实验二：列表元组字典

请双击下面的单元格，填写你的姓名和学号：

班级	24计科
学号	未填写
姓名	未填写
Email	未填写

- 列表（list）
- 元组（tuple）
- 条件控制语句
- for循环语句
- if语句
- 字典(dict)
- while循环语句

实验注意事项

1. 请在指定的地方按照实验指导要求来编写代码。
2. 请按照实验指导要求使用指定的变量名或函数名，不要使用其他的名字。
3. 不要添加任何额外的语句。
4. 不要添加任何额外的代码单元格。
5. 不要在不需要的地方修改作业代码，比如创建额外的变量，修改测试文件中的代码。
6. 实验指导中的 `...` 表示需要你补充代码的部分，其他部分的代码不用修改。
7. 代码提示中会给出估计的代码行数，例如大约1行代码，估计的代码行数只是一个参考值，实际编写时可能会有出入，请根据实际情况来编写。
8. 请独立完成作业，禁止抄袭，发现抄袭行为成绩记零分

In [ ]:
```
# 导入测试代码
from testset2 import *
```

列表介绍

- 列表由一系列特定顺序(sequence)排列的元素组成。
- 在Python中，用方括号[]表示列表，用逗号分隔其中的元素。
- 有索引： 从0开始
- 最后一个元素后面的逗号会被忽略。

In [ ]:
```
bicycles = ['trek', 'cannondale', 'redline', 'specialized',]
print(bicycles)
```

python列表没有类型限制，列表中可以存放任意类型的元素

In [ ]:
```
elements = [3, 'hello', 2.5, True, 'world', ]
print(elements)
```

利用索引访问列表元素

In [ ]:
```
numbers = [1, 2, 3, 4, 5]

print(numbers[0])
print(numbers[4])

# 可以使用符数作为索引，-1表示最后一个元素
print(numbers[-1])
print(numbers[-5])
```

习题一

补全下面函数的代码，函数输入是一个列表(列表不会空)，函数返回列表的第一个元素和最后一个元素组成的列表

In [ ]:
```
def get_first_last(lst):
    """回列表的第一个元素和最后一个元素组成的列表"""
    # fist_elem = ...     # 获取列表的第一个元素
    # last_elem = ...     # 获取列表的最后一个元素
```

```
    # 大约2行代码
    # 你编写的代码从这里开始
    fist_elem = ...     # 获取列表的第一个元素
    last_elem = ...     # 获取列表的最后一个元素

    # 你编写的代码到这里结束
    return [first_elem,  last_elem]
```

In [ ]:
```
test_get_first_last(get_first_last)
```

操作列表(一)

与简单变量(str,int,float)不同：

- 列表是可变的
- 可以修改列表中的元素
- 可以添加和删除列表中的元素。

通过索引查找或者修改元素

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles[0] = 'ducati'
print(motorcycles)
```

在末尾附加元素：append方法

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles.append('ducati')
print(motorcycles)
```

在列表中插入元素： insert方法

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
motorcycles.insert(0, 'ducati')
print(motorcycles)
```

使用del语句删除元素

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
del motorcycles[0]
print(motorcycles)
```

pop语句删除列表末尾的元素并返回元素值

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
print(motorcycles)

popped_motorcycle = motorcycles.pop()
print(motorcycles)
print(popped_motorcycle)
```

pop语句删除列表任意位置的元素

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', ]
first_owned = motorcycles.pop(0)
print(f'The first motorcycle I owned was a {first_owned.title()}.')
print(motorcycles)
```

remove方法根据值删除列表中的元素

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
motorcycles.remove('ducati')
print(motorcycles)
```

如果列表中有相同的值，remove方法删除第一个匹配的值

In [ ]:
```
motorcycles = ['honda', 'yamaha', 'suzuki', 'honda']
motorcycles.remove('honda')
print(motorcycles)
```

习题二

补全下面函数的代码，函数输入是一个列表（不为空），函数返回删除列表中第一个元素，然后将删除的元素添加到列表的末尾

In [ ]:
```
def pop_and_append(lst):
```

