基础数据类型

字符串

1、转义字符

（1）概念“\”及其后面的某些字符会构成转义字符，即两个字符当一个看

（2）种类：①\n换行②\t制表③\”双引号④\斜杠

*反斜杠()可以作为续行符，表示下一行是上一行的延续。也可以使用 """...""" 或者 '''...''' 跨越多行（字符串和条件语句需要加“\”，列表和函数调用则不必）

```
if condition1 and condition2 and condition3 and \
   condition4 and condition5:
    print("All conditions are met.")
```

*字符包括\n这样的转义字符只能出现在字符串里面，必须用引号括起来

*字符串前面加r表示字符串里的\就是\，不会和后面的字符合并起来看待

```
print(r'ab\ncd')  #>>ab\ncd
```

```
>>> print('Ru\noob')
Ru
oob
>>> print(r'Ru\noob')
Ru\noob
>>>
```

2、字符串的切片（子串）

（1）a[x:y]表示字符串a里从下标x到下标y那一部分的子串（不包括下标y的那个字符），终点省略就是一直取到最后一个字符，起点省略就是从头开始取

（2）a[x:y:z]，从a[x]取到a[y]（a[y]不算），每z个字符取一个，最后拼起来，z为负数则代表倒着取，x，y全省略表示从头取到尾或从尾取到头

*字符串切片的用法也适用于元组和列表

*字符串不能使用负数索引（-1，-2，……，-n）

*字符串乘法：*表示复制n次

3、字符串的分割

（1）split函数：s.split（x），用字符串x做分隔符分割字符串s，得到分隔后的列表，两个相邻分隔符之间会被分隔出一个空串

（2）通过正则表达式用多个分隔串进行分割

import re

re.split(x,s)：用正则表达式x里面的分隔串分割s，x里不同分隔串用“|”隔开，一些特殊字符，比如: ? ! " ' ( ) | * $ \ [ ] ^ { } . 在正则表达式里出现时，前面需要加 \，两个相邻分隔符之间会隔出一个空串

```
import re
a = 'Beautiful, is; beoktter*than\nugly'
print(re.split(';| |,|\*|\n|ok',a)) #分隔串用 | 隔开]

';' ' ' ',' '*' '\n' 'ok' 都被看作分隔串

#>>['Beautiful', '', 'is', '', 'be', 'tter', 'than', 'ugly']
```

4、字符串的函数

（1）count求子串出现次数：s.count(“AA”)

（2）len求字符串长度：len(s)

（3）upper, lower转大写、小写，不改变s本身：s.upper();

使用 capitalize() 方法可以将字符串的第一个字符转换为大写;

使用 swapcase() 方法可以将字符串中的大写字母转换为小写，小写转换为大写

（4）find（从头），rfind（从尾），index，rindex在字符串中查找子串

```
在字符串中查找子串，返回找到的位置(下标)。找不到的话，find
返回-1,index 引发异常
s="1234abc567abc12"
print(s.find("ab")) #>> 4 , "ab"第一次出现在下标4
print(s.rfind("ab")) #>>10
#find从头开始找，rfind从尾巴开始找，返回第一个找到的位置
try :
      s.index("afb") #找不到"afb"因此会产生异常
except Exception as e:
      print(e) #>> substring not found
```

*find还可以指定查找起点

```
s="1234abc567abc12"
print(s.find("12",4))  #>>13 指定从下标4处开始查找
```

（5）replace替换

```
s="1234abc567abc12"
b = s.replace("abc","FGHI") #b由把s里所有abc换成FGHI而得
print(b)    #>> 1234FGHI567FGHI12
print(s)    #>> 1234abc567abc12
print(s.replace("abc","")) #>> 123456712
```

（6）isdigit(), islower(), isupper()判断字符串是否全是数、小写、大写等

（7）startwith, endwith判断字符串是否以某子串开头、结尾

```
print("abcd".startswith("ab"))  #>> True
print("abcd".endswith("bcd"))   #>> True
print("abcd".endswith("bed"))   #>> False
```

