目录

运算符

算术运算符 +,-,*, /, %, //, **

x = 10

y = 3.0

x / y

```
3.3333333333333335
```

x // y

```
3.0
```

x % y

```
1.0
```

x ** y

```
1000.0
```

注：等量公式 x%y=x-(x//y)*y

decimal 模块

作用：使小数加减结果准确

from decimal import Decimal

0.1+0.1+0.1-0.3

```
5.551115123125783e-17
```

Decimal('0.1')+Decimal('0.1')+Decimal('0.1')-Decimal('0.3')

```
Decimal('0.0')
```

注：Decimal()括号中必须是字符串

Decimal(0.1)+Decimal(0.1)+Decimal(0.1)-Decimal(0.3)

```
Decimal('2.77555756156515654042363166 8E-17')
```

fractions 模块

作用：提供分数运算

from fractions import Fraction

1. 默认 Fraction(分子=0,分母=1)

print(Fraction(4))

```
4
```

2. 字符串形式，返回分数形式

print(Fraction('0.4'))

```
2/5
```

3. 运算

x = Fraction(1, 3)

y = Fraction(4, 6)

x + y

```
Fraction(1, 1)
```

x * y

```
Fraction(2, 9)
```

随机数模块

导入模块方法：import random
用法：a=random.randint(1,10)　#在 1-10 中取随机数给变量 a
应用：
①随机生成一个(-1,1)区间的数
random()*2-1
②用蒙特卡洛法估计圆周率

```
from random import random
n=100000
n0=0
for i in range(n):
    x=random()*2-1
    y=random()*2-1
    if(x**2+y**2<1):
        n0+=1
pi=4*n0/n
print('pi=',pi)
```

```
pi= 3.14156
```

列表

列表对象支持的运算符

运算符	功能	说明
+	连接列表	生成新列表
+=	向列表追加元素	原地操作，修改原列表
*	重复列表	生成新列表
*=	重复列表	原地操作，修改原列表

添加

.append( )　在列表最后添加一个元素

```
s=[1,2,3,4,5]
s[len(s):] = [6]　#等价于 s.append(6)
```

.extend( )　在列表最后添加多个元素

```
s=[1,2,3,4,5,6]
s[len(s):] = [7,8,9]　#等价于 s.extend([7,8,9])
```

.insert( )　在列表中任意位置插入

```
s=[1,3,4,5]
s.insert(1,2)　#第一个数字为下标，第二个数字为插入元素
```

删除

del 语句　删除指定下标元素

s=[1,2,3,4,5,6]

del s[4] #删除下标为 4 的元素

.remove( )　删除( )内指定元素

s=[1,2,3,4,5,6]

s.remove(5)　#删除 s 中的元素 5

.pop( )　删除( )内下标对应元素

s=[1,2,3,4,5,6]

s.pop(4)　#删除下标为 4 的元素，即元素 5

.clear( ) 删除列表中所有元素

替换

①单个指定元素替换

s=[1,2,3,4,5,6]

s[1]=9　#把元素 2 换成 9

②多个元素替换

s=[1,2,3,4,5,6]

s[3:]=[7,8,9]　#将元素 4,5,6 换成 7,8,9

排序

.sort( ) 从小到大排序

s=[5,6,1,4,3,2]

s.sort( )　#使元素变成 1,2,3,4,5,6

.reverse( ) 颠倒顺序

s=[5,6,1,4,3,2]

s.reverse( )　#使元素变成 2,3,4,1,6,5

.sort(reverse=True) 从大到小排序

s=[5,6,1,4,3,2]

s.sort(reverse=True)　#使元素变成 6,5,4,3,2,1

查找

.count( )　统计某个元素个数

s=[1,2,2,2,3,9,8,5]

s.count(2)　#查找 2 的个数

.index( )　查元素索引值

s=[1,2,2,2,3,9,8,5]

s.index(3)　#3 的索引值为 4

#可以指定查找范围

s.index(3，1，5) #在 1-5 号元素中查找 3 的位置

.copy( )　复制元素

浅拷贝

s=[1,2,2,2,3,9,8,5]

ss=s.copy( )　#将 s 元素拷贝到 ss 中

sss=s[:] 这样也可以拷贝整个列表

浅拷贝的理解：

s=[[1,2,3],[4,5,6],[7,8,9]]

y=s.copy( )

s[1][1]=0

```
s
[[1, 2, 3], [4, 0, 6], [7, 8, 9]]
y
[[1, 2, 3], [4, 0, 6], [7, 8, 9]]　#改变一个里面的值，另一个也会改变
```

深拷贝

import copy　#导入拷贝模块

s=[[1,2,3],[4,5,6],[7,8,9]]

y=copy.deepcopy(s)

s[1][1]=0

```
s
[[1, 2, 3], [4, 0, 6], [7, 8, 9]]
y
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]　#不会全部改变
```

列表推导式

①列表数值*2

s=[1,2,3,4,5]

s=[i*2 for i in s]　#实现将中所有元素*2

构建方法：

s=[i+1 for i in range(10)]

```
s
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

#等同于

s=[ ]

for i in range(10):

　s.append(i+1)

②处理字符串

s=[s*2 for s in “cym”]

```
s
['cc', 'yy', 'mm']
```

ord 函数　将单个字符串转换成对应编码

code=[ord(s) for s in “cym”]

```
code
[99, 121, 109]
```

③二维列表处理

s=[[1,2,3],

　[4,5,6],

```
   [7,8,9]]
[num for elem in s for num in elem]   #二维列表的平铺
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
col2=[row[2] for row in s]     #提取每行第三个元素
```

```
col2
[3, 6, 9]
```

```
s=[[1,2,3],
   [4,5,6],
   [7,8,9]]
diag=[s[i][i] for i in range(len(s))]   #获取对角线元素
```

```
diag
[1, 5, 9]
```

```
diag=[s[i][2-i] for i in range(len(s))] #获取另一个对角线元素
                                        #还可以写成 diag=[s[i][-i-1] for i in range(len(s))]
```

```
diag
[3, 5, 7]
```

④二维列表创建

```
s=[[0]*3 for i in range(3)]
 #等同于
s=[0]*3
for i in range(3):
```

```
s
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

错误创建：

```
s=[[0]*3]*3
#虽然看似结果一样，但一旦改变其中一个值所有值都会改变
s[1][1]=1
```

```
s
[[0, 1, 0], [0, 1, 0], [0, 1, 0]]
```

```
#其他两种上述写法则不会
```

Ⅰ.创建时可以进行筛选

```
s=[i for i in range(10) if i % 2 == 0]   #只存入偶数
```

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

Ⅱ.筛选特定内容

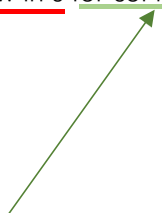
```
s=[“Great”,”Fantastic”,”Excellent”,”Formal”,”Victory”]
fwords=[ for w in s if w[0]==’F’]
```

```
fwords
['Fantastic', 'Formal']
```

Ⅲ.嵌套列表推导式

一阶：

```
s=[[1,2,3],[4,5,6],[7,8,9]]
flatten=[col for row in s for col in row]
```



```
flatten
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
#等同于
s=[[1,2,3],[4,5,6],[7,8,9]]
flatten = [ ]
for row in s:
    for col in row:
        flatten.append(col)
```

二阶：

```
[[x,y] for x in range(10) if x % 2 == 0 for y in range(10) if y % 3 == 0]
```

```
[[0, 0], [0, 3], [0, 6], [0, 9], [2, 0], [2, 3], [2, 6], [2, 9], [4, 0], [4, 3],
 [4, 6], [4, 9], [6, 0], [6, 3], [6, 6], [6, 9], [8, 0], [8, 3], [8, 6], [8, 9]]
```

```
#等同于
s=[ ]
for x in range(10):
    if x % 2 == 0:
        for y in range(10):
            if y % 3 == 0:
                s.append([x,y])
```

⑤矩阵转置

元组

①使用方法

```
rhyme=(1,2,3,4,5,"上山打老虎")
#等同于   rhyme=1,2,3,4,5,"上山打老虎"
```

②索引

```
rhyme[0]      #索引第一个元素 1
```

注：元组元素不可变无法修改其中元素

```
rhyme[1] = 10
```

```
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    rhyme[1] = 10
TypeError: 'tuple' object does not support item assignment
```