```
    """删除列表的第一个元素，并将其添加到列表的末尾"""
    # first_elem = ...    # 删除列表的第一个元素
    #                     # 添加删除的元素到列表的末尾
    # 大约2行代码
    # 你编写的代码从这里开始


    # 你编写的代码到这里结束
    return lst
```

In [ ]:
```
test_pop_and_append(pop_and_append)
```

操作列表(二)

使用sort()方法对列表进行永久性(in place)排序

In [ ]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
cars.sort()
print(cars)
```

In [ ]:
```
cars.sort(reverse=True)
print(cars)
```

使用sorted()函数对列表进行临时排序

In [ ]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
print(f"The original list:\n {cars}\n")
print(f"The sorted list:\n {sorted(cars)}\n")
print(f"The original list:\n {cars}\n")
```

倒着打印列表：使用reverse()方法

In [ ]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru', ]
print(cars)
cars.reverse()
print(cars)
```

确定列表的长度： 使用len()函数

In [ ]:
```
len(cars)
```

很多其他数据类型和数据结构都可以使用len()函数

In [ ]:
```
len('hello world')
```

使用 `for` 循环语句遍历整个列表

In [ ]:
```
magicians = ['alice', 'david', 'carolina', ]
for magician in magicians:
    print(magician)
```

在Python语言中，使用 : 和缩进表示代码块开始，没有缩进来表示代码块结束

In [ ]:
```
magicians = ['alice', 'david', 'carolina', ]
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\n")

print("Thank you, everyone. That was a great magic show!")
```

使用 `range()` 函数创建数值列表

In [ ]:
```
for value in range(1, 5):
    print(value)
```

使用range()函数产生1到10的偶数, range()函数的第三个参数表示步长

In [ ]:
```
for value in range(2, 11, 2):
    print(value)
```

range()函数的步长可以是负数

In [ ]:
```
for value in range(5, 0, -1):
    print(value, end=' ')
```

使用range()函数创建数字列表

In [ ]:
```
numbers = list(range(1, 6))
print(numbers)
```

对数字列表进行统计计算

In [ ]:
```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, ]
min(digits)
```

In [ ]:
```
max(digits)
```

In [ ]:
```
sum(digits)
```

列表推导（List Comprehension）

In [ ]:
```
#列表推导格式：[ 变量表达式  for  变量  in  列表        ]                                                    ]
squares =    [ value**2  for  value  in range(1, 11)]
print(squares)
```

习题三

立方推导式 使用列表推导式生成一个列表，其中包含前 n 个整数的立方。

In [ ]:
```
def cubic_numbers(n):
    """返回1到n的立方数列表"""
    # cubes = ...    # 1到n的立方数列表
    # 大约1行代码
    # 你编写的代码从这里开始
    cubes = ...

    # 你编写的代码到这里结束
    return cubes
```

In [ ]:
```
test_cubic_numbers(cubic_numbers)
```

习题四

使用列表推导将列表中的名字全部变成首字母大写

```
scientists = ['albert einstein', 'marie curie', 'issac newton', 'charles darwin']
# 期望结果：  ['Albert Einstein', 'Marie Curie', 'Issac Newton', 'Charles Darwin']
```

In [ ]:
```
def get_titled_names(names):
    """返回首字母大写的名字列表"""
    # titled_names = ...    # 首字母大写的名字列表
    # 大约1行代码
    # 你编写的代码从这里开始
    titled_names = ...

    # 你编写的代码到这里结束
    return titled_names
```

In [ ]:
```
test_get_titled_names(get_titled_names)
```

列表切片（Slice）

打印列表的前三个元素，使用切片从0开始，到索引3（不包括索引3）结束

In [ ]:
```
players = ['charles', 'martina', 'michael', 'florence', 'eli', ]
print(players[0:3])  # 切片的长度等于3-0=3
```

