#作业一

##变量及赋值

###一元一次方程

```
x=5.0
monadic_equation=2*x+1
print("monadic_equation=",monadic_equation)
```

###list

```
lst_n=list(range(5,20,3))
print(lst)
print("The list length={}".format(len(lst_n)))
print("Maximum value={}".format(max(lst_n)))
print("Minimum value={}".format(min(lst_n)))
```

#####分片(List Slices)

```
lst_s=list(map(chr,range(100,110)))
print(lst_s)
print("_"*50)
print("[3:6]->{}".format(lst_s[3:6]))
print("[-3:-1]->{}".format(lst_s[-3:-1]))
print("[-3:]->{}".format(lst_s[-3:]))
print("[:3]->{}".format(lst_s[:3]))
print("[:]->{}".format(lst_s[:]))
```

###列表索引示例

```
['d', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm']
__________________________________________________
[3:6]->['g', 'h', 'i']
[-3:-1]->['k', 'l']
[-3:]->['k', 'l', 'm']
[:3]->['d', 'e', 'f']
[:]->['d', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm']
```

###用help()方法查看说明

```
Help on class map in module builtins:
```

```
class map(object)
 |  map(func, *iterables) --> map object
 |
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
```

#####带有步长(step length)参数的分片

```
print(lst_s)
print("_"*50)
print("[0:10:2]->{}".format(lst_s[0:10:2]))
print("[::3]->{}".format(lst_s[::3]))
print("[9:3:-2]->{}".format(lst_s[9:3:-2]))
print("[20:3:-2]->{}".format(lst_s[20:3:-2]))
print("[7::-2]->{}".format(lst_s[7::-2]))
print("[:3:-2]->{}".format(lst_s[:3:-2]))
```

#####元素赋值+分片赋值+删除元素+列表相加+列表的乘法

```
print(lst_s)
print("_"*50)
lst_s[5]=99 #元素赋值
print("lst_s[5]=99->{}".format(lst_s))
lst_none=lst_s+[None]*6
print("lst_s+[None]*6->{}".format(lst_none))
lst_none[13]=2015
print("lst_none[13]=2015->{}".format(lst_none))
lst_none[-6:-3]=list(range(100,104,2)) #分片赋值
print("lst_none[-6:-3]=list(range(100,104,2))->{}".format(lst_none))
```

```
lst_none[1:1]=[0,0,0,12]
print("lst_none[1:1]=[0,0,0,12]->{}".format(lst_none))
del lst_none[-2:] #删除元素
print("del lst_none[-2:]->{}".format(lst_none))
```

#####列表的方法

```
lst_s_2=list(map(chr,range(100,105)))
print(lst_s_2)
print("_"*50)
lst_s_2.append(99)
print("lst_s_2.append(99)->{}".format(lst_s_2))
lst_s_2.append(list(range(50,80,5)))
print("lst_s_2.append(list(range(50,80,5)))->{}".format(lst_s_2))
lst_spechars=['*',')','*']
lst_s_2.extend(lst_spechars)
print("lst_s_2.extend(lst_spechars)->{}".format(lst_s_2))
print("lst_s_2.count('*')={}".format(lst_s_2.count('*')))
print("lst_s_2.index('e')={}".format(lst_s_2.index('e')))
lst_s_2.insert(2,[1000,1200,1500])
print("lst_s_2.insert(2,[1000,1200,1500])->{}".format(lst_s_2))
print("lst_s_2.pop(7)_popup->{}".format(lst_s_2.pop(7)))
print("lst_s_2.pop(7)_retention->{}".format(lst_s_2))
lst_s_2.remove('*')
print("lst_s_2.remove('*')_retention->{}".format(lst_s_2))
list_n_2=[2,42,6,95,4,3]
list_n_2.sort()
print("list_n_2.sort()->{}".format(list_n_2))
```

###tuple

```
tuple_1=2,3,5,
print("tuple_1=2,3,5,->{}".format(tuple_1))
print("3*(20*3,)->{}".format(3*(20*3,)))
print("tuple((5,8,9))->{}".format(tuple((5,8,9)))) #用()
print("tuple([5,8,9])->{}".format(tuple([5,8,9]))) #用[]
```

