python元组 (tuple)

在python中，元组是一种与列表非常相似的数据类型，除了append，insert等改变数据内容的方法，列表支持的方法，元组几乎都支持, 它们之间最大的区别在于元组里的元素不能被修改。元组几乎就是列表的翻版，你学会了列表，就可以用操作列表的方法来操作元组。

但显然，事情不会这么简单，从第3小节开始，本篇教程会向你阐明元组存在的意义，这部分内容已经超出了你目前能够理解的知识范围，只作为了解，等到你掌握了足够多的知识以后，可以再回到这里学习。

1. 元组的概念（tuple）

元组和列表非常相似，不同之处在于，元组里的元素不能修改。
元组使用小括号，列表使用中括号

```
tup1 = (1,2,3,4)
tup2 = (1, )
```

如果元组里只有一个元素，也必须使用逗号
对于元组的理解，参照列表即可

元组与列表之间可以使用list()和tuple()函数进行转换

```
lst = [1,2,3]
tup = tuple(lst)
lst2 = list(tup)
```

2. 元组练习题

写出下面代码的执行结果和最终结果的类型

1. (1, 2)*2
2. (1,)*2
3. (1)*2

答案如下

```
1. (1, 2, 1, 2)
2. (1, 1)
3. 2
```

第一题应该没有异议，关键是第2题和第3题，元组里只有一个数据时，必须有逗号，如果没有逗号，就变成了第3题的形式，第3题本质上是1*2,那对小括号就如同我们小学学过的小括号一样，只是为了体现运算优先级而已。

当元组只有一个数据时，如果省略了逗号，那么小括号的作用就不再是表示元组，而是表示运算优先级

3. 为什么python的元组看起来很多余?

乍看起来，元组是一种非常多余的数据类型，因为列表已经足够用了，而元组似乎只是重复了列表的功能，却又要求不能被修改，事情果真是这样么?

当然不是，元组在特定场景下可以实现列表无法实现的效果

3.1 函数返回多个结果时，元组可以作为返回值

```
def func(x, y):
    return x, y, x+y

res = func(2, 3)
print(res)
```

当函数有多个返回值时，最终以元组的形式返回，程序输出结果为

```
(2, 3, 5)
```

当函数返回多个结果时，以列表的形式返回，难道不也是可行的么？从程序设计的角度看，函数返回多个结果时，以元组形式返回好于以列表形式返回，原因在于列表是可变对象，这意味着函数的返回结果是可修改的，那么函数在使用时就容易出现修改函数返回值的情况。

某些情况下，我们不希望函数的返回值被他人修改，元组恰好满足了我们的要求，如果函数本意就是返回一个列表，那么在renturn时，就应该直接返回列表，而不是返回多个结果。

3.2 元组作为函数的可变参数

```
def func(*args):
    print(args, type(args))

func(3, 4, 5)
```

定义一个支持可变参数的函数时，args的类型是元组，在函数内可以从args中依次取出传入的参数，那么同样的问题，为什么不是列表呢？还是从程序设计的角度出发，如果args被设计成列表，由于列表可以被修改，保不齐某个程序员在实现函数的时候修改了args，传入的参数被修改了，或是增加，或是减少，这样就会引发不可预知的错误。

但现在，python将其设计成元组，元组无法修改，你在函数内部无法修改传入的参数，这就保证了函数入参的安全性。

3.3 元组可以作为字典的key，可以存储到集合中

想要成为字典的key，或是存储到集合中，必须满足可hash这个条件，所有的可变对象，诸如列表，集合，字典都不能做key，但元组可以，在一些特定情境下，用元组做key，非常实用，比如下面这个练习题目

题目要求：已知有两个列表

```
lst1 = [3, 5, 6, 1, 2, 4, 7]
lst2 = [6, 5, 4, 7, 3, 8]
```

从两个列表里各取出一个数，他们的和如果为10，则记录下来，请写程序计算，这种组合一共有几种，分别是什么，要求组合不能重复。

从lst1中取3， lst2中取7，这对组合满足要求，从lst1中取7，lst2中取3，也满足要求，但是这个组合已经存在了，因此不算。

使用嵌套循环，就可以轻易的完成这个题目，但是这中间过程要去除掉重复的组合，正好可以利用元组，算法实现如下

```
lst1 = [3, 5, 6, 1, 2, 4, 7]
lst2 = [6, 5, 4, 7, 3, 8]

res_set = set()
for i in lst1:
    for j in lst2:
        if i + j == 10:
            if i > j:
                res_set.add((j, i))
            else:
                res_set.add((i, j))

print(res_set)
```