③切片

和列表一样

```
rhyme[ :3]    #前三个元素，即 1,2,3
rhyme[ : :-1]   #所有元素倒叙
rhyme[ : : 2]   #输出跨度为 2
```

④查

和列表相同

⑤拼接

```
s=(1,2,3)
```

```
t=(4,5,6)
s+t
```

```
(1, 2, 3, 4, 5, 6)
```

⑥重复

```
s=(1,2,3)
s*3
```

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

⑦嵌套

```
s=(1,2,3)
t=(4,5,6)
w=s,t
```

```
w
((1, 2, 3), (4, 5, 6))
```

⑧列表推导式

```
s=(1,2,3,4,5)
[each*2 for each in s]
```

```
[2, 4, 6, 8, 10]
```

⑨生成一个元素的元组

```
x = (520,)
```

```
x
(520,)
type (x)    #查看 x 的类型
<class 'tuple'>    #为元组
```

错误生成：

```
x = (520)
```

```
x
520
type(x)
<class 'int'>    #x 为整型，并非元组
```

⑩打包和解包（列表相同）

```
s=(123,"cym",3.14)    #此过程为打包
x,y,z=s    #此过程为解包
```

```
x
123
y
'cym'
z
3.14
```

PS：Python 可以多重赋值，原理就是元组打包

```
x,y=10,20    #将 10 赋值给 x，将 20 赋值给 y
```

⑪当元组元素指向一个可变列表时，其中元素可变

```
s=[1,2,3]
t=[4,5,6]
```

w=(s,t)

```
w
([1, 2, 3], [4, 5, 6])
```

w[1][1]=10

```
w
([1, 2, 3], [4, 10, 6])
```

字符串

大小写字母换来换去

.capitalize( ) 首字母大写，其他字母小写

s="i love My mum foreVer"

s.capitalize( )

```
'I love my mum forever'
```

.casefold( ) 所有字母全小写（可以处理其他语言）

s.casefold( )

```
'i love my mum forever'
```

.title( ) 每个单词首字母大写,其他字母小写

s.title( )

```
'I Love My Mum Forever'
```

.swapcase( ) 大小写翻转

s.swapcase( )

```
'I LOVE mY MUM FOREvER'
```

.upper( ) 所有字母大写

s.upper( )

```
'I LOVE MY MUM FOREVER'
```

.lower( ) 所有字母小写（只可以处理英文）

s.lower( )

```
'i love my mum forever'
```

左中右对齐

.center( ) 居中

s="Charming，是大帅哥!"

s.center(5) #若输入宽度小于字符串长度，则字符串原样输出

```
' Charming，是大帅哥!'
```

s.center(25) #大于则居中

```
'        Charming，是大帅哥!        '
```

.ljust( ) 左对齐

s.ljust(25)

```
'Charming，是大帅哥!                '
```

.rjust( ) 右对齐

s.rjust(25)

```
'                  Charming，是大帅哥!'
```

.zfill( ) 左侧用 0 补齐

s.zfill(25)

```
'00000000000Charming，是大帅哥!'
```

PS:负数也可以

x='-520'

x.zfill(5)

```
'-0520'
```

另外可以用任意字符填充空格(只拿 center 举例):

s=" Charming，是大帅哥!"

s.center(25,'棒')

```
'棒棒棒棒棒棒 Charming，是大帅哥!棒棒棒棒棒'
```

查找

.count( ) 统计某个元素个数

s="上海自来水来自海上"

scount("海",0,5) #在 0-4 围内查找"海"的字数

.find( ) 从左往右查找索引下标值

s="上海自来水来自海上"

s.find("海") #查找"海"的下标值

```
1
```

.rfind( ) 从右往左查找索引下标值

s.find("海")

```
7
```

.index 和 find 基本相同，唯有对字符串没有的元素查找时给出的结果不同

s.find("陈")

```
-1
```

s.index("陈")

```
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    s.index('陈')
ValueError: substring not found
```

替换

.expandtabs( ) 将 Tab 换成空格，()中参数指定一个 Tab 等于多少空格

s="""

```
        print("I love you")
    print("You love me")"""
```

ss=s.expandtabs(4)

```
ss
    print("I love you")
```

```
    print("You love me")
```

.replace("旧类容","新类容",参数) 替换文字类容，其中参数不设置默认为-1，即全部替换

'在吗？你在干啥啊？！'.replace('在吗？','想你！')

```
'想你！你在干啥啊？！'
```

.translate(table)　返回一个根据 table 参数转换后的新字符串

*table 获取方法：str.maketrans( )，其支持第三个参数，第三个参数表示这个字符串忽略

table= str.maketrans("ABCDEF","123456") #第一个""里类容换成第二个""中类容

"CYM is HANDSOME!". translate(table)

```
'3YM is H1N4SOM5!'
```

#上面写法等同于"CYM is HANDSOME!". translate(str.maketrans("ABCDEF","123456"))

table= str.maketrans("ABCDEF","123456","is")

"CYM is HANDSOME!". translate(table)

```
'3YM　H1N4SOM5!'
```

判断

.startswith　判断指定的子字符串是否为字符串起始位置

x='我爱 Python'

x.startswith('我')

```
True
```

元组可以写入多个待匹配元素

x='我爱 Python'

if x.startswith(('你','我','她')):

　print('大家都爱 Python')

```
大家都爱 Python　#'你','我','她'中'我'匹配成功
```

.endswith　判断指定的子字符串是否为字符串结束位置

x='我爱 Python'

x.endswith('cym')

```
False
```

x.endswith('Py',0,4)　#从下标[0]到[3]之间索引

```
True
```

.istitle　测试字符串中每个单词是否以大写字母开头，其他为小写字母

.isupper 测试字符串中是否所有字母全为大写

.isalpha 测试字符串中是否全为字母

'I love Python' .isalpha( )

```
False　#因为空格不是字母
```

.isspace 判断是否为一个空白字符串

注：Tab，空格，\n 等都为空白字符串

.isprintable　判断字符串是否全部可以打印

注：转义字符不可以打印

.isdecimal( )　.isdigit( )　.isnumeric( )　都是测试数字的，具体区别如下图：

```
>>> x = "12345"
>>> x.isdecimal()
True
>>> x.isdigit()
True
>>> x.isnumeric()
True
>>> x = "2²"
>>> x.isdecimal()
False
>>> x.isdigit()
True
>>> x.isnumeric()
True
>>> x = "Ⅰ ⅡⅢⅣ Ⅴ"
>>> x.isdecimal()
False
>>> x.isdigit()
False
>>> x.isnumeric()
True
>>> x = "一二三四五"
>>> x.isdecimal()
False
>>> x.isdigit()
False
>>> x.isnumeric()
True
```

.isalnum( )　.isdecimal( ) 、 .isdigit( ) 、 .isnumeric( ) 、 .isalpha 这四个中有一个返回 True，它都会返回 True

.isidentifier( )　检测是否为 Python 合法标识符

.iskeyword( ) 判断是否为 Python 保留标识符（需要调用 keyword 函数）

import keyword

keyword.iskeyword(“if”)

```
True
```

keyword.iskeyword(“cym”)

```
False
```

截取

.lstrip( )　去除左侧空白

'　　I love Python'.lstrip( )

```
'I love Python'
```

.rstrip( )　去除右侧空白

. strip( )　去除左侧和右侧空白

注：( )中可以添加想去除的字符串，这个删除的是单个字符

'www.baidu.com'.strip('wm.')

```
'baidu.co'
```

.removeprefix( )　删除指定前缀

.removesuffix( )　删除指定后缀

注：这个两个删除的是指定的整个字符串

'www.baidu.com'.removeprefix('www.')

```
'baidu.com'
```

拆分&拼接

.partition( )　从左到右找分隔符，并从指定分隔符分开，形成三元组

'www.bai.du.com'.partition('.')

```
('www', '.', 'baidu.com')
```

.rpartition( )　从右到左找分隔符，并从指定分隔符分开，形成三元组

'www.bai.du.com'.rpartition('.')

```
('www.bai.du', '.', 'com')
```

.split( )　从左往右切割成很多小块

.rsplit( )　从右往左切割成很多小块

注：括号里两个参数，第一个指定分隔符，第二个指定分隔次数，默认值为-1，即全部切割

“大丘丘病了，二丘丘瞧，三丘丘采药，四丘丘熬”. split(‘，’,2)

```
['大丘丘病了', '二丘丘瞧', '三丘丘采药，四丘丘熬']
```

.splitlines( )　按行分割，结果以列表形式输出（可以识别各种换行符：\n,\r,\r\n）

“大丘丘病了\n 二丘丘瞧\r 三丘丘采药\r\n 四丘丘熬”. splitlines( )

```
['大丘丘病了', '二丘丘瞧', '三丘丘采药', '四丘丘熬']
```