###dict
#####使用dict()函数建立字典

```
import random
items=[(0,[i for i in range(5)]),(1,[random.sample(list(range(100,200,1)),3)]),
(2,'python')] #[i for i in range(5)] 为列表推导式 list comprehension/derivation
print("items->{}".format(items))
dic=dict(items)
print("dic=dict(items)->{}".format(dic))
print("dic[1]->{}".format(dic[1]))
```

#####字典的基本操作

```
print("len(dic)->{}".format(len(dic)))
dic[3]=(random.random(),random.uniform(200,300))
print("dic[3]=(random.random(),random.uniform(200,300))->{}".format(dic))
del dic[1]
print("del dic[1]->{}".format(dic))
print("3 in dic->{}".format(3 in dic))
print("5 in dic->{}".format(5 in dic))
print("dic.keys()->{}".format(dic.keys())) #应该放在字典的方法一节里
print("dic.values()->{}".format(dic.values()))
print("dic.items()->{}".format(dic.items()))
print("_"*50)
print("list(dic.keys())->{}".format(list(dic.keys())))
```

```
for k,v in enumerate(dic.items())
print(k,v)
```

#####字典的基本操作

```
lst_A=list(range(6,20,3))
lst_B=list(range(100,150,15))
print("lst_A={},lst_B={}".format(lst_A,lst_B))
dic_2={0:lst_A,1:lst_B}
print("dic_2={}".format(dic_2))
print("_"*50)
dic_assignment=dic_2
print("dic_assignment={}".format(dic_assignment))
dic_2.clear()
print("dic_2.clear()->{}".format(dic_2))
print("dic_assignment={}".format(dic_assignment))
dic_2[5]=list(range(1,9,2))
print("dic_2[5]=list(range(1,9,2))->{}".format(dic_2))
dic_copy=dic_2.copy()
print("dic_copy=dic_2.copy()->{}".format(dic_copy))
dic_2[8]=[5,7]
print("dic_2[8]=[5,7]->{}".format(dic_2))
print("dic_copy={}".format(dic_copy))
dic_copy[5].remove(5)
print("dic_copy[5].remove(5)->{}".format(dic_copy))
dic_copy.setdefault(6,[77,99]) #返回指定键的值，如果不存在该键，则字典增加新的键/值对
print("dic_copy.setdefault(6,[77,99])->{}".format(dic_copy))
dic_2.pop(5) #移除指定键/值，并返回该值
print("dic_2.pop(5)->{}".format(dic_2))
dic_update={8:[5,7,6,3,2],9:[3,2,33,55,66]}
print("dic_update={}".format(dic_update))
dic_2.update(dic_update) #更新字典
```

```
print("dic_2.update(dic_update->{}".format(dic_2))
print("dic_2.get(9)->{}".format(dic_2.get(9)))
dic_2.popitem() #随即弹出一对键/值，并在该字典中移除
print("dic_2.popitem()->{}".format(dic_2))

dic_3={}.fromkeys([0,1,2,3,4,'A']) #给定键，建立值为空的字典
print("dic_3={}"+".fromkeys([0,1,2,3,4,'A'])->{}".format(dic_3)) #找下escape
characters 脱字符
```

###string-基础
#####常用方法作

```
lst_s_3=list("Hello Python!")
print("lst_s_3=list(\"Hello Python!\")->{}".format(lst_s_3)) #"\" escape character
print("\"Hellow\"+\" Python!\"->{}".format("Hellow"+" Python!"))
print("\"+\".join(str(123456))->{}".format("+".join(str(123456))))
print("len(\"Hellow Python!\")->{}".format(len("Hellow Python!")))
coordinates="120.132007,30.300508,9.7"
print("coordinates.split(\",\")->{}".format(coordinates.split(",")))
print("eval(coordinates)->{}".format(eval(coordinates))) #通常用eval()方法将字符
串，转换为对应数值形式;
print("\"Hello Python!\".lower()->{}".format("Hello Python!".lower()))
print("\"Hello Python!\".upper()->{}".format("Hello Python!".upper()))
print("\"Hello Python!\"[6:]->{}".format("Hello Python!"[6:]))
print("\"    Hello Python!    \".strip()->{}".format("    Hello Python!
".strip()))
print("\"Hello Python!\".replace(\"Python\",\"Grasshopper\")->{}".format("Hello
Python!".replace("Python","Grasshopper")))
hello_lst=list("Hello Python!")
hello_lst.sort()
print("hello_lst.sort()>{}".format(hello_lst))
print("\"Hello Python!\".find(\"Py\")->{}".format("Hello Python!".find("Py")))
```