程序输出结果

```
{(3, 7), (4, 6), (5, 5), (2, 8)}
```

去重的过程恰好利用了元组，将组合以元组的形式存储到集合中，利用集合的不重复性来达到去重的目的，元组里第一个元素是组合中较小的那个数

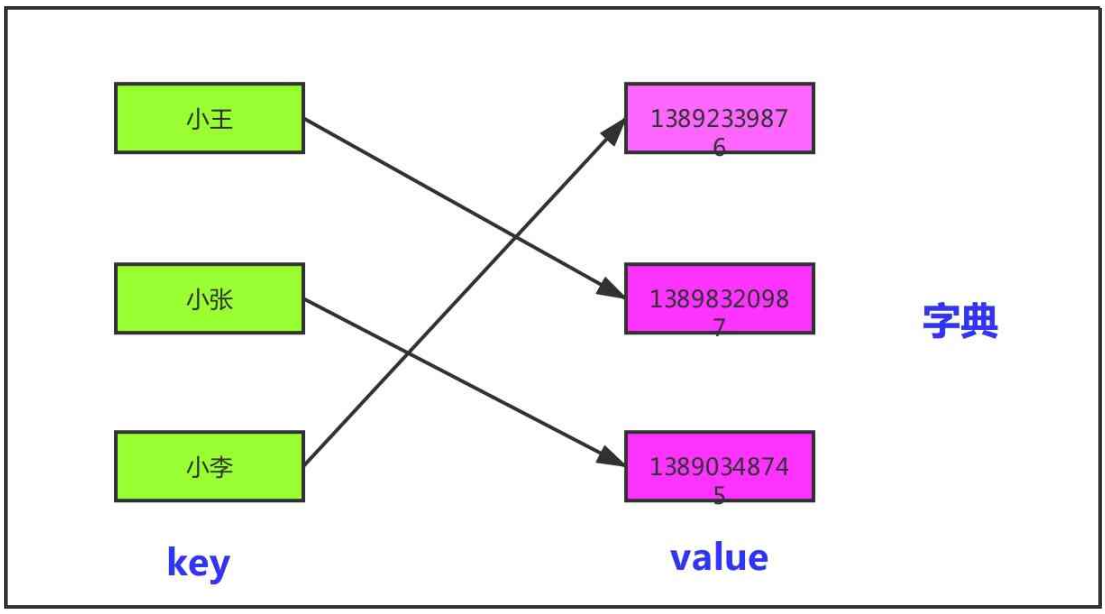
字典

1. 什么是字典

字典并不是什么全新的概念，早在上小学时，你就已经接触过字典，没错，就是《新华字典》，新华字典的结构和python语言中的字典，在结构上是一样的。

咱们以读音来查找一个汉字，比如"张"这个字，读音是zhang,一声，你一定可以在字典里找到与它对应的页数，假设是第100页，那么zhang 和 100之间就有一个映射关系，你知道了zhang，也就知道了100。

另一个较为常见的例子是手机通讯录，你想找一个人的电话时，你应该在通讯录里找到这个人的名字，然后点进去查看它的电话号，姓名和电话号之间存在着映射关系，你知道姓名，就知道电话号，下面这张图展示了字典的数据结构



左侧是key,右侧是value。

字典 (dict)

python中的字典(dict)是数据的无序集合， 与列表，元组，集合将单个值作为存储数据不同， 字典存储的是key:value对， 字典中存储的数据总是成对的， key被称之为键， value称之为值， 值可以是任意数据且可重复， 而键不可重复且必须是可hash的。

1. 创建一个字典

1.1 创建字典

字典是一个容器类型数据，字典里存储的是键值对,键值对用冒号分割key与value，每个键值(key-value)对用逗号分割，整个字典包裹在花括号{}中。

下面是一个字典的示例

```
contacts_dict = {
    "小王": '13892339876',
    "小张": '13898320987',
    "小李": '13890348745'
}
```

如果字典里没有任何键值对，那么它就是一个空字典

```
empty_dict = {}
```

如果想知道字典中有多少数据，可以使用len函数来获取

```
print(len(contacts_dict))
```

想要知道一个key是否在字典中存在，使用in 或者not in 成员操作符

```
print('小王' in contacts_dict)
```

1.2 什么数据可以做key,什么数据可以做value

什么数据可以做key

并不是所有的数据都可以做key，想成为key，是有要求的，数据必须是可hash的，下面罗列的5种数据类型是可以做字典key的数据类型

1. bool
2. int
3. float
4. 字符串
5. 元组

下面3中数据类型不可以做字典的key

1. 列表
2. 集合
3. 字典

凡是可变对象都不可以做字典的key，凡是不可变对象，都可以做字典的key。

bool类型的数据只有True和False两个值，虽然他们可以做字典的key，但实践时，你最好不要这样做，会导致古怪的问题，比如下面的代码

```
int_dict = {
    1: '1做key',
    True: 'True做key'
}

print(int_dict)
```

执行代码，输出结果为

```
{1: 'True做key'}
```

是不是很奇怪，明明两个键值对,实际输出却只有一个，而且key从True变成了1，这是怎么一回事?

```
print(1==True)      # 判断1是否与True相等
print(issubclass(bool, int))  # 判断bool类型是否为int类型的子类
```