In [ ]:
```
print(players[1:4])
```

切片从索引0开始的时候，可以省略第一个索引

In [ ]:
```
print(players[:4])
```

切片到列表末尾的时候，可以省略第二个索引

In [ ]:
```
print(players[2:])
```

切片的开始索引和结束索引都可以是负数， `players[-3:]` 表示最后三个元素

In [ ]:
```
print(players[-3:])
```

切片第二个冒号的后面是步长，当没有使用步长时，步长默认是1

In [ ]:
```
numbers = list(range(1, 11))
print(numbers[1:10:2])
```

步长可以为负数，负数步长可以用来倒着打印列表

In [ ]:
```
print(numbers[9:0:-2])
```

In [ ]:
```
print(numbers[-1:-10:-2])
```

切片的语法可以使用到很多顺序结构的数据上

In [ ]:
```
message = "Hello World!"
print(message[:5])
```

复制列表

- 列表变量和简单变量一样，都不是盒子，都是标签
- 列表变量不存储列表的值，而是存储列表的地址
- 两个列表变量可以指向同一个列表
- 和简单变量不同，列表变量是可变的，列表中存储的值可以改变
- 当两个列表变量指向同一个列表时，一个列表变量的改变会影响另一个列表变量

In [ ]:
```
my_food = ['pizza', 'falafel', 'carrot cake', ]
friend_food = my_food  # 两个列表指向同一个列表

# 两个列表变量的值会同时改变
friend_food.append('cannoli')
my_food.append('ice cream')

print(my_food)
print(friend_food)
print(id(my_food))
print(id(friend_food))
```

In [ ]:
```
my_food = ['pizza', 'falafel', 'carrot cake', ]

friend_food = my_food[:]   # 复制并创建一个新的列表

friend_food.append('cannoli')  # 只有friend_food列表的值会改变
my_food.append('ice cream')   # 只有my_food列表的值会改变

print(my_food)
print(friend_food)
print(id(my_food))
print(id(friend_food))
```

使用extend方法扩展列表,也可以使用 + 连接两个列表

In [ ]:
```
numbers = list(range(1, 11))

# extend方法扩展了原有列表
numbers.extend([11, 12, 13])
print(numbers)

# 使用加号连接两个列表，然后创建了新的列表连接
numbers = numbers + [14, 15]

print(numbers)
```

习题五

函数的输入是一个姓名列表，请复制这个列表，将列表中的姓名首字母大写，然后对姓名列表进行排序，最后返回排序后的列表

In [ ]:
```
def sort_capitalized_names(names):
    """返回首字母大写的名字列表，按字母顺序排序"""
    # sorted_names = ...    # 首字母大写的名字列表，按字母顺序排序
    # 大约1-2行代码
    # 你编写的代码从这里开始
    sorted_names = ...

    # 你编写的代码到这里结束
    return sorted_names
```

In [ ]:
```
test_sort_capitalized_names(sort_capitalized_names)
```

元组（Tuple）

使用圆括号来表示元组，圆括号中元组的元素之间用逗号分隔

In [ ]:
```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

遍历元组：元组和列表一样可以使用索引来遍历

In [ ]:
```
for dimension in dimensions:
    print(dimension)
```

元组中数据是不可以修改的

In [ ]:
```
# dimensions[0] = 250
```

只包含一个元素的元组，必须在元素后面加上逗号

In [ ]:
```
m_t = (3, )
m_t
```

元组的不可修改是相对的，如果元组包含了列表元素，该列表元素是可以被修改的，因此不要把可变对象(例如：list)保存到元组中.

In [ ]:
```
my_tuple = (1, 2, [10, 20, 30])
print(my_tuple)

my_tuple[2].append(40)
print(my_tuple)