#####字符串格式化

```
format_str="Hello,%s and %s!"
values=("Python","Grasshopper")
new_str=format_str % values
print("new_str=format_str % values->{}".format(new_str))

format_str_2="Pi with three decimals:%.3f,and enter a value with percent sign:%.2f
%%" #如果字符串里包含实际的%，则通过%%即两个百分号进行转义

from math import pi
new_str_2=format_str_2 % (pi,3.1415926)
print("new_str_2=format_str_2 % (pi,3.1415926)->{}".format(new_str_2))
```

```
format_str_3="%10f,%10.2f,%.2f,%.5s,%.*s,%d,%x,%f"
new_str_3=format_str_3%(pi,pi,pi,"Hello Python!",5,"Hello Python!",52,52,pi)
print("{}".format(new_str_3))
```

#####Template()函数-模板字符串+$+st.substitute()

```
from string import Template
s_template_1=Template("$x,glorious,$x!")
s_1=s_template_1.substitute(x="Python")
print("s_1=s_template_1.substitute(x=\"Python\")->{}".format(s_1))
s_template_2=Template("${x}thon is amazing!")
s_2=s_template_2.substitute(x="py")
print("s_2=s_template_2.substitute(x=\"py\")->{}".format(s_2))
s_template_3=Template("$x and $y are both amazing!")
substitute_dict=dict([('x','Python'),('y','Grasshopper')])
print("substitute_dict={}".format(substitute_dict))
s_3=s_template_3.substitute(substitute_dict)
print("s_3=s_template_3.substitute(substitute_dict)->{}".format(s_3))
```

###string-re(regular expression)正则表达式

```
import re
pattern_1='Python'
text="Hello Python!"
print("re.findall(pattern_1,text)->{}".format(re.findall(pattern_1,text)))
pattern_2='python'
print("re.findall(pattern_2,text)->{}".format(re.findall(pattern_2,text)))
```

#####(.)点号

```
print("re.findall('.ython','Hello Python!')->{}".format(re.findall('.ython','Hello
Python!')))
print("re.findall('.ython','Hello gython!')->{}".format(re.findall('.ython','Hello
gython!')))
print("re.findall('.ython','Hello gPython!')->
{}".format(re.findall('.ython','Hello gPython!')))
print("re.findall('.ython','Hello Pthon!')->{}".format(re.findall('.ython','Hello
Pthon!')))
```

#####* + ？ {m,n}与r'string'

```
print("re.findall(r'w?
cadesign\.cn,w+\.cadesign\.cn','cadesign.cn,www.cadesign.cn')->
{}".format(re.findall(r'w?
cadesign\.cn,w+\.cadesign\.cn','cadesign.cn,www.cadesign.cn')))
print("re.findall(r'w{2}"+"\.cadesign\.cn','www.cadesign.cn')->
{}".format(re.findall(r'w{2}\.cadesign\.cn','www.cadesign.cn')))
print("re.findall(r'w{1,3}"+"\.cadesign\.cn','www.cadesign.cn')->
{}".format(re.findall(r'w{1,3}\.cadesign\.cn','www.cadesign.cn')))
```

#####* [] (^)

```
print("re.findall('[Py]*thon!','Hello Python!')->
{}".format(re.findall('[Py]*thon!','Hello Python!')))
print("re.findall('[Py]*thon!','Hello Pthon!')->
{}".format(re.findall('[Py]*thon!','Hello Pthon!')))
print("re.findall('[Py]*thon!','Hello ython!')->
{}".format(re.findall('[Py]*thon!','Hello ython!')))
print("re.findall('[Py]*thon!','Hello ython!')->
{}".format(re.findall('[Py]*thon!','Hello thon!')))
```

#####* [] (^)

```
print("re.findall('python|grasshopper','python')->
{}".format(re.findall('python|grasshopper','python')))
print("re.findall('python|grasshopper','grasshopper')->
{}".format(re.findall('python|grasshopper','grasshopper')))
print("re.findall('python|grasshopper','grasshopper and python')->
{}".format(re.findall('python|grasshopper','grasshopper and python')))
```

#####\d \D

```
print("re.findall('\d','number=10')->{}".format(re.findall('\d','number=10')))
print("re.findall('\D','number=10')->{}".format(re.findall('\D','number=10')))
print("re.findall('[^0-9]','number=10')->{}".format(re.findall('[^0-
9]','number=10')))
```