程序输出结果为

```
True
True
```

1 和 True是相等的，0和False是相等的，bool类型是int类型的子类，这就是原因，在定义字典时，第二个键值对覆盖了前面的键值对，在字典中，key是不会出现重复的，后加入的总是会覆盖前面的。

什么数据可以做value

对于value，字典没有任何要求，任何数据都可以做value，包括列表，集合，甚至是字典。

```
dic = {
    'list': [2, 3, 4],
    'set': set([2, 4, 6]),
    'dict': {
        '小明': 99
    }
}
```

2. 新增键值对和修改value

下面的字典里存储了学生的语文考试分数

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}
```

现在你发现，小丽的分数还没有写入到字典中，她考了100分，你需要新增一个键值对，记录小丽的分数

```
score_dict['小丽'] = 100
```

这就是字典里新增键值对的方式，一对中括号紧跟在字典后面，中括号里写key，等号右侧写value，是不是很简单。

如果一个key原本已经存在了，那么上面这种操作方式会产生什么样的结果呢?

```
score_dict['小明'] = 99
```

'小明'这个key已经存在于字典中了，上面这种写法尝试新增，但不会发生新增这种事情，因为字典里的key不允许重复，因此，最终的结果是value被修改，字典被修改成

```
score_dict = {
    '小明': 99,
    '小刚': 98,
    '小红': 94
}
```

简单总结一下，对于score_dict['小明'] = 99 这种代码，如果key不存在，那么就会新增键值对，如果key已经存在，就会修改value

3. 访问字典里的值

想要访问字典里的值，必须提供key，对字典的任何操作，都必须通过key才能进行，就如同对列表的操作都必须通过索引才能进行一样。

比如你想知道小红的语文分数，那你应该这样写代码

```python
score_dict = {
    '小明': 99,
    '小刚': 98,
    '小红': 94
}

print(score_dict['小红'])
```

score_dict['小红'] 这种写法的意思就是获取'小红'这个key所对应的value，如果你提供了一个并不存在的key，则会引发一个KeyError错误

```python
print(score_dict['小丽'])
```

错误为

```
Traceback (most recent call last):
  File "/Users/kwsy/PycharmProjects/pythonclass/mytest/demo.py", line 7, in <module>
    print(score_dict['小丽'])
KeyError: '小丽'
```

4. 删除字典里的键值对

现在想要从字典里删除小红的成绩，和访问字典里的key一样，你必须指定要删除的key

```python
del score_dict['小红']
```

除了这种方法外，你还可以使用字典的pop方法

```python
score_dict.pop('小红')
```

嵌套字典

同嵌套列表一样，python的字典作为一个容器，可以在字典里存储字典，类似下面的形式

```python
stu_dict = {
    'name': '小明',
    'age': 12,
    'score': {
        '语文': 90,
        '数学': 98
    }
}
```

在嵌套列表，嵌套字典中，对于嵌套的层次，没有任何要求，只要你自己能理清嵌套逻辑，你想嵌套多少层都可以。

如果你想获取小明的语文分数，那么就需要逐层的来获取value

```python
print(stu_dict['score']['语文'])
```

stu_dict['score']的值是

```
{
    '语文': 90,
    '数学': 98
}
```

仍然是一个字典，想要获取语文分数，就继续使用[]操作符来获取key所对应的value

字典方法介绍

方法	功能作用
clear()	删除字典内所有的元素
copy()	返回字典的浅复制
fromkeys()	以指定key创建一个新的字典
get()	返回指定key的值
items()	成对返回所有key和value
keys()	返回字典所有的key
values	返回字典所有value
setdefault()	为key设置对应的默认值
update()	更新字典
pop()	删除键值对

字典内置的方法可以帮我们实现很多功能，下面逐个介绍他们，讲解他们的用途

1. clear, 删除字典内所有的元素

```
dic = {
    '小明': 98
}
dic.clear()
print(dic)
```

使用clear方法后，字典dic变成了空字典，有人可能会问，这种清空字典的方法和直接将空字典赋值给变量dic有什么区别

```
dic = {
    '小明': 98
}
dic = {}
print(dic)
```

程序最终输出的结果同样是{},dic 变成了空字典。两种方式，变量dic都变成了空字典，但意义不同，使用clear方法，字典在内存中的地址没有发生变化，但是第二种方法，变量dic指向了一个新的空字典，原来的字典被垃圾回收机制回收了，我们可以通过输出变量的内存地址来验证

```
dic1 = {
```

```
    '小明': 98
}
print("使用clear方法前，dic1内存地址为", id(dic1))
dic1.clear()
print(dic1)
print("使用clear方法后，dic1内存地址为", id(dic1))

print("\n\n分割线"+"*"*30 + "\n"*2)

dic2 = {
    '小明': 98
}
print("赋值空字典之前，dic1内存地址为", id(dic2))
dic2 = {}
print(dic2)
print("赋值空字典之后，dic1内存地址为", id(dic2))
```

程序输出结果为