“大丘丘病了\n 二丘丘瞧\r 三丘丘采药\r\n 四丘丘熬”. splitlines(True)

```
['大丘丘病了\n', '二丘丘瞧\r', '三丘丘采药\n', '\r', '四丘丘熬']
```

.join( )　拼接，（）里可以用列表和元组

“.”.join([‘www’,’baidu’,’com’])

起到分隔符作用

```
'www.baidu.com'
```

格式化

.format( )

year=2003

“我出生于{}年”.format(year)

```
'我出生于 2003 年'
```

还可以添加多个元素

“{}看到{}就很激动”.format(“Charming”,”小姐姐”)

```
'Charming 看到小姐姐就很激动'
```

“{1}看到{0}就很激动”.format(“Charming”,”小姐姐”)　#其中{}里数字代表下标索引值

```
'小姐姐看到 Charming 就很激动'
```

关键字参数

"我叫{name}，我爱{fav}".format(name="Charming",fav="Python")

```
'我叫 Charming，我爱 Python'
```

对齐方式

值	含义
'<'	强制字符串在可用空间内左对齐（默认）
'>'	强制字符串在可用空间内右对齐
'='	强制将填充放置在符号（如果有）之后但在数字之前的位置（这适用于以 "+000000120" 的形式打印字符串）
'^'	强制字符串在可用空间内居中

"{1:>10}{0:<10}".format(520,250)　#关键字参数也可以使用

位置索引　对齐方向　显示宽度

```
'         250520          '
```

在显示宽度前面+0，表示为数字型启用感知正负号的 0 填充效果，只对数字有效

"{:010}".format(-520)

```
'-000000520'
```

符号选项（仅对数字有效）

值	含义
'+'	正数在前面添加正号（+），负数在前面添加负号（-）
'-'	只有负数在前面添加符号（-），默认行为
空格	正数在前面添加一个空格，负数在前面添加负号（-）

"{:+} {:-}".format(520,-250)

```
'+520 -250'
```

***千分位分隔符：“，”和“—”**

"{:,}".format(123456789)　#如果位数不足则不显示分隔符

```
'123,456,789'
```

精度（不允许应用在整数上）

※ 对于 [type] 设置为 'f' 或 'F' 的浮点数来说，是限定小数点后显示多少个数位

※ 对于 [type] 设置为 'g' 或 'G' 的浮点数来说，是限定小数点前后一共显示多少个数位

※ 对于非数字类型来说，限定的是最大字段的大小

※ 对于整数类型来说，则不允许使用 [.precision] 选项

"{:.3f}".format(3.1415926)　#精度为 3

```
'3.142'
```

"{:.3g}".format(3.1415926)

```
'3.14'
```

数据呈现

值	含义
'b'	将参数以二进制的形式输出
'c'	将参数以 Unicode 字符的形式输出
'd'	将参数以十进制的形式输出
'o'	将参数以八进制的形式输出
'x'	将参数以十六进制的形式输出
'X'	将参数以十六进制的形式输出
'n'	跟'd'类似，不同之处在于它会使用当前语言环境设置的分隔符插入到恰当的位置
None	跟'd'一样

整数

"{:b}".format(123)　#二进制

```
'1111011'
```

"{:c}".format(123)　#Unicode 字符

```
'{'
```

"{:d}".format(123)　#十进制

```
'123'
```

"{:o}".format(123)　#八进制

```
'173'
```

"{:x}".format(123)　#十六进制

```
'7b'
```

注：加#可以加一个前缀，方便区分进制

"{:#x}".format(123)

```
'0x7b'
```

值	含义
'e'	将参数以科学记数法的形式输出（以字母'e'来标示指数，默认精度为 6）
'E'	将参数以科学记数法的形式输出（以字母'E'来标示指数，默认精度为 6）
'f'	将参数以定点表示法的形式输出（"不是数"用'nan'标示，无穷用'inf'标示，默认精度为 6）
'F'	将参数以定点表示法的形式输出（"不是数"用'NAN'标示，无穷用'INF'标示，默认精度为 6）
'g'	通用格式，小数以'f'形式输出，大数以'e'的形式输出
'G'	通用格式，小数以'F'形式输出，大数以'E'的形式输出
'n'	跟'g'类似，不同之处在于它会使用当前语言环境设置的分隔符插入到恰当的位置
'%'	以百分比的形式输出（将数字乘以 100 并显示为定点表示法（'f'）的形式，后面附带一个百分号
None	类似于'g'，不同之处在于当使用定点表示法时，小数点后将至少显示一位；默认精度与给定值所需的精度一致

浮点数

f-字符串

year=2003

F"我出生于{year}年"　#这里用 f 亦可

```
'我出生于 2003 年'
```

F"{123456789:,}"

```
'123,456,789'
```

序列

id 函数

对于可变序列，id 是不会改变的

s=[1,2,3]

id(s)

```
2312540580608
```

s=s*2

s

[1, 2, 3, 1, 2, 3]

id(s)

```
2312572170496
```

对于不可变序列，id 会改变

```
t=(1,2,3)
id(t)
```

```
2312572119168
```

```
t*=2
t
(1, 2, 3, 1, 2, 3)
id(t)
```

```
2312571098336
```

is&is not 运算符　检测对象 id 值是否相等

```
x=”cym”
y=”cym”
x is y
```

```
True
```

```
x=[1,2,3]
y=[1,2,3]
x is y
```

```
False
```

in&not in 运算符 判断是否包含在系列中

```
"ming" in "Charming"
```

```
True
```

del 语句　删除一个或多个指定的对象

```
x=”cym”
y=[1,2,3]
del x,y
```

```
x
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    x
NameError: name 'x' is not defined
y
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    y
NameError: name 'y' is not defined
```

```
x=[1,2,3,4,5,6]
del x[1:4]  #删除指定元素
```

```
x
[1, 5, 6]
```

```
#上述方法等同于下面的切片方法
x=[1,2,3,4,5,6]
x[1:4]=[ ]
```

```
x=[1,2,3,4,5,6]
del x[::2]
```

```
x
[2, 4, 6]
```

```
#这样切片方法是行不通的
x=[1,2,3,4,5,6]
x[::2]=[ ]
```

```
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    x[::2]=[]
ValueError: attempt to assign sequence of size 0 to extended slice of size 3
```

list( )　将可迭代对象转化成列表

```
list("cym")　#字符串
```

```
['c', 'y', 'm']
```

```
list((1,2,3,4,5))　#元组
```

```
[1, 2, 3, 4, 5]
```

tuple( )　将可迭代对象转化成元组

```
tuple([1,2,3,4,5])
```

```
(1, 2, 3, 4, 5)
```

str( )　将可迭代对象转化成字符串

```
str((1,2,3,4,5))
```

```
'(1, 2, 3, 4, 5)'
```

min( )&max( )　找出最小或最大值

```
s=[1,1,9,6,4]
min(s)　　#可以直接穿参数：min(1,1,9,6,4)
```

```
1
```

```
t="cym is handsome"　#如果传入字符串，则输出最大编码值所对应的字符
max(t)
```

```
'y'
```

```
s=[]　#输入空的对象则会报错
max(s)
```

```
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    max(s)
ValueError: max() arg is an empty sequence
```

```
但是可以设置 default 参数
s=[]
max(s,default="nothing")
```

```
'nothing'
```

len( )&sum( )

```
s=[1,0,9,6,4]
sum(s)
```

```
20
```

sum(s,start=200) #可以指定开始求和位置

```
220
```

sorted( )& reversed( ) 排序

s=[2,0,8,4,1]

sorted(s)

```
[0, 1, 2, 4, 8]    #返回的是全新的列表，原来 s 列表不受影响
```

注：s.sort( )这样 s 列表会被改变

t=['Cym','Apple','Book','Pencil','Water']

sorted(t) #对比每一个字符串的编码值

```
['Apple', 'Book', 'Cym', 'Pencil', 'Water']
```

sorted(t,key=len) #key 参数是指定排序算法的参数，让列表中每个元素先调用 len 函数，对 len 函数返回值进行比较

```
['Cym', 'Book', 'Apple', 'Water', 'Pencil']
```

s=[2,0,1,3,1,4]

reversed(s)

```
<list_reverseiterator object at 0x0000028F7BE5A530>    #返回值是迭代器
```

list(reversed(s))