#####re模块的方法

```
print("re.findall('[a-z]','python')->{}".format(re.findall('[a-z]','python-3.0')))
print("re.search('[a-z]+','python')->{}".format(re.search('[a-z]+','python')))
if re.search('[a-z]+','python'):
    print("re.search('[a-z]+','python')->found it!")
print("re.split(',','Hello,,,,,,Python!')->
{}".format(re.split(',','Hello,,,,,,Python!')))
print("re.sub('Python','Grasshopper','Hello Python!')->
{}".format(re.sub('Python','Grasshopper','Hello Python!')))

pattern_compile=re.compile('Python')
```

```
print("pattern_compile.findall('Hello,,,,,,Python!')->
{}".format(pattern_compile.findall('Hello,,,,,,Python!')))

if re.match('p','python'):
    print("re.match('p','python')->found it!")
```

```
print("\'python\'.find(\'python\')->{}".format('python'.find('python')))
print("\'python\'.find(\'thon\')->{}".format('python'.find('thon')))
print("\'python\'.find(\'a\')->{}".format('python'.find('a')))
print("\'p\' in \'python\'->{}".format('p' in 'python'))
```

```
print("\'Hello,,,,,,Python!\'.split(',')->{}".format(
'Hello,,,,,,Python!'.split(',')))
print("\'Hello Python!\'.replace(\'Python\',\'Grasshopper\')->{}".format( 'Hello
Python!'.replace('Python','Grasshopper')))
```

#####匹配对象和组

```
match_1=re.match(r'www\.(.*)\..{3}','www.python.org')
print("match_1.gourp(1)->{}".format(match_1.group(1)))
print("match_1.start(1)->{}".format(match_1.start(1)))
print("match_1.end(1)->{}".format(match_1.end(1)))
print("match_1.span(1)->{}".format(match_1.span(1)))
match_2=re.match(r'www\.(.*)\.(.{3})','www.python.org')
print("match_2.group(1)->{}".format(match_2.group(1)))
print("match_2.group(2)->{}".format(match_2.group(2)))
```

###循环语句
#####for循环

```
lst_1=list(range(29,37,2))
print("lst_1={}".format(lst_1))
print("_"*50)
print("for i in lst_1:")
for i in lst_1:
    print(i)
print("for i in range(len(lst_1)):")
for i in range(len(lst_1)):
    print("idx={},val={}".format(i,lst_1[i]))
print("+"*50)
dic_4=dict(a=2,b=3,c=6,d=0)
print("dic_4={}".format(dic_4))
print("_"*50)
print("for k in dic_4:")
for k in dic_4:
    print(k)
print("for k,v in dic_4.items():")
for k,v in dic_4.items():
    print("key={},val={}".format(k,v))
```

#####while循环

```
x=1
while x<=100:
    print("x={}".format(x))
    x+=10 #增量赋值; x*=2
```

```
x=1
while x<=100:
    print("x={}".format(x))
    x+=10
    if x>=50:break

import sys
print("sys.maxsize={}".format(sys.maxsize))
for i in range(sys.maxsize):
    print("i={}".format(i))  #?
    i+=10
    if i>=30:break
```

#####While True/break语句

```
topography_fp='./data/elevation.txt' #在处理GIS时，更多的是使用GeoPandas等处理地理信息的库;另，大数据的存储读取通常用Numpy,(Geo)Pandas提供的方法，以及HDF5(binary data format)。
f=open(topography_fp,'r')
elevation_dat=[]
while True:
    line=f.readline()
    elevation_dat.append(line)
    if not line:break
f.close()
print("elevation_dat[:10]={}".format(elevation_dat[:10]))
```

```
import pandas as pd
elevation_df=pd.read_csv(topography_fp, delimiter = ",",header=None)
print(elevation_df)
```

#####并行迭代

```
lst_a=[0,1,2,3]
lst_b=['point_a','point_b','point_c','point_d']
zip_lst=zip(lst_a,lst_b) #The zip() function takes iterables (can be zero or
more), aggregates them in a tuple, and returns it.
print("zip_lst=zip(lst_a,lst_b)->{}".format(zip_lst))
print("dict(zip_lst)->{}".format(dict(zip_lst)))

zip_lst=zip(lst_a,lst_b) #迭代对象临时存储，读取完后为空
for a,b in zip_lst:
    print(a,b)

zip_lst=zip(lst_a,lst_b)
a,b=zip(*zip_lst)
print("a={},b={}".format(a,b))
```

