输出\必须用\\
num=random.randint(a,b)　#从 a 到 b 随机选取一个数字，并返回它的 int 形式
list_1=random.sample(abc,4)#从序列 abc（可以是字符串，列表之类的）中随机抽取 4 个生成列表，并返回此列表

len()
id()
type()
[a:b]切片　[a:]　[:b]　[-1]　[: : c]　[: : -1]

{} .format
f’　’
print(sep=’ ’,end=’ ’)
abs(-10)
#向上取整
math.ceil(3.14)
#向下取整
math.floor(3.14)
#开根号
math.sqrt(16)

除 0 外 int 全做 True
if 接受连续比较

for num in range(0, 12 , 2)　0-11 每次加 2　　也可以直接 range(3)
列表可以同时存多种类型数据
字符串可以通过下标取到其中某个字符
可以直接用 in 判断

dir(__builtins__)　查看内置函数
help(‘keywords’)　查看关键词
dir(变量)　获得该变量所有方法
help(变量.方法)　获得该变量对应方法说明书

列表

list_data = ['杨幂', 1, [1,3,5]]
#空列表创建
list_1 = []
list_2 = list()
list_1.append(数据)
list_1.insert(下标，数据)

#0，1 删除，插入贾玲和 35
list_2[0:2] = ['贾玲', 35] #也可以写 '贾玲'，35
#0，1 删除，插入 1，2，3，4，5。下标为 2 的数据不动，向后顺移
list_2[0:2] = 1,2,3,4,5
list_2.pop(2) 删除 list_1 中下标为 2 的数据并返回此数据 不填默认删最后一个
list_2.remove('贾玲') 基于数据删除 **删除第一个遇到的值，不会删除多个重复值
del list_2[0:2] 基于范围删除数据 不带[]则直接整个删除
list_2.clear() 清空列表
list_1.sort() 列表排序 reverse=True 从小到大
list_1.reverse() 列表翻转 只反转列表，但函数无返回值
numbers.count(1) 列表元素量统计，1 在 numbers 列表中的个数
List_2 = list_1 * 3 列表复制
list_2=list_2+list_1
list_1.extend(list_2)
复制
list1=list2 list1id 与 list2id 完全相同，一个改另一个也改
list1=list2.copy() list1 与 list2id 不同，但内存每个数据地址完全相同，一个改另一个也改
import copy list1=copy.deepcopy(list2) list1 与 list2 完全不同
列表切片生成还是列表 例 list_1=list_2[0:15:2]
join 列表元素合并
new_string='a'.join(list) #以 a 为分隔符，将列表 list 中的数据连接成字符串并返回此字符串
split 拆分
list1=string1.split('，') 以，为切割点拆分字符串 string1 为列表
列表的几个小功能 max min sum 函数，输出列表内最大、最小值或值的总和

元组

tuple=(1, '杨幂', [1,2,3])
#空元组创建 啥用没有，又不能往里写
tuple_1 = ()
tuple_2 = tuple()
一般先有一个列表，列表->元组，上锁；同样可以解锁
tuple_1=tuple(list_1)
#一个值的元组创建 #特别注意这里跟列表有区别 列表可以['杨幂']创建一个值，但是元组不能('杨幂')
tuple_1 = ('杨幂' ,) 或
tuple_2 = '杨幂' ,
对应的，两个值的就可以
tuple_2='杨幂' , 12 自动识别为 tuple
多变量操作
return_data = ('杨幂', '吉林大学')
name, school = return_data

元组的第一级数据不可以修改，但是第二级数据可以根据 当前数据类型进行修改
tuple_1 =tuple_1 + tuple_2　　元组合并
num=tuple_1.count(12)　元素量统计
tuple_1=tuple_2*3 自身复制
in 包含关系
元组[: :]切片生成元组