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

all( ) 判断是否所有元素全为真 & any( ) 判断是否有元素为真

enumerate( )

enumerate() 函数用于返回一个枚举对象，它的功能就是将可迭代对象中的每个元素及从 0 开始的序号共同构成一个二元组的列表。

s=["cym","lsj","dyt","jyt"]

list(enumerate(s))

```
[(0, 'cym'), (1, 'lsj'), (2, 'dyt'), (3, 'jyt')]
```

list(enumerate(s,10)) #可以指定开始序号

```
[(10, 'cym'), (11, 'lsj'), (12, 'dyt'), (13, 'jyt')]
```

应用：使用 enumerate 遍历列表

```
scores = [78,56,43,51,69,82,27,91]
for i,v in enumerate(scores):
    print(i,'th element is ',v)
```

```
0 th element is  78
1 th element is  56
2 th element is  43
3 th element is  51
4 th element is  69
5 th element is  82
6 th element is  27
7 th element is  91
```

zip( )

zip() 函数用于创建一个聚合多个可迭代对象的迭代器。
它会将作为参数传入的每个可迭代对象的每个元素依次组合成
元组，即第 i 个元组包含来自每个参数的第 i 个元素。

```
x=[1,2,3]
y=[4,5,6]
z=[7,8,9]
s=zip(x,y,z)
list(s)
```

```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

```
x=[1,2,3]
y=[4,5,6]
z="Charming"
s=zip(x,y,z)
list(s)
```

```
[(1, 4, 'C'), (2, 5, 'h'), (3, 6, 'a')]   #若三个长度不等，则以最短的为准
```

若要将所有元素都包含，就要导入模块 itertools

```
x=[1,2,3]
y=[4,5,6]
z="Charming"
import itertools
s= itertools.zip_longest(x,y,z)
list(s)
```

```
[(1, 4, 'C'), (2, 5, 'h'), (3, 6, 'a'), (None, None, 'r'), (None, None, 'm'), (None, None, 'i'), (None, None, 'n'), (None, None, 'g')]
```

map( )

map() 函数会根据提供的函数对指定的可迭代对象的每个元素
进行运算，并将返回运算结果的迭代器。

```
m=map(ord,"CyM")
```

函数（这里的函数是将字符串变成编码值）　要处理的字符串

```
list(m)
```

```
[67, 121, 77]
```

***函数有多参数也可以**

```
s=map(pow,[2,6,4],[10,3,4])
list(s)
```

```
[1024, 216, 256]
```

应用：遍历列表

```
from math import sqrt
s=[1,2,4,10.0]
ls=map(sqrt,s)
t=list(ls)
t
```

```
[1.0, 1.4142135623730951, 2.0, 3.1622776601683795]
```

更多关于 map 函数内容链接到函数章节的 lambda 关键字

filter( )

filter() 函数会根据提供的函数对指定的可迭代对象的每个元素进行运算，并将运算结果为真的元素，以迭代器的形式返回。

```
list(filter(str.islower,"CharMinG"))
```

用于过滤的函数　　要处理的字符串

```
['h', 'a', 'r', 'i', 'n']
```

注：可迭代对象可以多次使用，但迭代器只能使用一次

更多关于 filter 函数内容链接到函数章节的 lambda 关键字

iter( )　将可迭代对象变成迭代器

```
x=[1,2,3,4,5]
y=iter(x)
type(x)
```

```
<class 'list'>
```

```
type(y)
```

```
<class 'list_iterator'>
```

next( ) 逐个将迭代器中的元素提取出来

```
x=[1,2,3,4]
y=iter(x)
next(y)
```

```
1
```

```
next(y)
```

```
2
```

```
next(y)
```

```
3
```

```
next(y)
```

```
4
```

```
next(y)
```

```
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    next(y)
StopIteration
```

要让计算机不抛出异常，如下操作

```
z=iter(x)
next(z,"NULL")
```

```
1
```

```
next(z,"NULL")
```

```
2
```

```
next(z,"NULL")
```

```
3
```

```
next(z,"NULL")
```

```
4
```

next(z,"NULL")

```
'NULL'
```

字典

最简单举例:

s={"吕布":"口口布","关羽":"关习习"}

键 值

type(s)

```
<class 'dict'>
```

s["吕布"]

```
'口口布'
```

s["刘备"]="刘宝贝"

```
s
{'吕布': '口口布', '关羽': '关习习', '刘备': '刘宝贝'}
```

创建字典

a={'吕布': '口口布', '关羽': '关习习', '刘备': '刘宝贝'}
b=dict(吕布= '口口布', 关羽= '关习习', 刘备='刘宝贝')
c=dict([('吕布','口口布'),('关羽', '关习习'),('刘备', '刘宝贝')])
d=dict({'吕布': '口口布', '关羽': '关习习', '刘备': '刘宝贝'})
e=dict({'吕布': '口口布', '关羽': '关习习',},刘备='刘宝贝')
f=dict(zip(['吕布','关羽','刘备'],['口口布','关习习','刘宝贝']))
a==b==c==d==e==f

```
True   #六种创建方法相同
```

增

.fromkeys( ) 初始化字典

s=dict.fromkeys("Charming",520)

```
s
{'C': 520, 'h': 520, 'a': 520, 'r': 520, 'm': 520, 'i': 520, 'n': 520, 'g': 520}
```

s['a']=1314 #可以修改其中任意键的值

```
s
{'C': 520, 'h': 520, 'a': 1314, 'r': 520, 'm': 520, 'i': 520, 'n': 520, 'g': 520}
```

s['c']=250 #如果找不到对应的键，就会增加一个新的键值

```
s
{'C': 520, 'h': 520, 'a': 1314, 'r': 520, 'm': 520, 'i': 520, 'n': 520, 'g': 520, 'c': 250}
```

删

s.pop('c') #删除键值对

```
250
```

s.pop('v','NOTHING') #如果删除的键值对不存在，可以用 default 参数是程序不报错

```
'NOTHING'
```

.popitem( )　删除字典最后一个键值对（Python3.7 之后，若之前则随机删除一个）

```
s.popitem()
```

```
('g', 520)
```

del 关键字　删除指定键值对

```
del s['n']
```

```
s
{'C': 520, 'h': 520, 'a': 1314, 'r': 520, 'm': 520, 'i': 520}
```

```
del s　#也可以删除整个字典
```

```
s
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    s
NameError: name 's' is not defined
```

改

.update( )　多个替换

```
s=dict.fromkeys("Charming")
s.update({'h':100,'v':101})
```

```
s
{'C': None, 'h': 100, 'a': None, 'r': None, 'm': None, 'i': None, 'n': None, 'g': None, 'v': 101}
```

查

.get( )

```
s=dict.fromkeys("Charming")
s.get('c','没有哦')　#如果没有则返回 default 参数的值
```

```
'没有哦'
```

.setdefault( )　查找键，如果在返回对应值，不在指定新的值

```
s.setdefault('c','code')
```

```
'code'
```

```
s
```

```
{'C': None, 'h': None, 'a': None, 'r': None, 'm': None, 'i': None, 'n': None, 'g': None, 'c': 'code'}
```

.items( )键值对　& .keys( )键　& .values( )值

```
s={'C': 520, 'h': 100, 'a': 130, 'r': 123, 'm': 587, 'i': 520, 'n': 520, 'g': 520}
i=s.items()
k=s.keys()
v=s.values()
i
```

```
dict_items([('C', 520), ('h', 100), ('a', 130), ('r', 123), ('m', 587), ('i', 520), ('n', 520), ('g', 520)])
```

```
k
```

```
dict_keys(['C', 'h', 'a', 'r', 'm', 'i', 'n', 'g'])
```

v

```
dict_values([520, 100, 130, 123, 587, 520, 520, 520])
```

遍历

```
staff = {'Jane':19,'Martin':21,'Audrey':18,'Hellen':20,'John':18,'Michael':20}
for k in staff:
    print('name =',k,', age =',staff[k])
```

```
name = Jane , age = 19
name = Martin , age = 21
name = Audrey , age = 18
name = Hellen , age = 20
name = John , age = 18
name = Michael , age = 20
```

以下是之前类似的内容：

