字典 (dict, dictionary)

- 本质上，序列是整数下标到元素对象的有穷映射
  - 基于位置 (下标) 引用其元素；下标只能是从 0 开始的连续整数区间
- 更广义、灵活的映射 (mapping)：字典，类型名 dict
  - 支持从任意关键字集合到任意对象集合的有穷映射
  - 如果需要保存一批数据，且不适合基于下标作检索，则可采用字典，用其他对象 (如字符串等) 作为数据的索引值/键
    - 字/词典：字/词 → 解释
    - 通讯录：姓名 → 手机号码、邮箱地址等联络信息
    - 银行账户：账户编号 → 账户信息记录
    - 航班实时位置信息：航班号→ 位置 信息记录
    - ……

Python 的字典 (1)

- 字典 (dict) 是 Python 的基本数据类型，其实现被高度优化
  - 字典活跃在所有 Python 程序的背后，即使代码里没有直接用到它 —— ***By Andrew Kuchling, 《Beautiful code》***
    - 程序运行时，各名字空间 (namespace) 用字典实现，记录各变量与其关联对象的约束关系
    - 函数的关键字参数、类实例对象的属性等都基于字典实现
- 基本情况
  - 在字典中，每个元素对象对应与惟一的关键字 (key)
  - 字典是可变组合类型，支持两个基本操作 *(以及更多其它操作)*
    - 把一个任意类型的对象值保存在字典里，将其约束到一个给定的关键字 (基于下标表达式赋值)
    - 根据关键字访问对应的关联值 (基于下标表达式取值)

Python 的字典 (2)

- 字典里的关键字/键 (key) 具有惟一性 (无重复关键字)，必须是不变对象，且能够用 == 运算符比较相等
  - 可以用各种类型的数、字符串、元组 (其所有元素也必须是不变对象) 作为字典的关键字
  - 不能用 [1, 2]，([1], [2]) 等可变对象或含可变元素的对象
- Note：dict 要求关键字必须是可哈希/可散列的 (但元素值可以是任意类型的)

hashable -- 可哈希

一个对象的哈希值如果在其生命周期内绝不改变，就被称为 *可哈希* （它需要具有 __hash__() 方法），并可以同其他对象进行比较（它需要具有 __eq__() 方法）。可哈希对象必须具有相同的哈希值比较结果才会相同。

可哈希性使得对象能够作为字典键或集合成员使用，因为这些数据结构要在内部使用哈希值。

大多数 Python 中的不可变内置对象都是可哈希的；可变容器（例如列表或字典）都不可哈希；不可变容器（例如元组和 frozenset）仅当它们的元素均为可哈希时才是可哈希的。 用户定义类的实例对象默认是可哈希的。 它们在比较时一定不相同（除非是与自己比较），它们的哈希值的生成是基于它们的 id()。

Python 的字典 (3)

- 可用内置函数 **hash** 函数得到一个可哈希对象的哈希值
  - 该函数对不可哈希 **(unhashable)** 对象没有定义

hash(*object*)

返回该对象的哈希值（如果它有的话）。哈希值是整数。它们在字典查找元素时用来快速比较字典的键。相同大小的数字变量有相同的哈希值（即使它们类型不同，如 1 和 1.0）。

```
>>> print(hash(1), hash(1.0), hash(True), hash(1+0j))
1 1 1 1
>>> 1 == 1.0 == True == 1+0j
True
```

```
>>> hash((1, [1, 2]))
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    hash((1, [1, 2]))
TypeError: unhashable type: 'list'
```

字典的构造 (1)

1. 字典的直接描述形式 (display)：用 { } 括起一对对 键：值
2. 用 dict 类型名从二元组的表/元组构造字典 (类型转换)

```
>>> {} # 建立一个空字典
{}
>>> d1 = {'math': 114, 'phys': 247, 'chem': 306}
>>> d2 = dict([('math', 114), ('phys', 247), ('chem', 306)])
>>> # 键/元素对的顺序不影响所创建的字典
>>> d3 = dict([('phys', 247), ('chem', 306), ('math', 114)])
>>> d4 = dict(zip(['math', 'phys', 'chem'], (114, 247, 306)))
>>> # 字符串作为键时，可用关键字实参的方式
>>> d5 = dict(math=114, phys=247, chem=306)
>>>
>>> d1 == d2 == d3 == d4 == d5
True
>>> print(d1)
{'math': 114, 'phys': 247, 'chem': 306}
```