#####编号迭代

```
lst_c=['point_a','point_b','point_c','point_d']
dic_4={}
for idx,value in enumerate(lst_c):
    dic_4[idx]=value
print("dic_4={}".format(dic_4))
print("dict((i,v) for i,v in enumerate(lst_c))->{}".format(dict((i,v) for i,v in
enumerate(lst_c)))) #list comprehension
print("_"*50)
for i,(a,b) in enumerate(zip(lst_a,lst_b)):
    print('%d:%s,%s'%(i,a,b))
```

#####list comprehension(列表推导式)

```
[expr(val) for val in collection if condition]  /newlist=[expression for item in
iterable if condition == True]  | [expression if condition else expression for
item in iterable <if condition>]
```

```
print("[x*x for x in range(3,37,5) if x%2==0]->{}".format([x*x for x in
range(3,37,5) if x%2==0]))
print("[(x,y) for x in range(3)for y in range(2)]->{}".format([(x,y) for x in
range(3)for y in range(2)]))
print("[(x,y) for x,y in zip(range(3),range(2))]->{}".format([(x,y) for x,y in
zip(range(3),range(2))]))
nested_list=[[a for a in range(1,10,3)],2,3,[b for b in range(60,100,7)],[[c for c
in range(1000,2000,120)],3,9]]
print("[[a for a in range(1,10,3)],2,3,[b for b in range(60,100,7)],[[c for c in
range(1000,2000,120)],3,9]]->{}".format(nested_list)) #嵌套列表推导式

flatten_lst=lambda lst: [m for n_lst in lst for m in flatten_lst(n_lst)] if
```

```
type(lst) is list else [lst] #展平列表常用，可以放置到单独的.py文件中调用。lambda函数
后文阐述
print("flatten_lst(nested_list)->{}".format(flatten_lst(nested_list)))
```

###条件语句
#####条件语句模式

```
if expression:
    statements
elif expression:
    statements
elif expression:
    pass
...
else:
    statements
```

```
x=10
if x<0:
    print('It is negative.')
elif x==0:
    print('Equal to zero.')
elif 0<x<10:
    print('Positive but smaller than 10')
else:
    print('Positive and larger than or equal to 10.')
```

#####相等运算符==与同一性运算符is

```
x=y=[3,6,9]
z=[3,6,9]
print("x==y->{}".format(x==y))
print("x is y->{}".format(x is y))
print("x==z->{}".format(x==z))
print("x is z->{}".format(x is z))
print("x is not y->{}".format(x is not y))
print("x is not z->{}".format(x is not z))
print("id_x:{};id_y:{};id_z:{}".format(id(x),id(y),id(z))) #Memory Location

del x[2]
print("x={},y={},z={} after del x[2]".format(x,y,z))
```

#####成员资格运算符，in与not in

```
x=[3,6,9]
print("3 in x->{}".format(3 in x))
print("0 in x->{}".format(0 in x))
print("3 not in x->{}".format(3 not in x))
print("0 not in x->{}".format(0 not in x))
```

#####布尔运算符

```
a,b,c=0,10,15
if c>a and c<b:
    print('a<c<b')
else: print('a<c<b kidding!!!')
```

## 函数 function

###定义函数

```
def simple_func(x,y):
    z=pow(x,2)+y
    return z
```

```
print("simple_func(3,7)->{}".format(simple_func(3,7)))
print("simple_func(7,3)->{}".format(simple_func(7,3)))
print("simple_func(y=7,x=3)->{}".format(simple_func(y=7,x=3)))
```

#####定义斐波那契数列(Successione di Fibonacci)函数

```
def fibonacci(s,count): #定义fib函数的方法较笨
    fib_lst=[0,1]
    fib_part=[]
    if s==0 or s==1:
        fib_part[:]=fib_lst
        for i in range(count-2):
            fib_part.append(fib_part[-1]+fib_part[-2])
    else:
        while True:
            fib_lst[:]=[fib_lst[1],fib_lst[0]+fib_lst[1]]
            #print(fib_lst)
            if fib_lst[1]-s>=0:break
        fib_part[:]=fib_lst
        if abs(fib_lst[0]-s)>=abs(fib_lst[1]-s):
            for i in range(count-1):
                fib_part.append(fib_part[-1]+fib_part[-2])
            fib_part.pop(0)
        else:
            for i in range(count-2):
                fib_part.append(fib_part[-1]+fib_part[-2])
    return fib_part

print("fibonacci(6,9)->{}".format(fibonacci(6,9)))
print("fibonacci(7,9)->{}".format(fibonacci(7,9)))
```

