归并排序不考

def # 定义函数

In [1]:

```
def add():
    result = 11 + 22
    print(result)
```

In [2]:

```
def add_modify(a, b):
    result = a + b
    print(result)
```

调用函数

In [3]:

```
add()
add_modify(10, 20)
```

```
33
30
```

In [4]:

```
def add_modify(a, b):
    result = a + b
    add()
    print(result)
```

In [5]:

```
add_modify(10, 20)
```

```
33
30
```

In [6]:

```
def add_modify(a, b):
    result = a + b
    print(result)
    def test():                                    # 在函数中定义函数test()
        print("我是内层函数")
add_modify(10, 20)
```

```
30
```

In [7]:

```
def add_modify(a, b):
    result = a + b
    print(result)
    def test():                                    # 在函数中定义函数test()
        print("我是内层函数")
    test()                                         # 在函数中调用函数test()
add_modify(10, 20)
```

```
30
我是内层函数
```

函数参数的传递

位置参数的传递·

In [8]:

```
def get_max(a, b):
    if a > b:
        print(a,"是较大的值！")
    else:
        print(b,"是较大的值！")
get_max(8, 5)
```

```
8 是较大的值！
```

关键字参数的传递

In [9]:

```
def connect(ip, port):
    print(f"设备{ip}:{port}连接！")
connect(ip="127.0.0.1", port=8080)
```

```
设备127.0.0.1:8080连接！
```

In [10]:

```
def func(a, b, /, c):  # / 指明前面的参数a、b均为仅限位置形参
    print(a, b, c)
```

In [11]:

```
# 错误的调用方式
# func(a=10, 20, 30)
# func(10, b=20, 30)
# 正确的调用方式
func(10, 20, c=30)
```

```
10 20 30
```

默认参数的传递

In [12]:

```
def connect(ip, port=8080):
    print(f"设备{ip}:{port}连接！")
```

In [13]:

```
connect(ip="127.0.0.1")
connect(ip="127.0.0.1", port=3306)
```

```
设备127.0.0.1:8080连接！
设备127.0.0.1:3306连接！
```

参数的打包与解包

In [14]:

```
def test(*args):
    print(args)
```

In [15]:

```
test(11, 22, 33, 44, 55)
```

```
(11, 22, 33, 44, 55)
```

In [16]:

```
def test(**kwargs):
    print(kwargs)
```

In [17]:

```
test(a=11, b=22, c=33, d=44, e=55)
```

```
{'a': 11, 'b': 22, 'c': 33, 'd': 44, 'e': 55}
```

In [18]:

```
def test(a, b, c, d, e):
    print(a, b, c, d, e)
```

In [19]:

```
nums = (11, 22, 33, 44, 55)
test(*nums)
```

```
11 22 33 44 55
```

In [20]:

```
nums = {"a":11, "b":22, "c":33, "d":44, "e":55}
test(**nums)
```

```
11 22 33 44 55
```

混合传递

In [21]:

```
def test(a, b, c=33, *args, **kwargs):
    print(a, b, c, args, kwargs)
```

In [22]:

```
test(1, 2)
test(1, 2, 3)
test(1, 2, 3, 4)
test(1, 2, 3, 4, e=5)
```

```
1 2 33 () {}
1 2 3 () {}
1 2 3 (4,) {}
1 2 3 (4,) {'e': 5}
```

In [23]:

```
def filter_sensitive_words(words):
    if "山寨" in words:
        new_words = words.replace("山寨", "**")
        return new_words
```

In [25]:

```
result = filter_sensitive_words("这个手机是山寨版吧！")
print(result)
```

```
这个手机是**版吧！
```

In [24]:

```
def move(x, y, step):
    nx = x + step
    ny = y - step
    return nx, ny  # 使用return语句返回多个值
result = move(100, 100, 60)
print(result)
```

```
(160, 40)
```