```
使用clear方法前，dic1内存地址为 4352796640
{}
使用clear方法后，dic1内存地址为 4352796640


分割线******************************


赋值空字典之前，dic1内存地址为 4729716312
{}
赋值空字典之后，dic1内存地址为 4729716168
```

clear是清空字典，而将一个空字典赋值给变量，并不是清空，只是修改了变量的引用而已。

2. copy， 返回字典的浅复制

```
dic1 = {
    '小明': 98
}

dic2 = dic1.copy()

print(dic1)
print(dic2)
```

程序输出结果为

```
{'小明': 98}
{'小明': 98}
```

dic2是dic1的复制品，他们的内容一模一样，在python中，还有一个模块，可是实现数据的复制功能，它就是copy模块

```
import copy


dic1 = {
    '小明': 98
}

dic2 = copy.copy(dic1)

print(dic1)
print(dic2)
```

这两段代码都实现了浅复制，浅复制是一种危险的复制，建议你不要使用，因为这种复制并没有创建新的对象，因此，你对dic2的修改会影响到dic1

```
dic1 = {
    'stu': ['小明', '小红']
}

dic2 = dic1.copy()
dic2['stu'].append('小刚')

print(dic1)
```

程序输出结果为

```
{'stu': ['小明', '小红', '小刚']}
```

关于对象的深拷贝和浅拷贝，会有专门的章节进行讲解

3. fromkeys,以指定key创建一个新的字典

```
stu_dict = dict.fromkeys(['小明', '小刚'], 90)
print(stu_dict)
```

程序输出结果为

```
{'小明': 90, '小刚': 90}
```

fromkeys方法接受两个参数，第一个参数是序列，可以是列表，也可以是元组，方法将以这个序列里的元素做key，生成新的字典。value由第二个参数来决定，我在代码里传入参数90，所有key所对应的value就都是90，如果不传这个参数，默认value为None

4. get，返回指定key的值

get方法，是一种安全的获取value的方法，如果key不存在，则返回default，default可以由你来指定，如果你不指定，则默认为None

```
empty_dict = {}

print(empty_dict.get('python'))
print(empty_dict.get('python', 100))
```

程序输出结果

```
None
100
```

5. items(),成对返回所有key和value

items()方法通常被用在字典遍历的场景下

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

for key, value in score_dict.items():
    print(key, value)
```

items()方法返回一个可迭代对象，使用for循环遍历这个可迭代对象时，得到的是一个元组，元组内包含key和 value

下面的代码向你揭示items()方法返回的对象的本质面目

```
from collections import Iterable
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

iter_obj = score_dict.items()
print(isinstance(iter_obj, Iterable))

for item in iter_obj:
    print(item)
```

程序输出结果为

```
True
('小明', 96)
('小刚', 98)
('小红', 94)
```

6. keys,返回字典所有的key

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

keys = score_dict.keys()

print(keys, type(keys))

for key in keys:
    print(key)
```

程序输出结果

```
dict_keys(['小明', '小刚', '小红']) <class 'dict_keys'>
小明
小刚
小红
```

keys()方法在py2.7里，返回的是包含了所有key的列表，但在py3.6中，返回的是可迭代对象，遍历这个对象，就可以得到字典所有的key

7. values,返回字典所有value

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

values = score_dict.values()

print(values, type(values))

for value in values:
    print(value)
```

values()方法返回的是一个可迭代对象，遍历这个可迭代对象，可以获得字典所有的value

8. setdefault，为key设置对应的默认值

这个方法和get有些类似，如果key不存在，则增加新的键值对，如果key已经存在，则不做任何操作

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

score_dict.setdefault('小明', 100)     # 小明这个key已经存在，因此这行语句不产生任何影响
score_dict.setdefault('小丽', 97)      # 小丽这个key不存在，增加新的键值对，key为小丽，value为97

print(score_dict)
```

程序输出结果

```
{'小明': 96, '小刚': 98, '小红': 94, '小丽': 97}
```

9. update 更新字典

一般的使用模式是dic1.update(dic2)，将dic2的内容更新到dic1中

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

score_dict2 = {
    '小明': 100,
    '小丽': 98,
}

score_dict.update(score_dict2)
print(score_dict)
```

如果一个key，在两个字典中都存在，则value的最终结果取决于dic2

10. pop 删除键值对

不论是使用del 还是使用pop方法删除字典里的键值对，如果key不存在都会引发KeyError异常，pop与del的不同之处在于，pop会返回所删除键值对的value

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

del score_dict['小红']
print(score_dict.pop('小明'))

print(score_dict)
```

程序输出结果

```
96
{'小刚': 98}
```

字典最佳实践

本篇教程部分内容超出现在所学知识范围，但并没有超出你的能力范围，对于字典的遍历，我将放到for循环章节中进行讲解，对于新知识，你应当抱有学习的热情，主动搜索资料学习他们，积极探索，是掌握一门编程语言的最佳路径。

1. 直接取值有风险，建议使用get方法

直接从字典里取值，面临一定的风险，比如下面的代码

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

print(score_dict['小丽'])
```