（8）strip()返回除去头尾空白字符（空格，“\r”，“\t”，“\n”）后的字符串

lstrip()返回除去头部（左端）空白字符后的字符串

rstrip()返回除去尾部（右端）空白字符后的字符串

strip(s), lstrip(s), rstrip(s) 返回除去两端、左端、右端在s中出现的字符后的字符串

```
 print ( " \t12 34 \n ".strip())  #>>12 34
 print ( " \t12 34 5".lstrip())  #>>12 34 5
```

```
print ( "takeab \n".strip("ba \n"))  #>>take
#去除两端 'b','a',' ','\n'
print ( "cd\t12 34 5".lstrip("d\tc")) #>>12 34 5
#去除左端 'd','\t','c'
```

5、字符串的编码和格式化

（1）字符串的编码在内存中的编码是unicode（通常是两个字节）的，写入文件时可能是gbk（一个字节或两个字节）或者utf-8（可能三个字节）的

（2）字符串的格式化

```
x = "Hello {0} {1:10},you get ${2:0.4f}".format("Mr.","Jack",3.2)
print(x) #>> Hello Mr. Jack       you get $3.2000
x = "Hello {0} {1:>10},you get ${2:0.4f}".format("Mr.","Jack",3.2)
print(x) #Hello Mr.       Jack,you get $3.2000

{序号: 宽度.精度 类型}       宽度可以是0
> : 右对齐
< : 左对齐
^ : 中对齐
如 {0:>10.4f} 表示第0项是小数，以宽度至少是10字符，右对齐（宽度不足时空格补在左边），保留小数点后第4位的方式输出。


print("Today is %s.%d." % ('May',21))
# Today is May.21.
```

元组

1、概念：一个元组由数个逗号分隔的值组成，前后可加括号

*构造包含0个或1个元素的元组比较特殊，所以有一些额外的语法规则

```
tup1 = ()    # 空元组
tup2 = (20,) # 一个元素，需要在元素后添加逗号
```

2、特点

（1）元组不能修改，即不可增删元素，不可对元素赋值，不可修改元素顺序（如排序）

（2）元组的元素的内容有可能被修改，如若元素是列表就可以修改该列表

*元组的元素都是指针。元组元素不可修改，是指不可改变元组元素的指向，但是元组元素指向的内容，是有可能被修改的

3、表示：末尾加逗号，否则为字符串，如空元组和单元素元组

```
empty = ()    #空元组
singleton = 'hello',    #注意末尾的,如果没有，就不是元组而是字符串了
print(len(empty))           #>>0
print(len(singleton))       #>>1
x = ('hello',)  #无逗号则x为字符串
print(x)                    #>>('hello',)
```

4、操作

（1）可以对元组进行连接组合：tup3=tup1+tup2,tup+=(10,20)等

（2）元组运算和迭代

```
x = (1,2,3) * 3
print(x) #>>(1, 2, 3, 1, 2, 3, 1, 2, 3)
print( 3 in (1,2,3))  #>>True
for i in (1,2,3):
    print(i,end = "")  #>>123
```

（3）元组赋值

```
x = (1,2,3)
b = x
print(b is x) # true  is 表示两个操作数是否指向同一个东西，即是否是同
   一个对象
x += (100,)   # 等价于 x = x + (100,)新建了一个元组
print (x)     # (1, 2, 3, 100)
print (b)     # (1, 2, 3)
```

（4）元组比大小

①两个元组比大小，就是逐个元素比大小，直到分出胜负

②如果有两个对应元素不可比大小，则出runtime error

（5）元组的排序：元组不能修改，因此无sort函数,可以用sorted得到新的排序后的列表

```
def f(x):
    return (-x[2],x[1],x[0])
students = (('John', 'A', 15),  ('Mike', 'C', 19),
                  ('Wang', 'B', 12), ('Mike', 'B', 12),
               ('Mike', 'C', 12),('Mike', 'C', 18),
               ('Bom', 'D', 10))    #students是元组
print(sorted(students,key = f))     #sorted的结果是列表
```

列表

1、基本操作

（1）列表的增删和修改：列表可以增删元素，列表元素可以修改且可以是任何类型，append用于添加单个元素，del用于删除元素,如list.append（200），del list[2]/list.pop(2);也可以添加多个元素，如list+=[100.110]