my_tuple[2].extend([50, 60])
print(my_tuple)
```

if语句

cars = ['audi', 'bmw', 'subaru', 'toyota'] for car in cars: if car == 'bmw': print(car.upper()) else: print(car.title())

条件测试

if语句的核心是一个值为 `True` 或 `False` 的表达式

- `==` 和 `is` 操作符的区别
- 检测是否不相等
- 大于、小于
- 使用and检查多个条件
- 使用or检查多个条件
- 检查是否在列表中： in 操作符
- 检查某个数是不是为0，不为0表示True，0表示False

In [ ]:
```
car = 'bmw'
car == 'bmw'
```

`==` 操作符比较两个变量的值是否相等

In [ ]:
```
print(cars == cars[:])
print([1, 2, 3] == list(range(1, 4)))
print((1, 2, 3) == tuple(range(1, 4)))
print(1 == 1.0)
print(1+0j == 1)
print(1 == True)      # True == 1
print(0 == False)     # False == 0
```

`is` 操作符比较两个变量的id是否相同

In [ ]:
```
print(cars is cars[:])
print([1, 2, 3] is list(range(1, 4)))
print((1, 2, 3) is tuple(range(1, 4)))
print(1 is 1.0)
print(1+0j is 1)
print(1 is True)      # True == 1
print(0 is False)     # False == 0
```

如果两个变量使用 id() 函数返回相同的结果，对他们使用 is 会返回 True （充分必要条件）

In [ ]:
```
cars = ['audi', 'bmw', 'subaru', 'toyota']
print(cars == cars[:])
print(cars is cars[:])

my_cars = cars
print(my_cars == cars)
print(my_cars is cars)
```

In [ ]:
```
x = 1
y = 1.0
print(x == y)
print(x is y)
```

检测是否不相等： !=

In [ ]:
```
requested_toppings = 'mushrooms'
if requested_toppings != 'anchovies':
    print("Hold the anchovies!")
```

检测不是同一个对象： is not

In [ ]:
```
cars2 = cars[:]
print(cars2 is not cars)
```

数值比较大小： >, >=, <, <=

In [ ]:
```
age = 18
print(age > 18)
print(age >= 18)
print(age < 18)
print(age <= 18)
```

使用 and 和 or 检查多个条件

In [ ]:
```
age_0 = 22
age_1 = 18
print(age_0 >= 21 and age_1 >= 21)
print(age_0 >= 21 or age_1 >= 21)
```

- 使用 in 操作符检查特定值是否包含在列表、元组或者字符串中
- not in 检查特定值是否不包含在列表、元组或者字符串中
- 所有可迭代（iterative）的对象

In [ ]:
```
requested_toppings = ['mushrooms', 'onions', 'pineapple']
print('mushrooms' in requested_toppings)
print('onions' not in requested_toppings)
```

简化多个条件

In [ ]:
```
name = 'Jack'
pwd = '1234'
print((name, pwd) == ('Jack', '1234'))
```

In [ ]:
```
x = 1
print(x == 0 or x == 1)
print(x in (0, 1))
```

if-else 语句

In [ ]:
```
age = 17
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

if...else 表达式

In [ ]:
```
a, b = 10, 20
c = a if a else b
print(c)
```

if-elif-else 语句

In [ ]:
```
age = 12
if age < 4:
    print("Your admission cost is $0.")
elif age < 18:
    print("Your admission cost is $5.")
else:
    print("Your admission cost is $10.")
```

:= 海象（Walrus）运算符(Python 3.8+)

- `if (value := round(num)) > 0`
- 使用运算符右边对左边赋值
- 然后返回运算符左边的值

In [ ]:
```
# 不使用海象运算符
x = 60
y = 50

diff = x-y
if diff > 0:
    print(f"Diff is {diff}.")
```

In [ ]:
```
# 使用海象运算符
x = 60
y = 50

if (diff:= x-y) > 0:
    print("Diff is positive.")
```