程序会报错

```
Traceback (most recent call last):
  File "/Users/zhangdongsheng/experiment/test/test.py", line 7, in <module>
    print(score_dict['小丽'])
KeyError: '小丽'
```

使用一个在字典中不存在的key，会引发异常，为了避免这种异常，通常有两种方法，先来说第一种方法

判断key是否存在

在使用key之前，先判断这个key是否存在，存在了取值，不存在则根据业务要求采取对应的措施

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

if '小丽' in score_dict:
    print(score_dict['小丽'])
else:
    print("字典里没有小丽的分数")
```

使用get方法

字典的get方法是非常实用的方法,该方法可以指定两个参数,一个是key，另一个是key不存在时返回的数据，默认范围None

```
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

print(score_dict.get('小丽'))
```

上面的代码中，要获取'小丽'这个key所对应的value，但是'小丽'不存在于字典中，因此get方法默认返回None,如果你希望get方法在key不存在时返回0，你可以这样写

```
print(score_dict.get('小丽', 0))
```

2. 不要用bool类型数据做key

bool类型是int类型的子类，1和 Ture是相等，0和False是相等的，如果使用字典时，1已经做了key，那么你再用True去做key，就会修改1所对应的value，下面是一个简单示例

```
int_dict = {
    1: '1做key',
    True: 'True做key'
}

print(int_dict)
```

执行代码，输出结果为

```
{1: 'True做key'}
```

3. 使用dict.setdefault方法

一个列表里存放了若干个单词

```
lst = ['']
```

现在要求你统计每个单词出现的次数，并将单词与单词出现的次数存储到字典中

你可以这样实现代码

```
lst = ['green', 'white', 'black', 'white', 'green', 'green']

info = {}

for word in lst:
    if word not in info:
        info[word] = 0

    info[word] += 1

print(info)
```

上面的代码虽然完成了要求，但是不够优雅，不符合python一贯的简洁作风，你应该使用setdefault方法让代码看起来更加简洁

```python
lst = ['green', 'white', 'black', 'white', 'green', 'green']

info = {}

for word in lst:
    info.setdefault(word, 0)
    info[word] += 1

print(info)
```

setdefault方法尝试为key设置默认值，如果这个key已经存在，那么setdefault什么都不做，如果key不存在，则向字典里新增一个key-value键值对，value就是方法里的第二个参数。

上面的代码里，如果单词不存在于字典中，则设置这个单词所对应的value为0，info[word] += 1 等价于 info[word] = info[word] + 1 ，实现了单词出现次数加1的目的

4. 使用items()方法遍历字典

```python
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}
```

现在要求你输出字典里每一个学生的姓名和分数，你可以使用for循环来完成这个题目

```python
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

for key in score_dict:
    print(key, score_dict[key])
```

上面的代码，不够优雅，建议你使用items()方法来实现遍历

```python
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

for key, value in score_dict.items():
    print(key, value)
```

代码看起来是不是简明一些呢，省去了使用key获取value的过程，items()方法返回一个可迭代对象，使用for循环遍历这个可迭代对象时，得到的是一个元组，元组内包含key和 value

下面的代码向你揭示items()方法返回的对象的本质面目

```python
from collections import Iterable
score_dict = {
    '小明': 96,
    '小刚': 98,
    '小红': 94
}

iter_obj = score_dict.items()
print(isinstance(iter_obj, Iterable))

for item in iter_obj:
    print(item)
```

程序输出结果为

```
True
('小明', 96)
('小刚', 98)
('小红', 94)
```

字典练习题

通过几道python字典(dict)练习题来巩固对字典的掌握, 考察你对python字典常用方法的理解和使用,比如keys(), values(),如何判断一个key是否在字典中, 如何用字典来存储并表示数据

1. 字典基本操作

字典内容如下

```python
dic = {
    'python': 95,
    'java': 99,
    'c': 100
}
```

用程序解答下面的题目

1. 字典的长度是多少
2. 请修改'java' 这个key对应的value值为98
3. 删除 c 这个key
4. 增加一个key-value对，key值为 php, value是90
5. 获取所有的key值，存储在列表里
6. 获取所有的value值，存储在列表里
7. 判断 javascript 是否在字典中
8. 获得字典里所有value 的和
9. 获取字典里最大的value
10. 获取字典里最小的value
11. 字典 dic1 = {'php': 97}， 将dic1的数据更新到dic中

2. 字典应用（买水果）

小明去超市购买水果，账单如下

```
苹果  32.8
香蕉  22
葡萄  15.5
```

请将上面的数据存储到字典里，可以根据水果名称查询购买这个水果的费用

很简单哦，用水果名称做key，金额做value，创建一个字典