（2）列表相加

①列表相加可以得到新的列表（构造出c后c与a,b便无关联）

②对列表来说，a+=b和a=a+b不同

```
b = a = [1,2]
a += [3]      #b和a指向相同地方，在a末尾添加元素，b也受影响
print(a,b)    #>>[1, 2, 3] [1, 2, 3]
a = a + [4,5] #对a重新赋值，不会影响到b
print(a)      #>>[1, 2, 3, 4, 5]
print(b)      #>>[1, 2, 3]
```

（3）列表乘法

（4）列表切片：返回新的列表，用法和字符串切片相同

（5）列表比大小

①两个列表比大小，就是逐个元素比大小，直到分出胜负。

②如果有两个对应元素不可比大小，则出 runtime error。

（6）列表的遍历

```
lst = [1,2,3,4]
for x in lst:
    print(x,end = " ")
    x = 100  #不会修改列表的元素
print(lst)  #>>[1, 2, 3, 4]
for i in range(len(lst)):
    lst[i] = 100
print(lst)   #>>[100, 100, 100, 100]
```

2、列表的排序

（1）朴素排序算法：选择排序

```
def SelectionSort(a): #选择排序
    #将列表a从小到大排序
    n = len(a)
    for i in range(n-1):
        #每次从a[i]及其右边的元素里选出最小的，放在a[i]这个位置
        for j in range(i+1,n): #依次考察a[i]右边元素
            if a[j] < a[i]:
                a[i],a[j] = a[j],a[i]
lst = [1,12,4,56,6,2]
SelectionSort(lst)
print(lst) #>>[1, 2, 4, 6, 12, 56]
```

选择排序：时间复杂度$O(n^2)$，即对有n个元素的列表(数组)，需要做n^2次比较
冒泡排序，插入排序时间复杂度都是$O(n^2)$

好的排序算法，比如归并排序，快速排序，复杂度是O(n*log(n))
python自带的排序功能，复杂度是O(n*log(n))
可以认为排序这件事，复杂度就是O(n*log(n))

（2）用排序函数对简单列表排序

①用缺省的比较规则排序：a.sort()可以对列表a从小到大排序，sorted（a）返回a经过从小到大排序后的新列表，a不变；a.sort(reverse=True)可以对列表a从大到小排序，sorted类似

②自定义比较规则排序

```
➢ 自定义关键字函数 key
def myKey(x):  #关键字函数
    return x % 10
a = [25,7,16,33,4,1,2]
a.sort(key = myKey)
# key是函数，sort按对每个元素调用该函数的返回值从小到大排序
# [1, 2, 33, 4, 25, 16, 7]  按个位数排序
sorted("This is a test string from Andrew".split(),
       key=str.lower)
# ['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
不区分大小写排序
```

（3）复杂列表的自定义排序

①用不同关键字排序

```
students = [
          ('John', 'A', 15), # 姓名，成绩，年龄
          ('Mike', 'B', 12),
          ('Mike', 'C', 18),
          ('Bom', 'D', 10)]
students.sort(key = lambda x: x[2] ) # 按年龄排序
#[('Bom', 'D', 10), ('Mike', 'B', 12), ('John', 'A', 15), ('Mike', 'C', 18)]
students.sort(key = lambda x: x[0] ) # 按姓名排序
# [('Bom', 'D', 10), ('John', 'A', 15), ('Mike', 'B', 12), ('Mike', 'C', 18)]
```

*lambda表达式：lambda x:x[2]表示一个函数，参数是x，返回值是x[2]

②多级排序

```
def f(x):
    return (-x[2],x[1],x[0])
students = [('John', 'A', 15),  ('Mike', 'C', 19),
          ('Wang', 'B', 12), ('Mike', 'B', 12),
             ('Mike', 'C', 12),
             ('Mike', 'C', 18),
             ('Bom', 'D', 10)]
students.sort(key = f ) #先按年龄从高到低，再按成绩从高到低，再按姓名字典序
print(students)
#>>[('Mike', 'C', 18), ('John', 'A', 15), ('Mike', 'B', 12), ('Wang',
   'B', 12), ('Mike', 'C', 12), ('Bom', 'D', 10)]
```