字典

```
dict_data = {
    '学校':'吉林大学',  #字符串
    '成立于':1946,    #整数
    '现任领导':['姜智莹', '张希'],  #列表
    '学院':{'计算机学院', '软件学院', '数学学院'},  #集合
    '部门':('教务处', '财务处', '科技处'),  #元组
    '老师':{'杨幂':'Python 程序设计', '贾玲':'数据结构'},  #字典
}                                                                    必为 1 对 1
#空字典创建
dict_1 = dict()
dict_2 = {}
```

奇怪的创建方式

```
dict_1 = dict({'姓名':'mike', 'age':18})
dict_2 = dict(姓名='mike', age=18)
dict_1['城市'] = '上海'    #这个键不存在 就是添加，如果存在 就是修改
```

查询常规方法 print(dict_1[‘城市’])
带有异常返回的处理方法 dict_1.get('年龄') 不存在则返回 None
dict_1.pop(‘城市’)　删除数据，和 list 默认 pop 最后一个不同，这个删除必须给出 key
dict_1.popitem() 删除最后的键值对并返回它
dict_1.clear()
del dict_1[‘城市’]
del dict_1
快速初始化所有 key

```
keys = ['姓名', '电话', '年龄', '学历', '年龄']
new_dict = dict.fromkeys(keys, 1)    不填 1 默认为 None
```

核心功能
dict_1.items()　dict_1 中每个 key 和 data 为一对生成元组组成列表，搭配多变量赋值

```
for k,v in personal_info.items():
```

dict_1.keys()
dict_1.values()
字典更新
dict_1.update(dict_2)　　用 2 更新 1，遇到重复 key，值更新，遇到不存在则直接添加
dict.setdefault(key, default=None)　查询 key 不存在则直接添加，默认值为 None，返回 key

的值
in 包含关系仅能查找到 keys
一般用列表套字典，每个字典针对某个对象的各个信息

集合

set 创建
b = set()
或
a = [4,1,2,3,4,1,3,2,1,3,4,5,3,2,2,4,2,1]
my_set = set(a)

name_set.add('沈腾') 添加数据
多对象批量添加
name_set.update(d) d 为 list，tuple，添加所有 list，tuple 元素，d 为 dict，添加所有 keys
name_set.update('沈腾','李诞') 添加'腾', '沈', '李', '诞'
删除
data_set.remove('杨幂') 无此元素则报错
data_set.pop() 删除最后一个，返回删除的值，不可指定删除对象
data_set.discard('李诞') 无此元素也不会报错
data_set.clear()

枚举
enumerate(sequence, [start=0]) 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中

```
for index,value in enumerate(data_set):
    print(index,value)
```

```
#交集操作
print(data_A.intersection(data_B))  A 不变
print(data_A&data_B)
print(data_A.isdisjoint(data_B)) #检查是否有交集  有相同元素则返回 False
#并集操作
print(data_A.union(data_B))
print(data_A|data_B)
#差集操作
print(data_A.difference(data_B))  返回 A 删去与 B 重叠的元素后生成的集合，data_A 不变
print(data_A-data_B)
set_1.difference_update(set_2)  set_1 改变
#补集操作
print(data_A.symmetric_difference(data_B))
print(data_A^data_B)
```

函数

```
def hello(num1, num2):
    res_1 = num1 + num2
    print('Hello Python!')
    return 'I am return!'
```

可以同时 return 多个值，以元组形式返回

return 后边只能支持表达式 或 一个直接返回结果的函数(一种直接的逻辑)，不支持常规语句

my_func = my_function() 赋值结果给变量时函数即运行

打印自定义函数的说明

help(my_function)

print(my_function.__doc__)