变量作用域

局部变量和全局变量`

In [26]:

```
def test_one():
    number = 10       # 局部变量
    print(number)     # 函数内部访问局部变量
test_one()
print(number)
```

```
10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_21040/569783001.py in <module>
      3     print(number)     # 函数内部访问局部变量
      4 test_one()
----> 5 print(number)

NameError: name 'number' is not defined
```

In [ ]:

```
def test_one():
    number = 10
    print(number)  # 访问test_one()函数的局部变量number
def test_two():
    number = 20
    print(number)  # 访问test_two()函数的局部变量number
test_one()
test_two()
```

In [30]:

```
number = 10          # 全局变量
def test_one():
    print(number)    # 函数内部访问全局变量
test_one()
print(number)
```

```
10
10
```

In [27]:

```
# 定义全局变量
number = 10
def test_one():
    print(number)  # 在函数内部访问全局变量
    number += 1    # 在函数内部直接修改全局变量
test_one()
print(number)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_21040/621636121.py in <module>
      4     print(number)  # 在函数内部访问全局变量
      5     number += 1    # 在函数内部直接修改全局变量
----> 6 test_one()
      7 print(number)

~\AppData\Local\Temp/ipykernel_21040/621636121.py in test_one()
      2 number = 10
      3 def test_one():
----> 4     print(number)  # 在函数内部访问全局变量
      5     number += 1    # 在函数内部直接修改全局变量
      6 test_one()

UnboundLocalError: local variable 'number' referenced before assignment
```

global和nonlocal关键字`

In [28]:

```
number = 10                    # 定义全局变量
def test_one():
    global number               # 使用global声明变量number为全局变量
    number += 1
    print(number)
test_one()
print(number)
```

```
11
11
```

In [33]:

```
def test():
    number = 10
    def test_in():
        nonlocal number
        number = 20
    test_in()
    print(number)
test()
```

```
20
```

递归函数

In [36]:

```
def func(num):
    if num == 1:
        return 1
    else:
        return num * func(num - 1)
num = int(input("请输入一个整数："))
result = func(num)
print("5!=%d"%result)
```

```
请输入一个整数：5
5!=120
```

位置参数·

In [113]:

```
def add(a, b):
    print('In function:a={},b={}'.format(a,b))
    return a+b

print(add(3, 5))
```

```
In function:a=3,b=5
8
```

默认值参数

In [115]:

```
def add(a, b=5):
    print('In function:a={},b={}'.format(a,b))
    return a+b
print(add(3))
print(add(3, 8))
```

```
In function:a=3,b=5
8
In function:a=3,b=8
11
```

关键参数

In [117]:

```
def add(a, b):
    print('In function:a={},b={}'.format(a,b))
    return a+b

print(add(b=8, a=3))
```

```
In function:a=3,b=8
11
```

可变长度参数 *args(转成元祖) *kw（转成字典）

*args:

1. 关键标志为星号*，名称可以随意
2. 当传入函数中的参数个数未知，且不需要知道参数名称时，使用*args。
3. 传入函数中的几个参数组成元组

)

**kwargs:

1. 关键标志为星号**，名称可以随意。
2. 当传入函数中的参数个数未知但需要知道参数的名称时，使用**kw。
3. 传入函数中的几个参数组成字典。

In [121]:

```
def func_arg(farg, *args):
    print("formal arg:", farg)
    print(args)
func_arg(1,"youzan",'dba','2333')
'''总结:
    参数1传入给farg
    剩下的3个参数传入*arg并组成元组'''
```

```
formal arg: 1
('youzan', 'dba', '2333')
```

Out[121]:

```
'总结: \n    参数1传入给farg\n    剩下的3个参数传入*arg并组成元组'
```

In [122]:

```
def func_kw(farg, **kw):
    print("formal kw:", farg)
    print(kw)
func_kw(1,a="youzan",b='dba',c='2333')
'''总结:
    参数1传入给farg
    剩下的3个参数传入*arg3并组成字典'''
```

```
formal kw: 1
{'a': 'youzan', 'b': 'dba', 'c': '2333'}
```