3、列表相关函数

（1）append (x) 添加元素x到尾部

（2）count(x) 计算列表中包含多少个x

（3）extend (x) 添加列表x中的元素到尾部

（4）insert(i,x) 将元素x插入到下标i处

（5）remove(x) 删除元素x，如果x不存在，则引发异常

（6）reverse() 颠倒整个列表

（7）index(x) 查找元素x，找到则返回第一次出现的下标，找不到则引发异常

（8）index(x,s) 从下标s开始查找x

（9）x.join(s)使用x作为分隔符连接列表中的字符串

（10）zip(a,b)接收多个可迭代对象（如列表），然后然后把每个可迭代对象中的第i个元素组合在一起，形成一个新的迭代器，类型为元组

①提取迭代器数据

a.next方法

```
a=[1,2,3]
b=[5,6,7]
c=zip(a,b)
print(next(c))
print(next(c))
print(next(c))
```

```
(1, 5)
(2, 6)
(3, 7)
```

b.list方法

```
a=[1,2,3]
b=[5,6,7]
c=zip(a,b)
print(list(c))
```

```
[(1, 5), (2, 6), (3, 7)]
```

c.遍历方法

```
a=[1,2,3]
b=[5,6,7]
c=zip(a,b)
for i in c:
    print(i)
```

```
(1, 5)
(2, 6)
(3, 7)
```

*三个及以上元素也可以使用zip函数组合在一起。如果三个列表的长度不等，则zip所返回的迭代器的长度将有长度最短的那个列表决定

②zip()与*运算符相结合可以用来拆解一个列表，返回的数据直接就是解包后的元组

```
x = [1, 2, 3]
y = [4, 5, 6]
zipped = zip(x, y)
print(list(zipped))
print(zip(x, y)) # 迭代器对象，
print(*zip(x, y))# 组合好的多个元组数据
```

```
[(1, 4), (2, 5), (3, 6)]
<zip object at 0x7fd5c01719c0>
(1, 4) (2, 5) (3, 6)
```

（11）enumerate函数：返回一个枚举对象，其中每个元素都是一个包含索引和对应元素的元组

```
enumerate(iterable, start=0)
```

```
fruits = ['apple', 'banana', 'orange']

for index, fruit in enumerate(fruits):
    print(index, fruit)
```

```
0 apple
1 banana
2 orange
```

4、列表映射和过滤

（1）列表映射

①map(function, sequence)，可用于将一个序列(列表、元组、集合...)映射到另一个序列

②返回一个延时值求对象，可以转换成list,tuple,set....

（2）列表过滤

①filter(function,sequence),抽取序列中令function(x)为True的元素x

②返回一个延时求值对象

```
def f(x):
    return x % 2 == 0
lst = tuple(filter(f,[1,2,3,4,5]))       #抽取出偶数
print(lst)    #>>(2, 4)
```

字典

1、基本概念

（1）字典dt = {key1 : value1, key2 : value2..... }的每个元素是由“键：值”两部分组成，可以根据“键”快速查找

（2）字典元素的值是可赋值的，因此也是指针

（3）字典的键不可重复，指字典的键的内容不能一样

（4）键必须是不可变的数据类型，比如字符串、整数、小数、元组； 列表、集合、字典等可变的数据类型，不可作为字典元素的键。

（5）一个元组如果包含可变数据类型，也不能作为集合的元素

2、操作

（1）赋值/添加：dt[key]=x

（2）删除：del dt[key]

（3）空字典：scope={}

（4）判断是否有该元素键：print(“key” in scope)

```
scope['k'] = scope.get('k',0) + 1
#get(key,v) ：如果键key存在，则返回键为key的元素的值，否则返回v
print(scope['k'])   #>>1
scope['k'] = scope.get('k',0) + 1
print(scope['k'])   #>>2
```

（5）构造（字典从后往前叙述）：dt={key:item,...}

```
items = [('name','Gumby'),('age',42)]
d = dict(items)
print(d)          #>>{'name': 'Gumby', 'age': 42}
d = dict(name='Gumby',age=42,height=1.76)
print(d)      #>>{'height': 1.76, 'name': 'Gumby', 'age': 42}
```

```
>>> dict([('Runoob', 1), ('Google', 2), ('Taobao', 3)])
{'Runoob': 1, 'Google': 2, 'Taobao': 3}
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
>>> dict(Runoob=1, Google=2, Taobao=3)
{'Runoob': 1, 'Google': 2, 'Taobao': 3}
```