传参

关键字传参支持位置对调 func（a=1，b=2）

关键字形参必须在位置形参后

形参顺序：位置形参，星号形参*args，默认参数，关键字形参，双星号形参**kwargs

全局变量

不可变类型全局变量必须 global 引入函数，不然会直接函数内新建变量

可变类型全局变量不需要引入

函数的引用

```
def func(num1 ,num2):
  pass
```

func1=func 相当于 func1 是 func 的别名？如果 func1=func(1,2)，那就直接 func1 等于 func(1，2)返回的值了

递归

```
import turtle
```

turtle.forward(long) 向前移动画笔 long

turtle.backward(long) 向后移动画笔 long

turtle.right(degree) 画笔向右旋转 degree 度

turtle.exitonclick() 点击退出绘图

turtle.penup() 抬起画笔

turtle.pendown() 放下画笔

turtle.pensize() 设置画笔 size

turtle.pencolor() 设置画笔 color

匿名函数

```
res_2 = lambda n:n*2
```

lambda 参数:返回值

```
print(res_2(3))
result = lambda x,y: x+y
result(1,2)
```
三目 result = 'a 大于 b' if a>b else 'a 小于 b' if a>b 返回 a 大于 b 否则返回 a<b

```
def outer(n,x,y):
    print('外层  n ->', n,x,y)
    return lambda x,y: x + y + n    # (return 另一个函数的待命模式)
                #(参数)   #(逻辑运算)
f = outer(1,2,3)
f(8,10)

def keyword_search(keys, func):
    search_result = []
    for result in keys:
        if func(result):
            search_result.append(result)
    return search_result

def search_condition(c):
    return lambda x: True if c in x else False
python = search_condition('Python')    //先传参数使方法变函数，后传参数使用函数
java = search_condition('Java')
keyword_search(schools, python)
keyword_search(schools, java)
```
四个函数
map
map(func, data) map 返回的结果是一个迭代器，需要 list 转一下即可调用
```
new_1 = list(map(lambda n:n+1, data))
```
reduce
```
from functools import reduce
data = [1, 2, 3, 4, 5]
res = reduce(lambda x,y: x+y, data)
```
两个两个把 data 中元素传入设置的函数中并把结果插回 data
reduce 直接返回 int 结果
filter
```
res = list(filter(lambda x: x>2, data))
```
filter 中函数传参后返回的结果必须是 bools 类型，计算机根据返回的是 false 还是 true 判断是不是返回这个元素
也是直接生成列表
sorted
```
exam_result = {'杨幂':'540', '贾玲':'575', '沈腾':'583', '李诞':'569', '周奇墨':'557'}
dict_order = sorted(exam_result.items(), key = lambda d:d[1], reverse=True)
```
True 从大到小排列(降序)
```
dict_order=dict(dict_order)
```

```
l_data = [ [3,4,1], [3,3,3], [1,2,4], [9,1,0], [7,3,2]]
list_order = sorted(l_data, key=lambda d:d[0], reverse=False)
asc_order = sorted(l_data, key=lambda d:(d[0],d[1]), reverse=False)
```

装饰器

函数闭包：
必须一个函数在另一个函数内
内函数必须引用外函数的变量
封闭函数必须返回嵌套函数

```
def deco_func(original_func):
    def wrapper_func():
        pass
        return original_func()
    return wrapper_func
@deco_func
def show_age():
    print('I am 28')
show_age()

def deco_func(original_func):
    def wrapper_func(*args, **kwargs):
        pass
        return original_func(*args, **kwargs)
    return wrapper_func
@deco_func
def show_age():
    print('I am 28')
@deco_func
def show_info(name, age):
    print(name, age)

from functools import wraps

def a(func):
    @wraps(func)
    def warp():
        print('123')
        print(warp.__name__)
        return func()
```

```
    return warp

def b(func):
    @wraps(func)
    def warp():
        print('321')
        print(warp.__name__)
        return func()
    return warp

@a
@b
def xx():
    print('abc')
xx()
```

推导式

```
my_list_3 = [x for x in range(20)]
list_data = [x**2 for x in range(10) if x%2 == 0]
list_data = [(x,y) for x in [1,2,3] for y in [4,5,6] if x > 2 and y > 4]
list_data_new = [y for x in list_data for y in x]
list_data_new = [x for x in list_data for y in x]
```