Out[122]:

```
'总结:\n    参数1传入给farg\n    剩下的3个参数传入*arg3并组成字典'
```

In [119]:

```
def add(a, b, *args, **kwargs):
    print('In function:\na={}\nb={}\nargs={}\nkwargs={}'.format(a, b,args, kwargs))
    return a+b+sum(args)+sum(kwargs.values())

print(add(3, 8, 1, 2, 3, 4, x=5, y=6, z=7))
print('='*20)
print(add(1, 2, 3, 4, 5, 6, 7, 8, 9, x=10, y=11))
```

```
In function:
a=3
b=8
args=(1, 2, 3, 4)
kwargs={'x': 5, 'y': 6, 'z': 7}
39
====================
In function:
a=1
b=2
args=(3, 4, 5, 6, 7, 8, 9)
kwargs={'x': 10, 'y': 11}
66
```

count ()

```
Python count() 方法用于统计字符串里某个字符出现的次数。可选参数为在字符串搜索的开始与结束位置。
count()方法语法：str.count(sub, start= 0,end=len(string))
 参数
sub -- 搜索的子字符串
start -- 字符串开始搜索的位置。默认为第一个字符,第一个字符索引值为0。
end -- 字符串中结束搜索的位置。字符中第一个字符的索引为 0。默认为字符串的最后一个位置。
 返回值：整型
该方法返回子字符串在字符串中出现的次数。
```

In [111]:

```
str = "i love python,i am learning python"

print(str.count("i")) #star 和end 为默认参数
print(str.count("i",2)) # star值为2，end值为默认参数
print(str.count("i",2,5)) #star值为2，end值为5
print(str.count("am")) #多字符统计
```

```
3
2
0
1
```

eval函数

```
# eval()函数用来执行一个字符串表达式，并返回表达式的值。还可以把字符串转化为list、tuple、dict。
eval函数的语法：eval(expression[,globals[,locals]])

参数：
expression：表达式。
globals：变量作用域，如果被提供，必须是一个字典对象。
locals：变量作用域，如果被提供，可以说任何映射对象。
```

In [106]:

```
a="[1,2,3,4,5]"
b=eval(a)
 # a是字符串类型数据，b是列表类型数据
print(b)
a="{'name':'guo','age':25}"
b=eval(a)
# a为字符串类型数据，b为字典类型数据
print(b)

#元祖也是如此
```

```
[1, 2, 3, 4, 5]
{'name': 'guo', 'age': 25}
```

In [109]:

```
a=10;
print(eval("a+1"))

a=10;
g={'a':4}
print(eval("a+1",g))
'''【解析】：因为现在指定了globals，所以在expression部分的作用域就
是globals指定的字典范围内。所以此时外面的a=10被屏蔽，取用字典中的值'''
a=10
b=20
c=30
g={'a':6,'b':8}
t={'b':100,'c':10}
print(eval('a+b+c',g,t))
```

```
11
5
116
```

升序降序

In [82]:

```
from random import shuffle

data = list(range(5))
shuffle(data)                   # 随机打乱顺序
print(data)
print(sorted(data))             # 升序排序
print(sorted(data, key=str))    # 按转换成字符串后的大小升序排序
print(sorted(data, key=str,reverse=True))    # 降序排序
```

```
[1, 4, 3, 2, 0]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4]
[4, 3, 2, 1, 0]
```