*{x: x**2 for x in (2, 4, 6)} 该代码使用的是字典推导式

（6）深拷贝

（7）遍历

```
➢ items
x = {'username':'admin', 'machines':['foo', 'bar', 'baz'],
   'Age':15}
for i in x.items():
     print(i[0])
     print(i[1])
```

（8）排序

①按照字典的键

```
my_dict = {"c": 3, "a": 1, "b": 2}

sorted_dict = dict(sorted(my_dict.items(), key=lambda x: x[0]))

print(sorted_dict)
```

②按照字典的值

```
my_dict = {"c": 3, "a": 1, "b": 2}

sorted_dict = dict(sorted(my_dict.items(), key=lambda x: x[1]))

print(sorted_dict)
```

3、相关函数

（1）clear() 清空字典

（2）keys() 取字典的键的序列

（3）items() 取字典的元素的序列，可用于遍历字典

```
print (tinydict.keys())   # 输出所有键
print (tinydict.values()) # 输出所有值
```

（4）values() 取字典的值序列

（5）pop(x) 删除键为x的元素，如果不存在，产生异常

*上述"序列"，不是 list,tuple或set

（6）copy() 浅拷贝

（7）get(k,v) 如果有元素键为k，则返回该元素的值，如果没有，则返回v

集合

1、概念：一种无序、可变的数据类型，用于存储唯一的元素，使用大括号{}表示，元素之间用逗号分隔，集合中的元素不会重复

2、基本操作

（1）集合运算

```
print(a - b)     # a 和 b 的差集

print(a | b)     # a 和 b 的并集

print(a & b)     # a 和 b 的交集

print(a ^ b)     # a 和 b 中不同时存在的元素
```

x in a x是否在集合a中

a == b a是否元素和b一样

a != b a是否元素和b不一样

a <= b a是否是b的子集(a有的元素b都有)

a < b a是否是b的真子集(a有的元素b都有且b还包含a中没有的元素)

a >= b b是否是a的子集

a > b b是否是a的真子集

（2）可以使用 set() 函数创建集合，创建一个空集合必须用set()而不是{ }，因为{ }是用来创建一个空字典。

```
parame = {value01,value02,...}
或者
set(value)
```

*创建数集可以用set(range(a,b))

*可用于统计不同字符个数，清除重复项

*整数、小数、复数、字符串、元组都可以作为集合的元素。但是列表、字典和集合等可变的数据类型不可作为集合的元素。 一个元组如果包含可变数据类型，也不能作为集合的元素。

3、常用函数

（1）add(x)：添加元素x，如果x已经存在，则不添加

（2）clear()：清空集合

（3）copy()：返回自身的浅拷贝

（4）remove(x)：删除元素x。如果不存在元素x，则引发异常

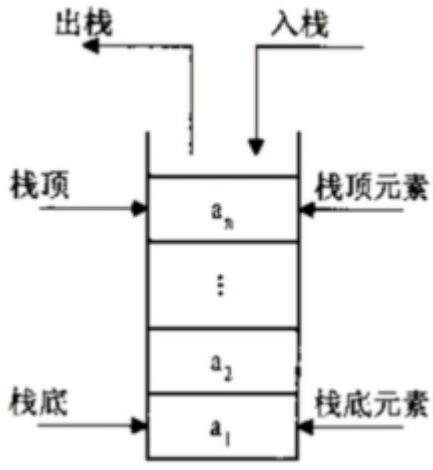
（5）update(x)：将序列x中的元素加入到集合

数据结构

栈（stack)

1、特点：一种容器，可存入数据元素、访问元素、删除元素，它的特点在于只能允许在容器的一端（称为栈顶端指标，英语：top）进行加入数据（英语：push）和输出数据（英语：pop）的运算。没有了位置概念，保证任何时候可以访问、删除的元素都是此前最后存入的那个元素，确定了一种默认的访问顺序