```
>>> e = d.copy()
>>> e
{'F': '70', 'i': 105, 's': 115, 'h': 104, 'C': '67'}
>>> len(d)
5
>>> 'C' in d
True
>>> 'c' not in d
True
>>> list(d)
['F', 'i', 's', 'h', 'C']
>>> list(d.values())
['70', 105, 115, 104, '67']
>>> e = iter(d)
>>> next(e)
'F'
>>> next(e)
'i'
>>> next(e)
's'
>>> next(e)
'h'
>>> next(e)
'C'
>>> next(e)
Traceback (most recent call last):
  File "<pyshell#38>", line 1, in <module>
    next(e)
StopIteration
```

浅拷贝

长度

是否存在

转变成列表

转变成迭代器

嵌套

```
d={"吕布" : {"Chinese":60,"math":80,"English":20},
          "关羽" : {"Chinese":90,"math":60,"English":100}}
d["吕布"]["math"]   #进行两次索引
```

```
80
```

其中也可以嵌套一个序列（下面以列表为例）

```
d={"吕布" :[60,80,20],"关羽" : [90,60,100]}
d['吕布'][1]
```

```
80
```

字典推导式

```
s={'C': 520, 'h': 100, 'a': 130, 'r': 123, 'm': 587, 'i': 520, 'n': 520, 'g': 520}
```

b={v:k for k,v in s.items() if v<150}

```
b
{100: 'h', 130: 'a', 123: 'r'}
```

c={s:ord(s) for s in "cym"}　#求编码值

```
c
{'c': 99, 'y': 121, 'm': 109}
```

集合　set 可变集合 & frozenset 不可变集合

```
>>> type({})
<class 'dict'>
>>> type({"one"})
<class 'set'>
>>> type({"one":1})
<class 'dict'>
```

集合与字典紧密相关

```
>>> {"FishC", "Python"}
{'FishC', 'Python'}
>>> {s for s in "FishC"}
{'h', 'i', 'F', 'C', 's'}
>>> set("FishC")
{'h', 'i', 'F', 'C', 's'}
>>> s = set("FishC")
>>> s[0]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    s[0]
TypeError: 'set' object is not subscriptable
```

集合的无序性

```
>>> 'C' in s
True
>>> 'c' not in s
True
>>> for each in s:
        print(each)


h
i
F
C
s
```

集合的唯一性

```
>>> set([1, 1, 2, 3, 5])
{1, 2, 3, 5}
>>> s = [1, 1, 2, 3, 5]
>>> len(s) == len(set(s))
False
```

. isdisjoint( )　判断两集合是否不相关

s=set("CYM")

s.isdisjoint(set("Charming"))

```
False　#有共同元素 C，所以相关
```

s.isdisjoint(set("Python"))

```
True　#无共同元素
```

.issubset( )　判断是否为子集

s.issubset("www.CYM.com")

```
True
```

.issuperset( )　判断是否为超集

对于两个集合 A、B，如果集合 B 中任意一个元素都是集合 A 中的元素，我们就说这两个集合有包含关系，称集合 A 为集合 B 的超集

s.issuperset("CM")

```
True
```

.union( )　求并集

s.union({1,2,3})

```
{'M', 'Y', 1, 2, 3, 'C'}
```

.intersection( )　求交集

s.intersection("CM")

```
{'C', 'M'}
```

.difference( )　求差集

对于两个集合 A、B，由所有属于集合 A 且不属于集合 B 的元素所组成的集合，叫做集合 A 与集合 B 的差集

s.difference("CM")

```
{'Y'}
```

注：以上三个都支持多参数

.symmetric_difference( )　求对称差集（不支持多参数）

对于两个集合 A、B，先排除集合 A 与集合 B 的所有共同元素，由剩余的元素组成的集合，叫做集合 A 与集合 B 的对称差集

s.symmetric_difference("Charming")

```
{'m', 'n', 'a', 'M', 'Y', 'h', 'i', 'r', 'g'}
```

以上集合类型都可以用符号求得（运算符两边必须都是集合）

符号	集合名称
<	真子集
<=	子集
>	真超集
>=	超集
|	并集
&	交集
-	差集
^	对称差集

```
>>> s=set("FishC")
```

```
>>> s <= set("FishC")
True
>>> s < set("FishC")
False
>>> s < set("FishC.com.cn")
True
>>> s > set("FishC")
False
>>> s >= set("FishC")
True
>>> s | {1, 2, 3} | set("Python")
{1, 2, 'h', 'i', 3, 'o', 'n', 'F', 'C', 'P', 't', 'y', 's'}
>>> s & set("Php") & set("Python")
{'h'}
>>> s - set("Php") - set("Python")
{'F', 'i', 'C', 's'}
>>> s ^ set("Python")
{'o', 'F', 'P', 't', 'i', 'n', 'C', 'y', 's'}
```

.update( )　增加集合元素

```
s=set("CYM")
s.update([1,2,2],"NB")
s
```

```
{1, 2, 'B', 'C', 'N', 'M', 'Y'}
```

.add( )　添加元素

```
s=set("CYM")
s.add("NB")
s
```

```
{'C', 'Y', 'NB', 'M'}
```

.remove( ) & .discard( )　删除，但二者对没有的元素删除抛出的结果不同

```
s=set("CYM")
s.remove("AB")  #抛出错误
```

```
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    s.remove("AB")
KeyError: 'AB'
```

```
s.discard("AB")  #静默处理
```

.pop( )　随机弹出一个元素

.clear( ) 清空

函数

```
def fuction():
    print("I am very happy!")
    print("We are all very happy!")
fuction()
```

```
I am very happy!
We are all very happy!
```

```
def fuctiontwo():
```

```
def fuctiontwo(name):   #括号里可以有参数
    print(name+'I love you!')
fuctiontwo('My gril friend,')
```

```
My gril friend,I love you!
```

```
def add(num1,num2):   #可以多参数
    sum=num1+num2
    print(sum)
add(2,8)
```

```
10
```

return　返回值

```
def add(num1,num2):
    return num1+num2
print(add(6,9))
```

```
15
```

函数文档

```
def fuctionthree(name):
    '函数定义过程中 name 叫形参'   #函数文档
    #因为它只是一个形式              #注释
    print('传递进来的'+name+'叫做实参')
fuctionthree('Charming')
```

```
传递进来的 Charming 叫做实参
```

```
fuctionthree.__doc__
```

```
'函数定义过程中 name 叫形参'
```

```
help(fuctionthree)
```

```
Help on function fuctionthree in module __main__:

fuctionthree(name)
    函数定义过程中 name 叫形参
```

```
print.__doc__
```

```
"print(value, ..., sep=' ', end='\\n', file=sys.stdout, flush=False)\n\nPrints the values to a stream, or to sys.stdout by default.\nOptional keyword arguments:\nfile:  a file-like object (stream); defaults to the current sys.stdout.\nsep:   string inserted between values, default a space.\nend:   string appended after the last value, default a newline.\nflush: whether to forcibly flush the stream."
```

```
help(print)   #和 print.__doc__打印的内容相同，只是将转义字符打印出来
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:   a file-like object (stream); defaults to the current sys.stdout.
    sep:    string inserted between values, default a space.
    end:    string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

关键字参数

```
def functionfour(name,words):
    print(name+'-->'+words)
functionfour(words="I want to find a grilfriend",name="CYM")
```

```
CYM-->I want to find a grilfriend
```

默认参数

```
def functionfour(name="CYM",words="I want to find a grilfriend"):
    print(name+'-->'+words)
functionfour()  #不加参数时直接嗲用默认参数
```

```
CYM-->I want to find a grilfriend
```

```
functionfour(name="Mr.Wang")  #若有参数，则使用参数
```

```
Mr.Wang-->I want to find a grilfriend
```

收集参数

```
def functionfive(*p):
    print("参数长度为：",len(p))
    print("第三个参数为：",p[2])
functionfive(1,3.14,"CYM",99)
```

```
参数长度为：  4
第三个参数为：  CYM
```

返回值

```
def back():
    return[1,'cym',3.1415]  #可以返回多个值
back()
```

```
[1, 'cym', 3.1415]
```