In [84]:

```
from random import shuffle

data = list(range(5))          # 创建列表
shuffle(data)                  # shuffle 随机打乱顺序
print(data)
reversedData = reversed(data) # 生成reversed对象
print(reversedData)
print(list(reversedData))      # 根据reversed对象得到列表
print(tuple(reversedData))     # 空元组，reversed对象中元素只能使用一次
```

```
[2, 4, 0, 1, 3]
<list_reverseiterator object at 0x000001D069C0FEE0>
[3, 1, 0, 4, 2]
()
```

functools operator是一个模块 详情可查阅资料

lambda （）

In [ ]:

```
1 作用：通常是用来在python中创建匿名函数的
2 格式： lambda 参数[,参数] : 表达式
（1）lambda定义的是单行函数， 如果需要复杂的函数，应该定义普通函数
（2）lambda参数列表可以包含多个参数，例如 lambda x,y : x + y
（3）lambda中的表达式不能含有命令， 而且只限一条表达式
```

In [39]:

```
add = lambda x,y:x+y
add(3,4)
```

Out[39]:

```
7
```

In [40]:

```
my_list = [3,5,-4,-1,0,-2,-6]
sorted(my_list, key=lambda x: abs(x))
```

Out[40]:

```
[0, -1, -2, 3, -4, 5, -6]
```

map()

map是python内置函数，会根据提供的函数对指定的序列做映射。 map()函数的格式是：
map(function,iterable,...) 第一个参数接受一个函数名，后面的参数接受一个或多个可迭代的序列，返回的是一个集合。 把函数依次作用在list中的每一个元素上，得到一个新的list并返回。注意，map不改变原list，而是返回一个新list。

In [29]:

```
# 定义一个函数
def square(x):
    return x**2
# 序列
nums = [1, 2, 3, 4, 5]
# 对nums序列每个数求平方，返回迭代器
nums_squared = map(square, nums)
# 输出结果
for num in nums_squared:
    print(num,end=" ")
```

```
1 4 9 16 25
```

In [61]:

```
nums = [1, 2, 3, 4, 5]
nums_squared = map(lambda x: x*x, nums)
for num in nums_squared:
    print(num,end='  ')
```

```
1  4  9  16  25
```

In [59]:

```
def multiply(x, y):
    return x * y
nums1 = [1, 2, 3, 4, 5]   # 序列1
nums2 = [6, 7, 8, 9, 10]  # 序列2
mult = map(multiply, nums1, nums2) # 每次从nums1和nums2中取值
for num in mult:
    print(num,end="  ")
```

```
6  14  24  36  50
```

使用map()映射多个函数。

In [55]:

```
def add(x):      # 函数1
    return x + x
def square(x):   # 函数2
    return x * x
nums = [1, 2, 3, 4, 5]

for i in nums:    # 逐个取数处理
    # lambda为处理函数，分别将add和square传给lambda
    vals = list(map(lambda x: x(i), (add, square)))
    print(vals,end='   ')
```

```
[2, 1]   [4, 4]   [6, 9]   [8, 16]   [10, 25]
```

列表推导式

In [50]:

```
def square(x): # 定义函数
    return x * x
nums = [1, 2, 3, 4, 5]
# 列表推导式相当于每次从nums中取一个数放入square中
nums_squared = [square(num) for num in nums]
for num in nums_squared:
    print(num,end=" ")
```

```
1 4 9 16 25
```

reduce （

rudece()函数

函数模型：

```
reduce(function,iterable[,initializer])
```

iterable:可迭代的对象

function:可接收两个参数的函数

initializer：可有可无，存放的是初始值

reduce函数的返回值是：function对iterable进行计算的最终结果

)

reduce工作原理

对于我们上面的代码，reduce的工作原理是像下面这张图所示的

57647

首先，reduce会计算列表中第一个和第二个元素的和，然后把这个结果和第三个元素相加，然后再把新的这个计算结果和第四个元素相加，每一次都是上一次计算的结果和下一个元素相加，所以这样就实现了求和运算。

In [45]:

```
from functools import reduce
def add(x,y):
    return x+y
l=[1,2,3,4,5]
res=reduce(add,l,100)
res
```

Out[45]:

115

In [47]:

```
from functools import reduce
from operator import add, mul, or_

seq = range(1, 10)
print(reduce(add, seq))       # 累加seq中的数字
print(reduce(mul, seq))       # 累乘seq中的数字
seq = [{1}, {2}, {3}, {4}]
print(reduce(or_, seq))       # 对seq中的集合连续进行并集运算
```

```
45
362880
{1, 2, 3, 4}
```

In [49]:

```
print(reduce(lambda x,y:x+y,[1,2,3,4,5]))
print(reduce(lambda x,y:x+y,[1,2,3,4,5],100))
```

```
15
115
```

In [62]:

```
print(lambda x,y:x+y,[1,2,3,4,5])
print(lambda x,y:x+y,[1,2,3,4,5],100)
```

```
<function <lambda> at 0x000001D069C725E0> [1, 2, 3, 4, 5]
<function <lambda> at 0x000001D069C725E0> [1, 2, 3, 4, 5] 100
```

filter()

filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。

该接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为 参数传递 给函数进行判，然后返回 True 或 False，最后将返回 True 的元素放到新列表中。

In [71]:

```
l = [x for x in range(10)]
print(list(filter(lambda x : x%2 == 0, l)))
```

```
[0, 2, 4, 6, 8]
```

In [68]:

```
def is_odd(n):
    """
    过滤出列表中的所有奇数：
    :param n:
    :return:
    return n % 2 == 1
    """
    return n % 2 == 1
newlist = filter(is_odd, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
# <filter object at 0x0000028702478A90> <class 'filter'>
print(newlist,type(newlist))
for i in newlist:
    print(i,end="  ")
```

```
<filter object at 0x000001D069C7AF10> <class 'filter'>
1  3  5  7  9
```

In [70]:

```
#过滤出1~100中平方根是整数的数：
import math

def is_sqr(x):
    return math.sqrt(x) % 1 == 0

newlist = filter(is_sqr, range(1, 101))
print(newlist)
for i in newlist:
    print(i,end="  ")
```

```
<filter object at 0x000001D069C7A670>
1  4  9  16  25  36  49  64  81  100
```

find 方法

In [ ]:

```
寻找子字符串在某个字符串中的位置三种参数使用形式:
第1个参数:子字符串
第2个参数:开始查找的位置【可以省略,如果省略，默认从0开始】
第3个参数:结束查找的位置【可以省略,如果省略，默认从最后一个位置开始】

#注意:查找返回的结果是子字符串在父字符串中的第一次出现的位置的索引值，如果没有找到则返回负一
```

In [1]:

```
s = "刘金玉编程,编程创造城市。博客http://ljy.kim"
#查找编程两全字所在的位置
print(s.find("编程"))
print(s.find("编程1"))
print(s.find("编程",4))
print(s.find("编程",4,7))
print(s.find("编程",4,11))
```

```
3
-1
6
-1
6
```

index方法

In [ ]:

```
比较index方法
返回的是一个子字符串在原字符串中的第一次出现的位置
''''区别: index如果没有找到子字符串，会直接报异常。而find直接返回-1
```

字符串的截取

字符串可以看做一个序列(数组)采用字符串的分片技术

In [2]:

```
s = "刘金玉编程，编程创造城市。博客http://ljy.kim"
#查找编程两个字所在的位置
print(s.find("编程",4))
startIndex = s.find("编程",4)   #下面的从这里开始
print (s[startIndex:startIndex+6])
```

```
6
编程创造城市
```

range

In [ ]:

```
range()函数
• 用于生成一个整数序列
• 创建range对象的三种方式
……range(stop)  创建一个[0,stop)之间的整数序列步长为1
……range(start,stop)创建一个[start,stop)之间的整数序列，步长为1
……range(start,stop,step)创建一个[start,stop)之间的整数序列步长为step
• 返回值是一个迭代器对象
range类型的优点:不管range对象表示的整数序列有多长，所有range对象占用的内存空间都是相同的,
因为仅仅需要存储start,stop和step，只有当用到range对象时，才会去计算序列中的相关元素
in与not in判断整数序列中是否存在(不存在)指定的整数
```