2、原理：由于栈数据结构只允许在一端进行操作，因而按照后进先出（LIFO, Last In First Out）的原理运作



3、结构实现：可以用顺序表（适用于对元素的访问频繁，而插入和删除操作较少的场景）或链表（适用于频繁插入和删除操作）

4、操作

（1）Stack() 创建一个新的空栈

（2）push(item) 添加一个新的元素item到栈顶

（3）pop() 弹出栈顶元素

（4）peek() 返回栈顶元素

（5）is_empty() 判断栈是否为空

（6）size() 返回栈的元素个数

```python
class Stack(object):
    """栈"""
    def __init__(self):
        self.items = []

    def is_empty(self):
        """判断是否为空"""
        return self.items == []

    def push(self, item):
        """加入元素"""
        self.items.append(item)

    def pop(self):
        """弹出元素"""
        return self.items.pop()

    def peek(self):
        """返回栈顶元素"""
        return self.items[len(self.items)-1]

    def size(self):
        """返回栈的大小"""
        return len(self.items)
```

```python
if __name__ == "__main__":
    stack = Stack()
    stack.push("hello")
    stack.push("world")
    stack.push("itcast")
    print stack.size()
    print stack.peek()
    print stack.pop()
    print stack.pop()
    print stack.pop()
```

队列

1、队列（queue）

（1）特点：只允许在一端进行插入操作，而在另一端进行删除操作的线性表，先进先出（FIFO）

（2）实现：用顺序表或者链表实现

（3）操作

①Queue() 创建一个空的队列

②enqueue(item) 往队列中添加一个item元素

③dequeue() 从队列头部删除一个元素

④is_empty() 判断一个队列是否为空

⑤size() 返回队列的大小

```
class Queue(object):
    """队列"""
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, item):
        """进队列"""
        self.items.insert(0,item)

    def dequeue(self):
        """出队列"""
        return self.items.pop()

    def size(self):
        """返回大小"""
        return len(self.items)
```

```
if __name__ == "__main__":
    q = Queue()
    q.enqueue("hello")
    q.enqueue("world")
    q.enqueue("itcast")
    print q.size()
    print q.dequeue()
    print q.dequeue()
    print q.dequeue()
```

2、双端队列（deque）

（1）特点：一种具有队列和栈的性质的数据结构，双端队列中的元素可以从两端弹出，其限定插入和删除操作在表的两端进行。双端队列可以在队列任意一端入队和出队。

*在deque两端插入和删除元素的时间复杂度近似O(1)，list则为O(n)。

（2）实现：from collections import deque

（3）操作

①创建：d=deque()

②常规函数：append(), appendleft(), extend(), extendleft(), pop(), popleft(), count(), insert(index,object)

③其他函数

a. rotate(n)：从右侧反转n步，若n为负数，则从左侧反转

```
from collections import deque

st = "abbcd"
dst = deque(st)
dst.rotate(1)
print(dst)
#结果:
#deque(['d', 'a', 'b', 'b', 'c'])
```

b. clear()：将deque中的元素全部删除

c. remove()：移除第一次出现的元素，如果没有找到，报出ValueError

d. maxlen：只读的属性，deque限定的最大长度，如果无，就返回None。当限制长度的deque增加超过限制数的项时, 另一边的项会自动删除。

```python
from collections import deque

dst = deque(maxlen=2)
dst.append(1)
dst.append(2)
print(dst)
dst.append(3)
print(dst)
print(dst.maxlen)
#结果:
#deque([1, 2], maxlen=2)
#deque([2, 3], maxlen=2)
#2
```

堆（heap）

1、概念：一种特殊的完全二叉树数据结构

2、类型

（1）大顶堆：父节点的值大于或等于其子节点的值

（2）小顶堆：父节点的值小于或等于其子节点的值

3、特点

（1）堆是一种完全二叉树，意味着当除最后一层外的所有层都被填满时，堆是满的，并且最后一层的节点都依次从左到右排列。

（2）在大顶堆中，每个节点的值都大于或等于其子节点的值；而在小顶堆中，每个节点的值都小于或等于其子节点的值

（3）堆中的任何节点都不保证是其子树中节点的最大或最小值

4、实现方式：heapq库

5、常见操作

（1）插入：将一个元素插入到堆中，heappush(heap,item)

（2）删除：删除堆中的根节点，heappop(heap)