局部变量和全局变量

```
def discounts(price,rate):
    final_price=price*rate
    return final_price #这里的 final_price 为局部变量，只在 discounts 函数生效
old_price=float(input('请输入原价：'))
rate=float(input('请输入折扣：'))
new_price=discounts(old_price,rate) # old_price，rate，new_price 都是全局变量
print('新价格：',new_price)
```

```
请输入原价：98
请输入折扣：0.65
新价格：  63.7
```

下面试图修改全局变量

```
def discounts(price,rate):
    final_price=price*rate
    old_price=50 #这里会新建一个名字也叫 old_price 的局部变量
    print('修改后的 old_price 的值 1：',old_price)
    return final_price
old_price=float(input('请输入原价：'))
rate=float(input('请输入折扣：'))
new_price=discounts(old_price,rate)
print('修改后的 old_price 的值 2：',old_price)
print('新价格：',new_price)
```

```
请输入原价：98
请输入折扣：0.65
修改后的 old_price 的值 1：  50
修改后的 old_price 的值 2：  98.0
新价格：  63.7
```

global 将局部变量变成全局变量

```
count=5
def f1():
    count=10
    print(count)
f1()
```

```
10
```

```
print(count)
```

```
5
```

下面用 global 关键字将局部变量变成全局变量

```
count=5
def f1():
    global count
    count=10
    print(count)
f1()
```

```
10
```

```
print(count)
```

```
10
```

函数嵌套

```
def fun1():
    print("fun1()正在被调用")
    def fun2():
```

```
        print("fun2()正在被调用") #在函数 fun1()里再定义一个函数，且 fun2()作用域
                                  只在 fun1()里面，外部将无法调用
    fun2()
fun1()
```

```
fun1()正在被调用
fun2()正在被调用
```

闭包

```
def funX(x):
    def funY(y): #这里 funY()函数就是闭
                 包函数
        return x*y
    return funY
i=funX(8)                                  i  #这里 i 的类型还是函数
```

```
<function funX.<locals>.funY at 0x00000282E6C370A0>
```

```
i(5)                                       funX(8)(5) #还可以这么写
```

```
40
```

```
40
```

常见错误：

```
def fun3():
    x=5  #这里的 x 是非全局变量的外部变量，在 fun4()函数中无法修改 x 的值
    def fun4():
        x*=x
        return x
    return fun4()
fun3()
```

```
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    fun3()
  File "<pyshell#29>", line 6, in fun3
    return fun4()
  File "<pyshell#29>", line 4, in fun4
    x*=x
UnboundLocalError: local variable 'x' referenced before assignment
```

解决方案 1：用容器类型存放，因为容器类型不存放在栈里，不会被屏蔽

```
def fun3():
    x=[5]
    def fun4():
        x[0]*=x[0]
        return x[0]
    return fun4()
fun3()
```

```
25
```

解决方案 2：用关键字 nonlocal

```
def fun3():
    x=5
```

```
    def fun4():
        nonlocal x #和 global 关键字使用差不多
        x*=x
        return x
    return fun4()
fun3()
```

```
25
```

匿名函数

lambda 关键字

```
c=lambda x:2*x+1
```

要传入的参数　执行的表达式

```
c(5)
```

```
11
```

filter 函数

*前面内容点此链接

```
list(filter(lambda x:x%3,range(10)))
```

```
[1, 2, 4, 5, 7, 8]
```

等价于：

```
def tri(x):
    return x%3
temp=range(10)
show=filter(tri,temp)
list(show)
```

```
[1, 2, 4, 5, 7, 8]
```

```
def de(x):
    return 2*x+1
ds(5)
```

```
11
```

map 函数 映射

*前面内容点此链接

```
list(map(lambda x:x*2,range(10)))
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

递归

```
def r():
    return r()
r()  #一直进行
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    r()
  File "<pyshell#5>", line 2, in r
    return r()
  File "<pyshell#5>", line 2, in r
    return r()
  File "<pyshell#5>", line 2, in r
```

```
    return r()
  [Previous line repeated 1022 more times]
RecursionError: maximum recursion depth exceeded
```

可以设置递归深度（默认 100 层）

```
import sys
sys.setrecursionlimit(1000) #设置 1000 层深度
```

eg1:求阶乘

```
num=int(input("请输入求的阶乘："))
def s(a):
    if a==1:
        return 1
    else:
        return a*s(a-1)
result=s(num)
print("%d 的阶乘是：%d"%(num,result))
```

```
请输入求的阶乘：10
10 的阶乘是：3628800
```

eg2：斐波那契数列

迭代算法：

```
def fab(n):
    n1=1
    n2=1
    n3=1
    if n<1:
        print("输入错误")
        return -1
    while (n-2)>0:
        n3=n1+n2
        n1=n2
        n2=n3
        n-=1
    return n3
result=fab(20)
if result!=-1:
    print('%d 兔子出生'%result)
```

```
6765 兔子出生
```

递归算法：

```
def fab(n):
    if n<1:
        print("输入错误")
        return -1
    if n==1 or n==2:
        return 1
    else:
        return
fab(n-1)+fab(n-2)
result=fab(20)
if result != -1:
    print('%d 兔子出生'%result)
```

```
6765 兔子出生
```

eg3：汉诺塔

```
def hanoi(n,x,y,z):
    if n==1:
        print(x,"-->",z)
    else:
```

```
        hanoi(n-1,x,y,z)
        print(x,'-->',z)
        hanoi(n-1,y,x,z)
n=int(input('请输入层数：'))
hanoi(n,'X','Y','Z')
```

异常

异常类型

AssertionError	断言语句（assert）失败
AttributeError	尝试访问未知的对象属性
EOFError	用户输入文件末尾标志 EOF（Ctrl+d）
FloatingPointError	浮点计算错误
GeneratorExit	generator.close()方法被调用的时候
sertImportError	导入模块失败的时候
IndexError	索引超出序列的范围
KeyError	字典中查找一个不存在的关键字
KeyboardInterrupt	用户输入中断键（Ctrl+c）
MemoryError	内存溢出（可通过删除对象释放内存）
NameError	尝试访问一个不存在的变量
NotImplementedError	尚未实现的方法
OSError	操作系统产生的异常（例如打开一个不存在的文件）
OverflowError	数值运算超出最大限制
ReferenceError	弱引用（weak reference）试图访问一个已经被垃圾回收机制回收了的对象
RuntimeError	一般的运行时错误
StopIteration	迭代器没有更多的值
SyntaxError	Python 的语法错误
IndentationError	缩进错误
TabError	Tab 和空格混合使用
SystemError	Python 编译器系统错误
SystemExit	Python 编译器进程被关闭
TypeError	不同类型间的无效操作
UnboundLocalError	访问一个未初始化的本地变量（NameError 的子类）

UnicodeError	Unicode 相关的错误（ValueError 的子类）
UnicodeEncodeError	Unicode 编码时的错误（UnicodeError 的子类）
UnicodeDecodeError	Unicode 解码时的错误（UnicodeError 的子类）
UnicodeTranslateError	Unicode 转换时的错误（UnicodeError 的子类）
ValueError	传入无效的参数
ZeroDivisionError	除数为零

举例说明：

1. AssertionError 断言语句（assert）失败

```
s=['cym is a cute boy']
assert len(s)>0   #assert 函数当表达式为假的时候报错
s.pop()
```

```
'cym is a cute boy'
```

```
assert len(s)>0
```

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    assert len(s)>0
AssertionError
```

2. AttributeError 尝试访问未知的对象属性

```
s.cym
```

```
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s.cym
AttributeError: 'list' object has no attribute 'cym'
```

3. IndexError 索引超出序列的范围

```
s=[1,2,3]
s[3]
```

```
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    s[3]
IndexError: list index out of range
```

4. KeyError 字典中查找一个不存在的关键字

```
s={'one':1,'two':2,'three':3}
s['four']
```

```
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    s['four']
KeyError: 'four'
```

异常检测

①try-exception

```
try:
    f=open('文件 101.txt')
    print(f.read())
    f.close()
except OSError:
    print('无此文件！！')
```

```
无此文件！！
```

```
try:
    f=open('文件 101.txt')
    print(f.read())
    f.close()
except OSError as reason: #reason 参数，可以输出具体错误原因
    print('无此文件！！ \n 错误原因：'+str(reason))
```

```
无此文件！！
错误原因：[Errno 2] No such file or directory: '文件 101.txt'
```

```
try:
    sum=1+'1'
    f=open('文件 101.txt')
    print(f.read())
    f.close()
except OSError as reason:
    print('无此文件！！ \n 错误原因：'+str(reason))
except TypeError as reason:
    print('类型出错！！ \n 错误原因：'+str(reason)) #报第一个错误，其他语句不会运行
```

```
类型出错！！
错误原因：unsupported operand type(s) for +: 'int' and 'str'
```

```
try:
    sum=1+'1'
    f=open('文件 101.txt')
    print(f.read())
    f.close()
except :
    print('出错啦！！') #任何错误都会报错，不推荐使用
```

```
出错啦！！
```