```
info = {
    。。。。
}
```

3. 字典应用（买水果2）

小明，小刚去超市里购买水果

小明购买了苹果，草莓，香蕉，一共花了89块钱，，小刚购买了葡萄，橘子，樱桃，一共花了87块钱

请从上面的描述中提取数据，存储到字典中，可以根据姓名获取这个人购买的水果种类和总费用。

以姓名做key，value仍然是字典

```
info = {
    '小明': {
        。。。。。。。
    },
    '小刚': {
        。。。。。。
    }
}
```

集合

编程语言中的集合概念，与初中时数学课上所学的集合概念相同。集合里不会存在重复数据，因此可以用来对数据进行去重处理，两个集合之间可以进行交集运算，并集运算，差集运算。

集合中的元素有3个特性：

1. 确定性：给定一个集合，任何对象是不是这个集合的元素是确定的了。
2. 互异性：集合中的元素一定是不同的。
3. 无序性：集合中的元素没有固定的顺序。

集合 (set)

在python中，集合是一个无序的不重复元素序列，因此集合通常用来去除重复的元素，创建集合可以使用set函数或者大括号{}，两个集合之间可以做并集，交集，差集操作.

1. 创建集合(set)

```python
set_1 = set()   # 空集合
set_2 = {'python', 32, 983.9, 'python', 'php'}
```

集合里数据不会重复，即便set_2在创建时，内部有两个'python' 字符串，在赋值语句执行结束后，set_2里将只有1个'python'

创建一个空集合，一定要用set()，而不是{}, {}用来表示空字典。

和前面学习过的列表，元组，字典一样，集合里每个数据之间用逗号分隔，集合里的数据必须是可hash的，因此，集合里不能有字典，集合，列表这三种数据。

2. 添加删除元素

```python
set_1 = set()
set_1.add('python')
set_1.add('python')
set_1.add('php')
set_1.add('java')
print(len(set_1))

set_1.remove('php')
```

添加元素使用add方法，上面的代码虽然添加了两次'python'，但最终集合中只有一个'python'。

从集合中移除数据，使用remove方法，如果集合中没有这项数据，那么remove将会引发异常，一个更安全的方法是discard，如果数据不存在，不会引发异常。

3. 集合常用操作

3.1 clear，清空集合

列表，字典，集合的清空操作，都可以使用clear来完成。

```python
my_set = {'python', 32, 983.9, 'python', 'php'}
my_set.clear()
print(my_set)
```

3.2 in 和 not in

in 和 not in 是成员运算符，对于列表，元组，字典，集合，判断一个数据是否在这些容器类型数据中，都可以使用成员运算符

```python
my_set = {'python', 32, 983.9, 'python', 'php'}
print('php' in my_set)
```

3.3 交集

```python
set1 = set(lst1)
set2 = set(lst2)

inter_set = set1.intersection(set2)
print(inter_set)
```

intersection计算两个集合的交集,程序运行结果是

```
{2, 5}
```

3.4 差集

```
diff_set = set1.difference(set2)
print(diff_set)
```

difference计算两个集合的差集，程序运行结果是

```
{1, 3, 6}
```

如果想计算哪些整数在lst2中而不在lst1中，则表达式为 set2.difference(set1)

3.5 并集

```
union = set1.union(set2)
print(union)
```

union计算两个集合的并集
程序运行结果

```
{1, 2, 3, 5, 6, 7, 9}
```

集合方法讲解

方法	描述
add()	为集合添加元素
clear()	移除集合中的所有元素
copy()	拷贝一个集合
difference()	返回多个集合的差集
difference_update()	在原集合上移除两个集合都存在的元素
discard()	删除集合中指定的元素
intersection()	返回集合的交集
intersection_update()	在原集合上移除与其他集合不重复的元素
isdisjoint()	判断交集是否为空
issubset()	判断指定集合是否为该方法参数集合的子集。
issuperset()	判断该方法的参数集合是否为指定集合的子集
pop()	随机移除元素
remove()	移除指定元素
symmetric_difference()	返回两个集合中不重复的元素集合。
symmetric_difference_update()	移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
union()	返回两个集合的并集
update()	给集合添加元素

1. add()

add方法向集合中添加元素，即便该元素已经存在于集合中也能向里添加

```
my_set = set()
my_set.add(1)
my_set.add(1)
my_set.add(2)
print(my_set)  # {1, 2}
```

2. clear()

clear方法清空集合

```
my_set = set()
my_set.add(1)
my_set.add(2)
print(my_set)   # {1, 2}
my_set.clear()
print(my_set)   # set() 空集合
```

3. copy()

拷贝一个集合

```
my_set = set()
my_set.add(1)
my_set.add(2)
print(my_set)       # {1, 2}

my_set_2 = my_set.copy()
print(my_set_2)     # {1, 2}
```

4. difference()

返回集合的差集

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}

print(s1.difference(s2))    # {1}
print(s2.difference(s1))    # {4}
```

5. difference_update()