即使 y 没用它也会执行

```
fields = ['姓名', '年龄', '城市']
info = ['杨幂', '28', '长春']
dict_data = {x:y for x,y in zip(fields, info)}
```

迭代器

生成迭代器（只能取一次的小仓库，取完就再也取不出来了）

new_iter = iter(l_data) l_data 为可迭代对象

new_magic = l_data.__iter__()

直接打印迭代器是无法获取数据的

迭代器会重新建立一个对象 一个迭代器对象 虽然看起来使用了 l_data，但是删除 l_data 之后迭代器依然存在

取数

```
next(new_magic)
new_iter.__next__()
for x in new_magic:
```

```
    print(x)
```
但不能 print(new_magic)

生成器

```
def gen_num(start, end):
    n = start
    while n < end:
        yield n    #yield 返回一次后会阻塞当前循环，直到生成器再次被调用才会行继续执
```
行
```
        n += 1

my_num = gen_num(10，20)     my_num 才是最终生成器
```
使用生成器
```
next(my_num)
for x in my_num:
    print(x)
```

元组加推导式生成器
```
my_gen = (x for x in range(10))
```

正则表达式
```
import re
txt='吉林大学'
patt=r'吉林'
res = re.findall(patt,txt)
```
res 为列表形式，存的是匹配到的所有字符串
```
res = re.findall(patt, txt, re.M)

txt = """
```
我在吉大学 Python
我在吉大学 Django
我在吉大学 Java
我在吉大学 C++
```
"""
patt = re.compile(r'学.')   compile 编译的意思，这里是编译当前的表达式
res = patt.findall(txt)
```

. 除了换行以外的任何字符
\d 取数字
\w 匹配字母、数字、下划线
\b 代表单词的开始或者结束 (以标点、空格、换行作为分割)
[ ]根据条件筛选匹配 如[a-eA-E]

re.S 换行也照样取值 (使 . 取值包含换行内的所有字符)
\s 匹配任何空白字符，那就是空格 和回车\n
\S 匹配任何非空白字符

＋匹配前一个表达式的规则任意次数（至少出现一次）
* 匹配前一个表达式的规则任意次数（不出现/0 次也包含）
{ }匹配指定次数 {3}指定 3 次 {2,5}指定 2-5 次范围
()数组分组

^从头取值 $从尾取值

或 or | 双多条件匹配

.* 的贪婪模式
.*? 非贪婪模式

切割 split 返回切割出来每个字符串组成的列表
patt = r'[*/&#]'
res = re.split(patt, qq)
匹配 match 传入字符串，字符串中有与之匹配的片段，则返回 bool
qqpatt = r'[1-9][0-9]{4,10}@qq.com'
qqres = re.match(qqpatt, qq) qq 是一个字符串

想将.作为通配符，则用\.

包

包是一个文件夹，里有模块.py 和一个__init__.py
直接导入
import random
跳跃式
from time import sleep
贪婪式
from time import *
自定义名称
from time import sleep as fight

import sys
sys.path.append('模块或包的路径')
print(sys.path)

文件处理

```
import os
import shutil
import zipfile
```

面向对象

```
class TeaShop：
    brand=’蜜雪冰城’
    __total=0
    def __init__(self,name1,size1):
        self.name=name1
        self.size=size1
        TeaShop.__total=TeaShop.__total+1
    def func(self,a):
        print(a)
    @classmethod:
    def total(clas):
        print(‘产品总数’,__total)
    @staticmethod:
    def info_channel():
        print(‘屁用没有’)
    def __del__:
        pass

tea_1=TeaShop(‘三拼爸爸’,’大杯’)
tea_1.func(‘h’)
TeaShop.brand=’蜜雪冰城 plus’
tea_1.size=’小杯’
TeaShop.total()
tea_1.info_channel()
print(tea_1._TeaShop__total)

class NewType(TeaShop):
    def __init__(self,name1,size1,price1):
        super().__init__(name1,size1)
        self.price=price1
    def func(self):
        print(‘重写父类函数’)
```