②try-finally

```
try:
    f=open('文件 101.txt','w')
    print(f.write('我存在了！'))
    sum=1+'1'
except (OSError,TypeError):    #出现两个中任意一个错误都会报错
    print('出错啦！！')
finally:                       #如果无 finally，文件不会关闭，输入类容不会保存
```

```
    f.close()
```

```
5
出错啦！！
```

引发异常

raise

```
raise ZeroDivisionError('分子不能为 0')
```

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    raise ZeroDivisionError('分子不能为 0')
ZeroDivisionError: 分子不能为 0
```

else & with 语句

else eg1：求最大因数

```
def maxfactor(num):
    count=num//2
    while count>1:
        if num % count==0:
            print("%d 最大因数为%d" % (num,count))
            break
        count-=1
    else:
        print("%d 是素数" % num)
num=int(input('请输入一个数：'))
maxfactor(num)
```

测试 1：

```
请输入一个数：36
36 最大因数为 18
```

测试 2：

```
请输入一个数：17
17 是素数
```

else eg2:异常

```
num = input('请输入：')
try:
    print(int(num))
except ValueError as reason:
    print('出错啦：'+str(reason))
else:
    print('没有异常')
```

测试 1：

```
请输入：123
123
```

```
没有异常
```

测试 2：

```
请输入：abc
出错啦：invalid literal for int() with base 10: 'abc'
```

with eg：文件打开

```
try:
    with open('data.txt','w') as f:
      for eachline in f:
          print(eachline)
except ValueError as reason:
    print('出错啦：'+str(reason))
```

等价

```
try:
    f=open('data.txt','w')
    for eachline in f:
        print(eachline)
except ValueError as reason:
    print('出错啦：'+str(reason))
finally:
    f.close()
```

```
出错啦：not readable
```

```
出错啦：not readable
```

图形用户界面编程：EasyGui

教学文件：https://www.cnblogs.com/hcxy2007107708/articles/10091466.html

类&对象

eg1：封装

```
class Charming:
    colour = 'yellow'
    weight = 50
    legs = 2
    shell = False
    def study(self):
        print("我要卷 si 你们")
    def play(self):
        print("Genshin Impact is funny")
    def eat(self):
        print("I like popcorn")
    def sleep(self):
        print("Go to bed~~")
cym=Charming()

cym.play()
```

```
Genshin Impact is funny
```

```
cym.legs
```

```
2
```

eg2：继承

```
class Mylist(list):     #继承 list 的属性和方法
    pass
list2=Mylist()
list2.append(9)
list2.append(3)
list2.append(6)
```

```
list2
[9, 3, 6]
```

eg3：多态

```
class A:
    def fun(self):
        print("I am A")
class B:
    def fun(self):
        print("I am B")
a=A()
b=B()
```

```
a.fun()
I am A
b.fun()
I am B
```

面向对象编程

self 相关解释说明

```
class Ball:
    def setName(self,name):
        self.name=name
    def kick(self):
        print("My name is %s, oh shit, don't kick me!" % self.name)
a=Ball()
a.setName('Ball A')
b=Ball()
b.setName('Ball B')
c=Ball()
c.setName('Potato')
a.kick()
```

```
My name is Ball A, oh shit, don't kick me!
```

```
c.kick()
```

```
My name is Potato, oh shit, don't kick me!
```

__init__(self)

```
class Ball:
    def __init__(self,name):
        self.name=name
    def kick(self):
        print("My name is %s, oh shit, don't kick me!" % self.name)
b=Ball('Potato')
b.kick()
```

```
My name is Potato, oh shit, don't kick me!
```

公有&私有

公有：

```
My name is Potato, oh shit, don't kick me!
class Person:
    name='Charming'
p=Person()
p.name
```

```
'Charming'
```

私有：

```
class Person:
    __name='Charming' #私有数据无法在外部访问
p=Person()
p.__name
```

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    p.__name
AttributeError: 'Person' object has no attribute '__name'
```

若想访问私有数据，则要从内部访问

```
class Person:
    __name='Charming'
    def getName(self):
        return self.__name
p=Person()
p.getName()
```

```
'Charming'
```

当然直接如下访问也可以：

```
class Person:
    __name='Charming'
p=Person()
p._Person__name
```

```
'Charming'
```

继承

```
class Parent:
    def hello(self):
        print("正在调用父类方法~")
class Child(Parent):
    pass
p=Parent()
p.hello()
```

```
正在调用父类方法~
```

```
c=Child()
c.hello()
```
```
正在调用父类方法~
```

注意：如果自类中定义与父类同名的方法或属性，则会自动覆盖父类对应的方法或属性

```
class Parent:
    def hello(self):
        print("正在调用父类方法~")
class Child(Parent):
    def hello(self):
        print("正在调用子类方法~")
c=Child()
c.hello()
```
```
正在调用子类方法~
```
```
p=Parent()
p.hello()
```
```
正在调用父类方法~
```

eg：小鱼游动

```
import random as r
class Fish:
    def __init__(self):
        self.x=r.randint(0,10)
        self.y=r.randint(0,10)
    def move(self):
        self.x-=1
        print("我的位置是：",self.x,self.y)
class Goldfish(Fish):
    pass
class Carp(Fish):
    pass
class Salmon(Fish):
    pass
class Shark(Fish):
    def __init__(self):
        self.hungry=True
    def eat(self):
        if self.hungry:
            print("I want to eat fish~~")
            self.hungry= False
        else:
            print("The fish is tasty!")
fish=Fish()
```

```
fish.move()
我的位置是:  8 4
fish.move()
我的位置是:  7 4
goldfish=Goldfish()
goldfish.move()
我的位置是:  6 3
goldfish.move()
我的位置是:  5 3
shark=Shark()
shark.eat()
I want to eat fish~~
shark.eat()
The fish is tasty!
shark.move()
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    shark.move()
  File "C:/Users/lenovo/AppData/Local/Programs/Python/Python310/9.py", line 7, in move
    self.x-=1
AttributeError: 'Shark' object has no attribute 'x'
```

错误产生的原因：子类将父类覆盖

下面介绍解决方案

①调用未绑定的父类方法

```
class Shark(Fish):
    def __init__(self):
        Fish.__init__(self)
        self.hungry=True
    def eat(self):
        if self.hungry:
            print("I want to eat fish~~")
            self.hungry= False
        else:
            print("The fish is tasty!")
```

②super 函数

```
class Shark(Fish):
    def __init__(self):
        super().__init__(self)
        self.hungry=True
    def eat(self):
        if self.hungry:
            print("I want to eat fish~~")
```

```
            self.hungry= False
        else:
            print("The fish is tasty!")
```

多重继承

```
class Date1:
    def fan1(self):
        print("12 月 14 日")
class Date2:
    def fan2(self):
        print("11 月 14 日")
class C(Date1,Date2):
    pass
c=C()
c.fan1()
```

```
12 月 14 日
```

```
c.fan2()
```

```
11 月 14 日
```