###递归

```
def factorial(n):
    if n==1:
        return 1
    else:
        return n*factorial(n-1)
print(factorial(7))
```

## 类 class
###定义类

```
class Bird:
    fly='Whirring' #美 /wɜːr/
    def __init__(self):
        self.hungry=True
    def eat(self):
        if self.hungry:
            print('Aaaah...')
            self.hungry=False
        else:
            print('No.Thanks!')

class Apodidae(Bird):  #/'æpədədi:/
    def __init__(self):
        super(Apodidae,self).__init__()
        self.sound='Squawk!' #美 /skwɔːk/
    def sing(self):
        print(self.sound)
```

#####f类与类的实例

```
blackswift=Apodidae()
scarceswift=Apodidae()
print("blackswift.sing()->")
blackswift.sing()
print("scarceswift.sing()->")
scarceswift.sing()
```

#####类的属性

```
print("blackswift.fly->{}".format(blackswift.fly))
blackswift.fly='humming' #重新赋予实例的属性
print("blackswift.fly after redefining the blackswif's attribute->
{}".format(blackswift.fly))
print("scarceswift.fly->{}".format(scarceswift.fly))
```

#####类的方法

```
object.method()
```

#####超类+子类

```
Help on class Bird in module __main__:

class Bird(builtins.object)
 |  Methods defined here:
 |  
 |  __init__(self)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  eat(self)
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  fly = 'Whirring'
```

###迭代+迭代器

```
class Fibs():
    def __init__(self):
        self.a=0
        self.b=1
    def next(self):  #实现迭代器的next方法
        self.a,self.b=self.b,self.a+self.b
        return self.a
    def __iter__(self): #实现迭代方法
        return self
```

```
lst_d=list(range(3,20,2))
print("lst_d={}".format(lst_d))
print("_"*50)
lst_iter=iter(lst)
print("next(lst_iter)->{}".format(next(lst_iter)))
print("next(lst_iter)->{}".format(next(lst_iter)))
for i in lst_iter:
    print(i)
next(lst_iter)
```

###生成器(generator)
#####一般方法

```
lst_e=[list(range(3,20,3)),[3,7,67,list(range(5)),89]]
print("lst_e={}".format(lst_e))
print("_"*50)
flatten_lst=[]
for v_1 in lst_e:
    try:
        for v_2 in v_1:
            try:
                for v_3 in v_2:
                    flatten_lst.append(v_3)
            except TypeError:
                flatten_lst.append(v_2)
    except TypeError:
        flatten_lst.append(v_1)
print("flatten_lst={}".format(flatten_lst))
```

#####生成器方法

```
def flatten(lst): #定义生成器（包含yield语句的函数）
    try: #使用语句try\except捕捉异常
        for n_lst in lst:  #循环列表
            for m in flatten(n_lst): #使用递归的方法循环子列表
                yield m  #使用yield语句，每次产生多个值，当返回一个值时函数就会被冻结，
当再次激活时，从停止的那点开始执行
    except TypeError:  #当函数被告知展开一个元素时，引发TypeError异常，生成器返回一个值
        yield lst #生成器返回引发异常的一个值
```

#####建立无穷列表

```
def infinite():
    n=0
    while True:
        yield 'num#'+str(n)
        n+=1
```

```
n=infinite()
print("next(n)->{}".format(next(n)))
print("next(n)->{}".format(next(n)))
print("next(n)->{}".format(next(n)))
print("[next(n) for i in range(5)]->{}".format([next(n) for i in range(5)]))
```