（3）构建堆：将输入的数据集合转换为堆的过程

①heappush(heap, num)，先创建一个空堆，然后将数据一个一个地添加到堆中。每添加一个数据后，heap都满足小顶堆的特性。

②heapify(array)，直接将数据列表调整成一个小顶堆

*两种方法实现的结果会有差异，但都满足小顶堆的特性，不影响堆的使用

（4）堆排序

```python
array = [10, 17, 50, 7, 30, 24, 27, 45, 15, 5, 36, 21]
heap = []
for num in array:
    heapq.heappush(heap, num)
print(heap[0])
# print(heapq.heappop(heap))
heap_sort = [heapq.heappop(heap) for _ in range(len(heap))]
print("heap sort result: ", heap_sort)
```

（5）获取堆中的最小值或最大值

```
array = [10, 17, 50, 7, 30, 24, 27, 45, 15, 5, 36, 21]
heapq.heapify(array)
print(heapq.nlargest(2, array))
print(heapq.nsmallest(3, array))
```

```
[50, 45]
[5, 7, 10]
```

（6）合并两个有序列表

```
array_a = [10, 7, 15, 8]
array_b = [17, 3, 8, 20, 13]
array_merge = heapq.merge(sorted(array_a), sorted(array_b))
print("merge result:", list(array_merge))
```

```
merge result: [3, 7, 8, 8, 10, 13, 15, 17, 20]
```

（7）替换数据

```
array_c = [10, 7, 15, 8]
heapq.heapify(array_c)
print("before:", array_c)
# 先push再pop
item = heapq.heappushpop(array_c, 5)
print("after: ", array_c)
print(item)

array_d = [10, 7, 15, 8]
heapq.heapify(array_d)
print("before:", array_d)
# 先pop再push
item = heapq.heapreplace(array_d, 5)
print("after: ", array_d)
print(item)
```

```
before: [7, 8, 15, 10]
after:  [7, 8, 15, 10]
5
before: [7, 8, 15, 10]
after:  [5, 8, 15, 10]
7
```

常用标准库

1、math库：sqrt, ceil, floor，math.inf(正无穷)，math.log2，math.pow(x,y)(返回x的y次幂)，math.log(x,base),math.log10，math.gcd(a,b)（返回a与b的最大公约数），math.comb(获取从n个项目中选择不重复且无顺序的k个项目的方法数量)

```
import math
a, b, k, n, m = map(int, input().split());
print((pow(a, n, 10007) * pow(b, m, 10007) * math.comb(k, m)) % 10007)
```

2、functools库：lru_cache

3、collections库：counter，defaultdict, deque

Counter

是dict的子类，同样具有dict相同的操作

以字典的形式返回序列中各个字符出现的次数，值为key，次数为value

```
>>> from collections import Counter
>>> c = Counter('abcdeabcdabcaba')
>>> c
>>> Counter({'a': 5, 'b': 4, 'c': 3, 'd': 2, 'e': 1})
>>> c.most_common(2)        # 获取出现次数最多的前两个元素
>>> [('a', 5), ('b', 4)]
>>> sorted(c)               # 列出所有不同的元素并排序
>>> ['a', 'b', 'c', 'd', 'e']
>>> c.elements()
>>> <itertools.chain object at 0x000001821EFF5710>
>>> ''.join(c.elements())      # 列出具有重复的元素
>>> 'aaaaabbbbcccdde'
>>> c.items()
>>> c.get('a')
>>> 5
```

4、heapq库：heapify(array),heappush(list,item),heappop(list)

5、bisect库：bisect_left, bisect_right，insort_left,insort_right

6、calendar库：isleap（year）判断是否为闰年

常用函数

int（x）：

①把字符串转换为整数/小数转成整数（去尾取整）

*x不会变成整数，只是这个表达式的值是整数

②转换十进制：int(“x”，进制)

```
binary_number = '1010'
decimal_number = int(binary_number, 2)
print(decimal_number)  # 输出：10
```

