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。

```
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))  # 4300825160
```

- 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。

```
print(hash('Name'))  # -9215951442099718823

print(hash((1, 2, 'Python')))  # 823362308207799471

print(hash([1, 2, 'Python']))
# TypeError: unhashable type: 'list'

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。
- 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

```
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
```

【例子】通过字符串或数值作为`key`来创建字典。

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

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

```
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'>
```

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。

```
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}
```

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

【例子】

```
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}
```

- 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。

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

```
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')}
```

- 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。

```
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.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。

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

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

- dict.items()以列表返回可遍历的 (键, 值) 元组数组。

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

```
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}
```

- 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。

```
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}
```

- key in dict in 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。

```
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。

```
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)  # {}
```

- 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。

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

- dict.clear()用于删除字典内所有元素。

```
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
```

- dict.copy()返回一个字典的浅复制。

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

【例子】直接赋值和 copy 的区别

```
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】

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