字典的构造 (2)

3. 字典推导式：**{关键字表达式：值表达式 for 变量 in 迭代器}**

   - 同样允许 if 段，允许任意多个 for 段和 if 段

```
>>> {n: f'file_{n:03d}' for n in range(3)}
{0: 'file_000', 1: 'file_001', 2: 'file_002'}
```

```
>>> {x: n for x in 'abc' for n in range(3)}
```

```
>>> {x*n: n for x in 'abc' for n in range(1, 3)}
{'a': 1, 'aa': 2, 'b': 1, 'bb': 2, 'c': 1, 'cc': 2}
```

字典的构造 (2)

3. 字典推导式：**{关键字表达式：值表达式 for 变量 in 迭代器}**

   - 同样允许 if 段，允许任意多个 for 段和 if 段

```
>>> Dial_codes = [(86, 'China'), (1, 'United States'),
                  (81, 'Japan'), (7, 'Ruassia'),
                  (91, 'India'), (55, 'Brazil')]
>>> {country: code for code, country in Dial_codes}
{'China': 86, 'United States': 1, 'Japan': 81, 'Ruassia':
7, 'India': 91, 'Brazil': 55}
>>>
>>> {code: country for code, country in Dial_codes
     if code > 80}
{86: 'China', 81: 'Japan', 91: 'India'}
```

字典的一组操作 (1)

- 设 dic 是一个字典，k 是一个关键字，v 是一个值

dic[k] = v	在 dic 里记录 k 关联的值为 v (无论原来有没有 k)
dic[k]	得到 dic 里与 k 关联的值，如果 k 不存在时则报错
len(dic)	得到 dic 元素 (关键字/值关联对) 的个数
k in dic k not in dic	检查 dic 的关键字，如果有字 k 就返回 True，否则 False 与 k in dic 相反
del dic[k]	删除关键字为 k 的关联，不存在 k 时报错 (可能需要先检索再 del)
iter(dic)	得到在 dic 的关键字上的一个迭代器，迭代顺序由元素插入顺序确定 (需要时可以先用 sorted 排序)
dic.get(k) dic.get(k, default)	得到 dic 中与 k 关联的值； 如果字典里不存在 k，返回 None 或者调用中提供的 default 值 (肯定不会报错)

字典的一组操作 (2)

dic.copy()	得到 dic 的一个拷贝，但是并不新建 dic 的关键字/元素的拷贝 — 浅拷贝 (也就是说，新建的字典与原字典共享关键字和值) 关键字是不变对象，这种共享不会带来问题；但值可以是可变对象，需要注意变动的影响
dic.pop(k)	从 dic 里删除关键字 k 并返回 k 的关联值，没有 k 时报错 (应结合关键字检索使用)
dic.pop(k, v)	带有参数 v 的形式：在 dic 里没有 k 时返回值 v
dic.popitem()	以二元组 (k, v) 的形式返回 dic 里的某个元素，并将该元素从 dic 里删除；键值对会按 LIFO 的顺序被返回

```
>>> dic = {"a": [1, 2, 3]}
>>> dic2 = dic.copy()
>>> dic2["a"][2] = 5
>>> dic
{'a': [1, 2, 5]}
```

字典的一组操作 (3)

list(dic)	返回字典 **dic** 中使用的所有键的列表
reversed(dic)	返回一个逆序获取字典键的迭代器
dic.clear()	删除 **dic** 里的所有元素，将其变回一个空字典
dic.update([*other*])	用另一个字典 **other** 的元素更新 **dic**，也可以用一批关键字参数做这个操作； 这种更新相当于一组基于关键字的关联赋值：如果 **dic** 里原来没有关键字 **k** 则加入 **k** 及其关联；如果已有，就修改 **k** 的关联值 (可实现批量更新)
dic.setdefault (k [, default])	如果 **dic** 里有关键字 **k**，就返回其关联值； 如果没有，则让 **dic[k] = default**，并返回 **default** (如果调用时没提供 **default**，则让 **dic[k] = None**)