在原集合上移除两个集合都存在的元素，方法没有返回值，下面示例中，s1里的元素被改变了

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}

s1.difference_update(s2)
print(s1)   # {1}
```

6. discard()

删除集合中的指定元素

```
s1 = {1, 2, 3}
s1.discard(1)
print(s1)   # {2, 3}
```

7. intersection()

返回集合的交集,集合可以是多个

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = {3, 4, 5}

print(s1.intersection(s2))      # {2, 3}
print(s1.intersection(s2, s3))      # {3}
```

8. intersection_update()

在原集合上移除与其他集合不重复的元素，其实就是在求交集，intersection也是计算交集，但不改变参与计算的集合内容，而intersection_update会改变原集合的内容。

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = {3, 4, 5}

s1.intersection_update(s2)
print(s1)   # {2, 3}
```

9. isdisjoint()

判断交集是否为空

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}

print(s1.isdisjoint(s2))
```

10. issubset()

判断指定集合是否为该方法参数集合的子集。

```
s1 = {1, 2, 3}
s2 = {1, 2, 3, 4}

print(s1.issubset(s2))  # 判断s1是s2的子集
```

11. issuperset()

也是判断是否为自己，只是与issubset判断方向相反

```
s1 = {1, 2, 3}
s2 = {1, 2, 3, 4}

print(s2.issuperset(s1))  # 判断s1是s2的子集
```

12. pop()

随机的移除一个元素,但在实践中，又与该说法相矛盾

```
s1 = {8, 1, 2, 3, 4, 5, 6, 7}
s1.pop()
print(s1)
```

上述代码，每次执行都是删除1

13. remove()

移除指定元素

```python
s1 = {8, 1, 2, 3, 4, 5, 6, 7}
s1.remove(4)
print(s1)
```

14. symmetric_difference()

返回两个集合中不重复的元素集合

```python
s1 = {1, 2, 3}
s2 = {2, 3, 4}
print(s1.symmetric_difference(s2))  # {1, 4}
```

15. symmetric_difference_update()

移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中

```python
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1.symmetric_difference_update(s2)
print(s1)
```

s1最终的结果等于s1.symmetric_difference(s2)的返回值

16. union()

返回两个集合的并集

```python
s1 = {1, 2, 3}
s2 = {2, 3, 4}

print(s1.union(s2))  # {1, 2, 3, 4}
```

17. update()

将另一个集合更新到原集合中，给集合添加元素

```python
s1 = {1, 2, 3}
s2 = {2, 3, 4}

s1.update(s2)
print(s1)       # {1, 2, 3, 4}
```

集合 (set) 练习题

集合 (set) 是python 的基础数据类型，本练习题主要考察你对集合的交集，并集，差集的理解和运用，这三个操作，是集合最常见也是最为重要的操作

集合基本操作

集合间的运算

```
lst1 = [1, 2, 3, 5, 6, 3, 2]
lst2 = [2, 5, 7, 9]
```

- 哪些整数既在lst1中，也在lst2中
- 哪些整数在lst1中，不在lst2中
- 两个列表一共有哪些整数

虽然题目一直在问两个列表，但用列表解答这3个题目效率很低，你应该用集合

```
lst1 = [1, 2, 3, 5, 6, 3, 2]
lst2 = [2, 5, 7, 9]

set1 = set(lst1)
set2 = set(lst2)

。。。。
```

列表，元组，集合，字典之间互相转换

本文着重介绍python 列表(list)，元组(tuple)，集合(set)，字典(dict)四种类型之间的相互转换，转换成python列表需要使用list函数，转成元组需要使用tuple函数，转成集合需要使用set函数，转成字典需要使用dict函数

1. 内置函数list

内置函数list可以将字符串，集合，元组转换为列表

```
a = 'python'    # 字符串
b = {1, 2, 3}   # 集合
c = (1, 2, 3)   # 元组

print(list(a))
print(list(b))
print(list(c))
```

输出结果是

```
['p', 'y', 't', 'h', 'o', 'n']
[1, 2, 3]
[1, 2, 3]
```

list函数能否将字典转换成列表呢，这个需要探索一下

```
print(list({'a': 3, 'b': 5}))
```

输出结果为

```
['a', 'b']
```

使用list函数转换字典，得到的是字典里所有的key，并没有获得value

2. 内置函数set

set函数可以将字符串，列表转换成集合

```
a = 'python'    # 字符串
b = [1, 2, 1, 2]  # 列表
c = {'a': 1, 'b': 2}  # 字典
d = (1, 2, 3, 4)  # 元组

print(set(a))
print(set(b))
print(set(c))
print(set(d))
```

程序输出结果

```
{'t', 'o', 'n', 'p', 'h', 'y'}
{1, 2}
{'b', 'a'}
{1, 2, 3, 4}
```

尝试转换字典时，只能获得由字典的key组成的集合

3. 内置函数tuple

```
a = 'python'    # 字符串
b = [1, 2, 1, 2]  # 列表
c = {'a': 1, 'b': 2}  # 字典
d = {1, 2, 3, 4}    # 集合

print(tuple(a))
print(tuple(b))
print(tuple(c))
print(tuple(d))
```