In [3]:

```
#range()的三种创建方式
'''第一种创建方式，只有一个参数（小括号中只给了一个数)'''
r = range(10) #【0，1,2,3,4,5,6,7,8,9】 默认为 0 开始，默认相差 1 为步长
print(r)  # range(0, 10)
print(list(r))#用于查看range对象中的整数序列 —> list 列表的意思

'''第二种创建方式，给了两个参数（小括号中给了两个数'''
r = range(1,10) #指定了起始值，从1开始到10结束，默认步长为1
print(list(r)) #[1, 2, 3, 4, 5, 6, 7, 8, 9]

'''第三种创建方式，给了三个参数（小括号中给了三个数）'''
r = range(1,10,2)
print(list(r)) # [1, 3, 5, 7, 9]

'''判断指定的整数在序列中是否存在 in ,not in
   这里 是周后一个print的输入的 r 里面 包含的是1.3.5.7.9'''
print(10 in r) #False ,10不在当前的r这个整数序列中
print(9 in r) #True,9在当关的r这个序列中
print(10 not in r) #True
print(9 not in r) #Fals
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
False
True
True
False
```

zip

zip函数的原型为：zip([iterable, ...])

参数iterable为可迭代的对象，并且可以有多个参数。该函数返回一个以元组为元素的列表，其中第 i 个元组包含每个参数序列的第 i 个元素。返回的列表长度被截断为最短的参数序列的长度。只有一个序列参数时，它返回一个1元组的列表。没有参数时，它返回一个空的列表。

In [4]:

```
data = zip('1234', [1, 2, 3, 4, 5, 6])
print(data)
# 在转换为列表时，使用了zip对象中的全部元素，zip对象中不再包含任何内容
print(list(data))
# 如果需要再次访问其中的元素，必须重新创建zip对象
data = zip('1234', [1, 2, 3, 4, 5, 6])
print(tuple(data))
data = zip('1234', [1, 2, 3, 4, 5, 6])
# zip对象是可迭代的，可以使用for循环逐个遍历和访问其中的元素
for item in data:
    print(item)
```

```
<zip object at 0x000001D069C15900>
[('1', 1), ('2', 2), ('3', 3), ('4', 4)]
(('1', 1), ('2', 2), ('3', 3), ('4', 4))
('1', 1)
('2', 2)
('3', 3)
('4', 4)
```

In [5]:

```
data = zip('1234', [1, 2, 3])
print(tuple(data))
```

```
(('1', 1), ('2', 2), ('3', 3))
```

In [79]:

```
names = ['zhangsan','lisi','wangwu','zhangliu','sunqi']
weights = [72,68,72,66,76]

user_weight = dict(zip(names,weights))
#输出：{'zhangsan': 72, 'lisi': 68, 'wangwu': 72, 'zhangliu': 66, 'sunqi': 76}
print(user_weight)
```

```
{'zhangsan': 72, 'lisi': 68, 'wangwu': 72, 'zhangliu': 66, 'sunqi': 76}
```

math

In [5]:

```
import math
print(math.fsum([0.1,0.2,0.3])) #对浮点数精确求和函数

print(math.pow(10,1/3))# pow() math.pow( x, y ) 方法返回 xy（x 的 y 次方） 的值。

print(math.atan(1))# atan() math.atan(x) 返回x的反正切弧度值。

print(math.gamma(11))# 伽玛函数

print(round(10.0/3,2)) # round函数对浮点数进行近似取值
print(round(10.0/3,4)) # round函数对浮点数进行近似取值

print(math.pi)
print(math.e)
```

```
0.6
2.154434690031884
0.7853981633974483
3628800.0
3.33
3.3333
3.141592653589793
2.718281828459045
```

max()