*bin,oct,hex函数分别可以将十进制转换为二进制、八进制、十六进制

float（x）：把字符串x转换成小数

round(要格式化的数，小数位数)：round(3.1415926,5)——3.14159

str（x）：把x转换为字符串

complex(a,b)：创建一个复数

eval（str）：把字符串x看作是一个python表达式，求其值（也可用于转成元组等）

repr(x)：把对象x转换为表达式字符串

ord(x):字符转换为编码

chr(x):编码转换为字符

bin(x)：整数转换为二进制字符串

oct(x):整数转换为八进制字符串

hex(x):整数转换为十六进制字符串

abs(x):求绝对值

tuple(s):序列转换为元组

list(s):序列转换为列表

set(s):序列转换为可变集合

frozenset(s):序列转换为不可变集合

dict(d):创建字典，d 必须是一个 (key, value)元组序列

for index,value in enumerate([a,b,c])

常用模型

判断完全平方数

```
def perfect(x):
    if (int(sqrt(x)))**2==x:
        return True
    return False
```

埃氏筛和欧拉筛

①埃氏筛：从小到大遍历所有的数，如果当前的数是质数（即没有被之前的数筛掉），那么将它的所有倍数标记为非质数；但会出现重复筛掉同一个数的情况，因此速度较慢

```
import math
def f(n):
    ls,x,y=[True]*(n+1),2,int(math.sqrt(n))+1
    ls[0]=ls[1]=False
    while x<y:
        if ls[x]==True:
            for i in range(x*x,n+1,x):
                ls[i]=False
        x+=1
    ls=[i for i in range(2,n+1) if ls[i]==True]
    return ls
print(f(int(input())))
```

②欧拉筛：在埃氏筛法的基础上，让每个合数只被它的最小质因子筛选一次，以达到不重复的目的；但需要数组另外储存质数，内存较大

```
def f(n):
    ls=[True]*(n+1)
    primes=[]
    for i in range(2, n+1):
        if ls[i]:
            primes.append(i)
            for j in primes:
                if i*j>n:
                    break
                ls[i*j]=False
                if i%j==0:
                    break
    return primes
print(f(int(input())))
```

螺旋矩阵

```
n=int(input())
board=[[1]*(n+2)]+[([1]+[0]*n+[1]) for _ in range(n)]+[[1]*(n+2)]
direction=[[0,1],[1,0],[0,-1],[-1,0]]
row,col=1,1
drow,dcol=direction[0]
t=0
for j in range(1,n**2+1):
    board[row][col]=j
    if board[row+drow][col+dcol]:
        t+=1
    drow,dcol=direction[t%4]
    row+=drow
    col+=dcol
for i in range(1,n+1):
    s=board[i][1:n+1]
    print(" ".join(map(str,s)))
```

注意点

1、注意输出格式：是否有前缀“Case n”，是否每组输出后有空行，是否有无解情况输出-1，是否句末有句点，“YES”“NO”的大小写等

2、注意是大于还是大于等于

3、reversed()外加list才能进行列表操作

4、排序！题目中给的样例可能已经排好序了，但自己做的时候需要记得加一步排序。

5、注意特殊情况：如起点与终点重合

6、注意执行操作的先后顺序，如打怪兽先打消耗魔力值少的可以使总数最多

7、for i in items这样的循环中i是一个临时变量，对i所做的更改不会影响到items列表中的元素；因此要修改items列表中的元素，应该用i指代索引，用索引items[i]=a进行赋值修改

8、变量名不要重复

9、矩阵的横轴纵轴看准

10、看清大于/大于等于

11、浮点数与整数（如除法、开方等得出的是小数）

12、缩进是否正确

13、多组数据try,except终止程序

14、边界条件

15、循环嵌套逻辑（双重循环在哪一重循环外判断）

16、多个矩阵看好索引是哪一个矩阵

17、多组输入数据注意是否有变量需要在每个循环中进行更新（特别是定义类型，需更新变量应置于定义内）

18、初始值的定义（特别是dp初值设成0or1等）

19、输入没有空格的话要用list分成列表

20、是否可以先打表再对每组数据直接求值

21、把tmp加入a时，先复制再加入a,即a.append(list(tmp))

22、二维数组索引值是prices[i] [j]

23、dp时是否设置了0点，关系到是j==t为分界还是j+1==t为分界；若没有加零点，索引都是实际值-1，差值则不必考虑是否加零点