程序输出结果

```
('p', 'y', 't', 'h', 'o', 'n')
(1, 2, 1, 2)
('a', 'b')
(1, 2, 3, 4)
```

尝试将字典转换为元组时，只能得到由字典的key组成的元组，看来字典有些特殊

4. 内置函数dict

无法直接使用dict函数将列表，元组，集合转换成字典，这是由字典的结构决定的，字典里的元素永远以key-value对的形式存在，key-value是一个映射关系，知道key，便可以获得value，而其他3种容器类型数据不存在映射关系，因此无法转换。

dict函数有自己特殊的使用方法

```python
my_dict = dict(a=1, b=2)
print(my_dict)
```

程序输出结果

```
{'a': 1, 'b': 2}
```

容器类型数据对比学习

python提供了4种基础容器类型数据，他们是列表，元组，字典，集合。

这四种数据的灵活应用，可以为我们解决大部分数据处理过程中遇到的问题，通过一系列对比学习，希望你可以将这部分知识掌握的更加扎实牢固

1.4个容器类型数据的共同点

4个容器类型数据存在的目的都是为了存储数据，这是他们最大的共同点，也是我们对容器类型数据的基本认识。

这个共同点也引出了他们的不同点，既然都能存储数据，为什么要弄出4个容器类型数据，而不是用一种就可以了呢?

2. 存储数据的目的不同

容器	存储目的	示例
列表	按顺序存储，按顺序使用，单纯的堆积数据	lst = [1, 2, 3]
元组	按顺序存储，按顺序使用，单纯的堆积数据	tup = (1, 2, 3)
字典	按关键字存储，提供key到value的映射，映射关系才是我们关心的	dic = {'python': 100, 'php': '90'}
集合	按关键字存储，主要目的是去重	set = {1, 4, 5}

3. 创建，新增，删除，修改，查询的相同与不同

容器	创建	新增	删除	修改	查询
列表	用[] 创建	append方法	del 方法	通过索引进行修改	通过索引进行查询
元组	用()创建	无新增方法	无删除方法	无修改方法	通过索引查询
字典	用{}创建	通过关键字赋值新增	del 方法	通过关键字赋值进行修改	通过关键字查询
集合	使用{}创建	add 方法	del 方法	无修改方法	无查询方法

解释一些疑问

疑问1 同样是新增，为什么列表的方法名是append，而集合的新增方法名是add?

列表新增，除了append方法，还有insert方法，append默认在列表尾部追加，insert要指定插入的索引位置，这两个方法名都比较含蓄的体现出了列表的有序特点

集合里没有索引的概念，也就没有尾部的概念，新增方法名用add比较合适，和顺序无关，就是增加一个数据

疑问2 为什么集合没有修改和查询的方法

从使用场景来分析，集合的主要作用是为了去重，不存在修改的操作，至于查询操作，没有索引，也就不能通过索引来查询数据，也不像字典那样一个key对应一个value，因此，也无法像字典那样通过key去查询value，唯一的近似查询的操作是in 这个成员操作符判断某个数据是否在集合中

4. 列表与元组，字典与集合对比

通过前面的学习，很容易发现，列表与元组有点像，字典与集合有点像，下面，对他们进行仔细的比较分析

4.1 列表与元组

列表与元组几乎一样，唯一的不同的是，元组不可以修改。元组没有新增，修改，删除这3个方法，那么元组存在的价值和意义是什么呢?

- 可以做字典和集合的key
- 元组可以做函数的返回值

由于元组是不可变对象，因此，元组可以用来做字典和集合的key, 使用元组做函数的返回值，则可以防止函数的使用者修改函数的返回结果，下面的这段代码像你演示这两种功能

```python
lst1 = [10, 3, 2, 6, 7, 5]
lst2 = [3, 2, 5, 6, 10, 7]

combine_set = set()

for item1 in lst1:
    for item2 in lst2:
        if item1 + item2 == 12:
            combine_set.add((min(item1, item2), max(item1, item2)))


print(combine_set)
```

从两个列表里各取出一个数，令他们的和为12，请问这种组合一共有多少?

5+7= 12，从lst1中取5，lst2中取7符合要求，从lst1中取7，lst2中取5也符合要求，但这两个组合是相同的，这就存在了一个去重的操作要求，为了去重，用两个数组成元组，值小的数放前面，值大的数放后面，这样就达到了去重的目的。

4.2 字典与集合

字典和集合的创建都用{}， 我们可以将集合看成是一个特殊的字典，集合里key和value是相同的，只是集合里隐去了value部分，只保留了key的部分。

字典和集合的key都是不能重复的，因为key存在的意义就是唯一的与一个value建立起映射关系，两个相同的key，不论是映射到相同的value还是不同的value，都没有存在的意义。