In [95]:

```
data = ['3', '22', '111']  #  '' 字符串的
print(max(data))
data = ['abc', 'Abcd', 'ab']
# 最大的字符串
print(max(data))
# 长度最大的字符串
print(max(data, key=len))
```

```
3
abc
Abcd
```

In [97]:

```
data = [3, 22, 111]
print(data)
# 对列表中的元素直接比较大小，输出最大元素
print(max(data))
print(min(data))
# 返回转换成字符串之后最大的元素
print(max(data, key=str))
```

```
[3, 22, 111]
111
3
3
```

len()

In [123]:

```
data = [1, 2, 3, 4]
# 列表中元素的个数
print(len(data))
data = 'Readability counts.'
print(len(data))
c={'a':1,'b':2}
len(c)
```

```
4
19
```

Out[123]:

```
2
```

sum ()

In [89]:

```
data = (1, 2, 3)
print(len(data))
print(sum(data))
data = {97: 'a', 65: 'A', 48: '0'}
print(len(data))
print(sum(data))
```

```
3
6
3
210
```

列子·

In [ ]:

```
def print_menu():
    print('========================')

    print('1.添加学生信息')
    print('2.删除学生信息')
    print('3.修改学生信息')
    print('4.查询学生信息')
    print('0.退出系统')
    print('========================')
def add_stu_info(stu_info,id):
    name=input('请输入新学生的姓名：')
    sex = input('请输入新学生的性别：')
    if (sex!='男' and sex!='女'):
        print('您的输入不合法，请重新输入！')
    else:

        iphone = input('请输入新学生的手机号码：')
        stu={'序号':id,'姓名':name,'性别':sex,'手机号码':iphone}
        stu_info.append(stu)
def del_stu_info(stu_info,dd):
    if len(stu_info)!=0 :
        dl=int(input('请输入要删除的序号：'))                          #dl为删除序号
        if dl in dd or dl<1 or dl>len(stu_info):
            print('查询不到该生信息！！')
        else:
            del stu_info[dl-1]
            dd.append(dl)
def modify_stu_info(stu_info,dd):
    reid=int(input('请输入要修改学生的序号：'))
    if reid<1 or reid in dd or reid>len(stu_info):
        print('您输入的序号不存在')
    else:
        rename = input('请输入要修改学生的姓名：')
        resex = input('请输入要修改学生的性别（男/女）：')
        reiphone = input('请输入要修改学生的手机号码：')
        stu_info[reid-1]['姓名']=rename
        stu_info[reid - 1]['性别'] = resex
        stu_info[reid - 1]['手机号码'] = reiphone
def show_stu_info(stu_info):
    if len(stu_info)==0:
        print('列表为空')
    else:
        print('序号  姓名    性别   手机号码')
        for i in stu_info:
            for j in i.values():

                print(j,end='    ')
            print('')
def main():
    stu_info=[]                                    #stu_info创建学生空列表
    dd=[]                                          #dd创建删除学生列表
    id=int(0)                                      #自动生成序号
    while True:
        print_menu()
        ip=input('请输入功能对应的数字：')
        if ip=='1':
            id+=1
            add_stu_info(stu_info,id)
        if ip=='2':
```

```
            del_stu_info(stu_info,dd)
        if ip=='3':
            modify_stu_info(stu_info,dd)
        if ip=='4':
            show_stu_info(stu_info)
        if ip=='0':
            ip1=input('亲，真的要退出吗？（yes or no）:')
            if ip1=='yes':
                print('谢谢使用！')
                break
            if ip1=='no':
                continue
            if ip1!='yes' or ip1!='no':
                print('您的输入不合法！')
if __name__=='__main__':
    main()
```

```
========================
1.添加学生信息
2.删除学生信息
3.修改学生信息
4.查询学生信息
0.退出系统
========================
```

In [ ]:

```

```

In [ ]:

```

```