海象运算符的文章

判断列表是否为空

In [ ]:
```
requested_toppings = []

if requested_toppings:
    for topping in requested_toppings:
        print(f"Adding {topping}.")
    print("\nFinished making your pizza!")
else:
    print("Are you sure you want a plain pizza?")
```

判断字符串是否为空

In [ ]:
```
msg = ''

if msg:
    print("msg is not empty")
else:
    print("msg is empty")
```

判断数值是否为0

In [ ]:
```
x = 0.000

if x:
    print("x is not zero")
else:
    print("x is zero")
```

In [ ]:
```
a = 0 + 0.00j
print(a == 0)
if a:
    print("a is not zero")
else:
    print("a is zero")
```

习题六

编写一个函数根据输入的里程和年龄计算火车票价，正常票价为里程乘以0.3，如果年龄低于6岁，火车票免费，如果是6到14岁，火车票打五折，如果是60以及60岁以上，火车票打七折，其他情况为正常票价。要求函数返回元组: (票价, 折扣)

例如：

- 输入里程100，年龄5，返回(0, 0)；
- 输入里程100，年龄10，返回(15, 0.5)；
- 输入里程100，年龄30，返回(30, 1)；
- 输入里程100，年龄65，返回(21, 0.7)；

In [ ]:
```
def get_ticket_price(miles, age):
    """根据里程以及乘客的年龄计算火车票价格"""
    # base_price = ...    # 火车票的基础价格
    # if age < ... :     # 如果乘客的年龄小于...
    #     discount = ...         # 折扣    #
    # elif  ...             # 其他的条件分支
    #     discount = ...
    # ticket_price = ...    # 根据折扣计算火车票价格
    # 大约10行代码
    # 你编写的代码从这里开始


    # 你编写的代码到这里结束
    return ticket_price, discount
```

In [ ]:
```
test_get_ticket_price(get_ticket_price)
```

字典(dict)

In [ ]:
```
# 创建字典
alien_0 = {'color': 'green', 'points': 5}

# 读取字典中的值
print(alien_0['color'])
print(alien_0['points'])
```

添加键值对

In [ ]:
```
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

创建空字典

In [ ]:
```
alien_0 = {}
print(alien_0)
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

修改字典中的值

In [ ]:
```
alien_0['color'] = 'yellow'
print(f"The alien is now {alien_0['color']}.")
```

删除键值对

In [ ]:
```
del alien_0['points']
print(alien_0)
```

由类似对象组成的字典

In [ ]:
```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
}

language = favorite_languages['sarah'].title()
print(f"Sarah's favorite language is {language}.")
```

如果键不存在，使用字典的[]语法会报错，这时应该使用get()方法来返回，如果键存在将返回键所对应的值，如果键不存在则返回一个默认值。

In [ ]:
```
alien_0 = {'color': 'green', 'speed': 'slow'}
```

```
    # print(alien_0['points'])
```

In [ ]:
```
print(alien_0.get('points', 'No point value assigned.'))
```

遍历字典键值对

In [ ]:
```
user_0 = {
        'username': 'efermi',
        'first': 'enrico',
        'last': 'fermi',
    }

    for key, value in user_0.items():
        print(f'\nKey:{key}')
        print(f'Value:{value}')
```

In [ ]:
```
for name, language in favorite_languages.items():
    print(f"{name.title()}'s favorite language is {language.title()}.")
```

遍历字典的键

In [ ]:
```
for name in favorite_languages.keys():
    print(name.title())
```

对字典的键进行排序

In [ ]:
```
print(sorted(favorite_languages.keys()))
```

遍历字典中的值

In [ ]:
```
for language in favorite_languages.values():
    print(language.title())
```

