dic2")
# {'Age': 7, 'Name': 'Lsgogroup', 'Class': 'First'}
```

```
dic2
```

【例子】直接赋值和 copy 的区别

```
In [110]: dic1 = {'user': 'lsgogroup', 'num': [1, 2, 3]}

# 引用对象
dic2 = dic1
# 浅拷贝父对象（一级目录），子对象（二级目录）不拷贝，还是引用
dic3 = dic1.copy()

print(id(dic1))  # 148635574728
print(id(dic2))  # 148635574728
print(id(dic3))  # 148635574344

# 修改 data 数据
dic1['user'] = 'root'
dic1['num'].remove(1)

# 输出结果
print(dic1)  # {'user': 'root', 'num': [2, 3]}
print(dic2)  # {'user': 'root', 'num': [2, 3]}
print(dic3)  # {'user': 'runoob', 'num': [2, 3]}
```

- `dict.update(dict2)` 把字典参数 `dict2` 的 `key:value` 对更新到字典 `dict` 里。

【例子】

```
In [111]: dic = {'Name': 'Lsgogroup', 'Age': 7}
dic2 = {'Sex': 'female', 'Age': 8}
dic.update(dic2)
print(dic)
# {'Sex': 'female', 'Age': 8, 'Name': 'Lsgogroup'}
```

```
{'Name': 'Lsgogroup', 'Age': 8, 'Sex': 'female'}
```