#####生成器表达式(generator expression)

```
n=3
print("[[(2*pi*(2*(u/(n-1))-1),2*pi*(2*(v/(n-1))-1)) for u in range(n)] for v in
range(n)]->{}".format([[(2*pi*(2*(u/(n-1))-1),2*pi*(2*(v/(n-1))-1)) for u in
range(n)] for v in range(n)]))
print("_"*50)
print("([(2*pi*(2*(u/(n-1))-1),2*pi*(2*(v/(n-1))-1)) for u in range(n)] for v in
range(n))->{}".format(([(2*pi*(2*(u/(n-1))-1),2*pi*(2*(v/(n-1))-1)) for u in
range(n)] for v in range(n))))
gen=([(2*pi*(2*(u/(n-1))-1),2*pi*(2*(v/(n-1))-1)) for u in range(n)] for v in
range(n))
print("next(gen)->{}".format(next(gen)))
print("next(gen)->{}".format(next(gen)))
```

## 异常 try exception

###try/except捕捉异常

```
try:
    do something
except:
    do something
```

```
def f_convert_a(x):
    try:
        return float(x)
    except:
        return x
print("f_convert_a('3.1415')->{}".format(f_convert_a('3.1415')))
print("f_convert_a('string')->{}".format(f_convert_a('string')))
print("f_convert_a(3,6,7)->{}".format(f_convert_a((3,6,7))))
```

```
try:
    do something
except ValueError:
    do something
```

```
def f_convert_b(x):
    try:
        return float(x)
    except ValueError:
        return x
print("f_convert_b('3.1415')->{}".format(f_convert_b('3.1415')))
print("f_convert_b('string')->{}".format(f_convert_b('string')))
print("f_convert_b(3,6,7)->{}".format(f_convert_b((3,6,7))))
```

#####使用多条except子句可以指定多个异常处理代码块

```
try:
    do something
except TypeError:
    do something
except ValueError:
    do something
```

```
def f_convert_c(x):
    try:
        return float(x)
    except ValueError:
        return x
    except TypeError:
        print(x,'TypeError')
print("f_convert_c('3.1415')->{}".format(f_convert_c('3.1415')))
print("f_convert_c('string')->{}".format(f_convert_c('string')))
print("f_convert_c(3,6,7)->{}".format(f_convert_c((3,6,7))))
```

#####处理程序也可以捕捉多种类型的异常

```
try:
    do something
except (TypeError, ValueError ):
    do something
```

```
def f_convert_d(x):
    try:
        return float(x)
    except (ValueError,TypeError):
        print(x,'ValueError or TypeError')
print("f_convert_d('3.1415')->{}".format(f_convert_d('3.1415')))
print("f_convert_d('string')->{}".format(f_convert_d('string')))
print("f_convert_d(3,6,7)->{}".format(f_convert_d((3,6,7))))
```

```
try:
    do something
except (TypeError, ValueError ) as e:
    do something
```

```
def f_convert_e(x):
    try:
        return float(x)
    except (ValueError,TypeError) as e:
        return print(x,e)
print("f_convert_e('3.1415')->{}".format(f_convert_e('3.1415')))
print("f_convert_e('string')->{}".format(f_convert_e('string')))
print("f_convert_e(3,6,7)->{}".format(f_convert_e((3,6,7))))
```

#####使用pass语句可以忽略异常

```
def f_convert_f(x):
try:
    return float(x)
except ValueError as e:
    pass
```

```
def f_convert_f(x):
    try:
        return float(x)
    except (ValueError,TypeError) as e:
        pass
print("f_convert_f('3.1415')->{}".format(f_convert_f('3.1415')))
print("f_convert_f('string')->{}".format(f_convert_f('string')))
print("f_convert_f(3,6,7)->{}".format(f_convert_f((3,6,7))))
```

#####try语句也支持else子句，但是不需跟在最后一个except子句的后面。如果try代码块中的程序没有引发异常，将会执行else子句中的程序

```
def f_open_a(fp):
try:
    do something
except IOError as e:
    do something
else:
    do something
```

```
def f_open_a(fp):
    try:
        f=open(fp,'r')
    except IOError as e:
        print('Unable to open',fp,':%s\n' %e)
    else:
        data=f.read()
        f.close
        return data
tryException_content=f_open_a("./data/tryException.txt")
print("tryException_content->{}".format(tryException_content))
f_open_a("./data/tryException_.txt")
```

```
def f_open_a(fp):
try:
    do something
except IOError as e:
    do something
else:
    do something
finally:
    do something
```

```
def f_open_b(fp):
    try:
        f=open(fp,'r')
    except IOError as e:
        print('Unable to open',fp,':%s\n' %e)
    else:
        data=f.read()
        f.close()
        return data
    finally:
        print('done!')
f_open_b("./data/tryException.txt")
```