使用set()函数剔除重复项

In [ ]:
```
print(set(favorite_languages.values()))
```

set集合数据结构

- 不包含重复的元素
- 和dict同样使用 { } 来定义,但只包含键，没有值。
- 空的集合用 set() 表示, {} 表示的是空的字典

嵌套的数据结构： 字典列表

In [ ]:
```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}

aliens = [alien_0, alien_1, alien_2]
aliens
```

在字典中存储列表

In [ ]:
```
pizza = {
    'crust': 'thick',
    'toppings': ['mushrooms', 'extra cheese'],
}

print(pizza)
```

In [ ]:
```
favorite_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
}

print(favorite_languages)
```

在字典中存储列表

In [ ]:
```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
```

```
        'mcurie': {
            'first': 'marie',
            'last': 'curie',
            'location': 'paris',
            },
    }

    print(users)
```

while循环

当我们遍历的列表在循环过程中不断变化时，应该使用while循环，不能使用for循环

In [ ]:
```
# 未被确认的用户列表
unconfirmed_users = ['alice', 'brian', 'candace']

# 已经被确认的用户列表
confirmed_users = []

# 当列表不为空时，继续循环，在循环过程中列表元素会被删除或增加，这时应该使用while循环
while unconfirmed_users:
    current_user = unconfirmed_users.pop()
    print(f"Verifying user: {current_user.title()}")
    confirmed_users.append(current_user)

print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.title())
```

习题七

编写一个函数计算购物总额,输入为两个字典:

1. 商品单价字典 prices
2. 购买数量字典 quantities

函数返回元组:(总价格, 商品种类数)

测试用例:

```
prices = {'apple': 5, 'banana': 3, 'orange': 4}
quantities = {'apple': 2, 'banana': 3}
calculate_total(prices, quantities)  # 返回 (19, 2)

prices = {'apple': 5, 'banana': 3, 'orange': 4}
quantities = {'orange': 1}
calculate_total(prices, quantities)  # 返回 (4, 1)
```

注意：测试用例的数据中quantities字典的键一定会出现在prices字典中。

In [ ]:
```
def calculate_total(prices, quantities):
    """计算商品总价"""
    # total = ...    # 初始化总价
    # count = ...    # 初始化商品种类数
    # for   ...  :    # 使用for循环或者while循环遍历quantities字典
    #   price = ...       # 获取商品的单价
    #   quantity = ...    # 获取商品的数量
    #   total += ...      # 更新商品的总价
    #   count += ...      # 更新商品的种类数
    # return total, count
    # 大约5 - 8行代码
    # 你编写的代码从这里开始



    # 你编写的代码到这里结束
    return total, count
```

In [ ]:
```
test_calculate_total(calculate_total)
```

实验自动评分

请运行下面的代码进行实验自动评分

In [ ]:
```
grade_all_tests([get_first_last, pop_and_append, cubic_numbers,
                 get_titled_names, sort_capitalized_names,
                 get_ticket_price, calculate_total])
```

实验总结

请尽量使用自己的语言回答下面的问题。

问题一： list是Python中最常用的数据结构，请问list主要有哪些操作，列举至少6个操作，分析一下这些操作中哪些是效率较高的，哪些是效率较低的？

问题2： 分析为什么遍历的列表在循环过程中不断变化时，应该使用while循环，不能使用for循环？

生成实验报告

In [ ]:
```
import sys
import os

# Add the root directory (parent directory of `util`) to sys.path
root_dir = os.path.abspath("..")  # Adjust based on the structure
sys.path.append(root_dir)

from util import notebook2pdf, notebook_info_extractor

notebook_file = "实验二-列表元组字典.ipynb"
stu_info  = notebook_info_extractor.extract_from_ipynb(notebook_file)

html_file = "notebook.html"
pdf_file = f"{stu_info['class_id']}-{stu_info['student_id']}-{stu_info['name']}.pdf"

notebook2pdf.notebook_to_html(notebook_file, html_file)
notebook2pdf.html_to_pdf(html_file, pdf_file)
```