组合

将几个横向关系的类放在一起叫做组合

```
class Turtle:
    def __init__(self,x):
        self.num=x
class Fish:
    def __init__(self,x):
        self.num=x
class Pool:
    def __init__(self,x,y):
        self.turtle=Turtle(x)
        self.fish=Fish(y)
    def print_num(self):
        print("There are %d turtles, %d fishes" % (self.turtle.num,self.fish.num))
pool=Pool(2,5)
pool.print_num()
```

```
There are 2 turtles, 5 fishes
```

issubclass( )　若第一个参数是第二个参数子类就返回 Ture

注：一个类被认为是其自身的子类

```
class A:
    pass
class B(A):
    pass
issubclass(B,A)
```

```
True
```

```
issubclass(B,B)
```

```
True
```

```
issubclass(B,object) #object 是所有类的基类
```

```
True
```

isinstance( ) 检查一个实例对象是否属于一个类

```
class A:
    pass
class B(A):
    pass
class C:
    pass
b1=B()
isinstance(b1,B)
```

```
True
```

```
isinstance(b1,A)
```

```
True
```

```
isinstance(b1,(A,B,C))
```

```
True
```

hasattr( ) 测试对象是否有指定属性

```
class D:
    def __init__(self,x=0):
        self.x=x
d1=D()
hasattr(d1,'x')
```

```
True
```

getattr( ) 返回对象的属性值

```
getattr(d1,'x')
```

```
0
```

```
getattr(d1,'y','您访问的参数不存在！！') #可以设置第三个参数
```

```
'您访问的参数不存在！！'
```

setattr( ) 设定指定属性的值

```
setattr(d1,'y','Charming')
getattr(d1,'y','您访问的参数不存在！！')
```

```
'Charming'
```

delattr( ) 删除指定属性的值

property( ) 通过属性设置属性,三个参数分别是获得，设置和删除属性的方法

```
class D:
    def __init__(self,size=10):
        self.size=size
    def getSize(self):
        return self.size
```

```
    def setSize(self,value):
        self.size=value
    def delSize(self):
        del self.size
    x=property(getSize,setSize,delSize)
d1=D()
d1.x #等同于 d1.size
```

```
10
```

```
d1.x=22
d1.x
```

```
22
```

```
del d1.x
d1.size
```

```
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    d1.size
AttributeError: 'D' object has no attribute 'size'
```

魔法方法

构造和析构

__init__( )

```
class Rectangle:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def getPeri(self):
        return (self.x+self.y)*2
    def getArea(self):
        return self.x*self.y
r=Rectangle(3,4)
r.getPeri()
```

```
14
```

```
r.getArea()
```

```
12
```

注：__init__( )没有返回值

__new__( )

```
class CapStr(str):
    def __new__(cls,string):
        string = string.upper()
        return str.__new__(cls,string)
```

```
a=CapStr("Cym is hanndsome")
a
```

```
'CYM IS HANNDSOME'
```

__del__( )

```
class S():
    def __init__(self):
        print("我是__init__方法，正在被调用")
    def __del__(self):
        print("我是__del__方法，正在被调用")
s1=S()
```

```
我是__init__方法，正在被调用
```

```
s2=s1
s3=s2
del s3
del s2
del s1
```

```
我是__del__方法，正在被调用
```

注：__del__方法只有当对生成的对象的引用全部删去，才会调用

工厂函数

__add__(self, other)	定义加法的行为：+
__sub__(self, other)	定义减法的行为：-
__mul__(self, other)	定义乘法的行为：*
__truediv__(self, other)	定义真除法的行为：/
__floordiv__(self, other)	定义整数除法的行为：//
__mod__(self, other)	定义取模算法的行为：%
__divmod__(self, other)	定义当被 divmod() 调用时的行为
__pow__(self, other[, modulo])	定义当被 power() 调用或 ** 运算时的行为
__lshift__(self, other)	定义按位左移位的行为：<<
__rshift__(self, other)	定义按位右移位的行为：>>
__and__(self, other)	定义按位与操作的行为：&
__xor__(self, other)	定义按位异或操作的行为：^
__or__(self, other)	定义按位或操作的行为：|

eg1：以__add__和__sub__为例

```
class New_int(int):
    def __add__(self,other):
        return int.__sub__(self,other)
    def __sub__(self,other):
        return int.__add__(self,other)
a=New_int(6)
b=New_int(11)
a+b
```

```
-5
```

eg2:对于反运算

```
class Nint(int):
    def __radd__(self,other):
        return int.__sub__(self,other)
a=Nint(5)
b=Nint(8)
a+b
```

```
13
```

```
5+b
```

```
3
```

魔法方法	含义
	基本的魔法方法
__new__(cls[, ...])	1. __new__ 是在一个对象实例化的时候所调用的第一个方法 2. 它的第一个参数是这个类，其他的参数是用来直接传递给 __init__ 方法 3. __new__ 决定是否要使用该 __init__ 方法，因为 __new__ 可以调用其他类的构造方法或者直接返回别的实例对象来作为本类的实例，如果 __new__ 没有返回实例对象，则 __init__ 不会被调用 4. __new__ 主要是用于继承一个不可变的类型比如一个 tuple 或者 string
__init__(self[, ...])	构造器，当一个实例被创建的时候调用的初始化方法
__del__(self)	析构器，当一个实例被销毁的时候调用的方法
__call__(self[, args...])	允许一个类的实例像函数一样被调用：x(a, b) 调用 x.__call__(a, b)
__len__(self)	定义当被 len() 调用时的行为
__repr__(self)	定义当被 repr() 调用时的行为
__str__(self)	定义当被 str() 调用时的行为
__bytes__(self)	定义当被 bytes() 调用时的行为
__hash__(self)	定义当被 hash() 调用时的行为
__bool__(self)	定义当被 bool() 调用时的行为，应该返回 True 或 False

__format__(self, format_spec)	定义当被 format() 调用时的行为
	有关属性
__getattr__(self, name)	定义当用户试图获取一个不存在的属性时的行为
__getattribute__(self, name)	定义当该类的属性被访问时的行为
__setattr__(self, name, value)	定义当一个属性被设置时的行为
__delattr__(self, name)	定义当一个属性被删除时的行为
__dir__(self)	定义当 dir() 被调用时的行为
__get__(self, instance, owner)	定义当描述符的值被取得时的行为
__set__(self, instance, value)	定义当描述符的值被改变时的行为
__delete__(self, instance)	定义当描述符的值被删除时的行为
	比较操作符
__lt__(self, other)	定义小于号的行为：x < y 调用 x.__lt__(y)
__le__(self, other)	定义小于等于号的行为：x <= y 调用 x.__le__(y)
__eq__(self, other)	定义等于号的行为：x == y 调用 x.__eq__(y)
__ne__(self, other)	定义不等号的行为：x != y 调用 x.__ne__(y)
__gt__(self, other)	定义大于号的行为：x > y 调用 x.__gt__(y)
__ge__(self, other)	定义大于等于号的行为：x >= y 调用 x.__ge__(y)
	算数运算符
__add__(self, other)	定义加法的行为：+
__sub__(self, other)	定义减法的行为：-
__mul__(self, other)	定义乘法的行为：*
__truediv__(self, other)	定义真除法的行为：/
__floordiv__(self, other)	定义整数除法的行为：//
__mod__(self, other)	定义取模算法的行为：%
__divmod__(self, other)	定义当被 divmod() 调用时的行为
__pow__(self, other[, modulo])	定义当被 power() 调用或 ** 运算时的行为
__lshift__(self, other)	定义按位左移位的行为：<<

__rshift__(self, other)	定义按位右移位的行为：>>
__and__(self, other)	定义按位与操作的行为：&
__xor__(self, other)	定义按位异或操作的行为：^
__or__(self, other)	定义按位或操作的行为：|
	反运算
__radd__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rsub__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rmul__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rtruediv__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rfloordiv__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rmod__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rdivmod__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rpow__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rlshift__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rrshift__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rand__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__rxor__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
__ror__(self, other)	（与上方相同，当左操作数不支持相应的操作时被调用）
	增量赋值运算
__iadd__(self, other)	定义赋值加法的行为：+=
__isub__(self, other)	定义赋值减法的行为：-=
__imul__(self, other)	定义赋值乘法的行为：*=
__itruediv__(self, other)	定义赋值真除法的行为：/=
__ifloordiv__(self, other)	定义赋值整数除法的行为：//=
__imod__(self, other)	定义赋值取模算法的行为：%=
__ipow__(self, other[, modulo])	定义赋值幂运算的行为：**=
__ilshift__(self, other)	定义赋值按位左移位的行为：<<=
__irshift__(self, other)	定义赋值按位右移位的行为：>>=
__iand__(self, other)	定义赋值按位与操作的行为：&=
__ixor__(self, other)	定义赋值按位异或操作的行为：^=

__ior__(self, other)	定义赋值按位或操作的行为：|=
	一元操作符
__pos__(self)	定义正号的行为：+x
__neg__(self)	定义负号的行为：-x
__abs__(self)	定义当被 abs() 调用时的行为
__invert__(self)	定义按位求反的行为：~x
	类型转换
__complex__(self)	定义当被 complex() 调用时的行为（需要返回恰当的值）
__int__(self)	定义当被 int() 调用时的行为（需要返回恰当的值）
__float__(self)	定义当被 float() 调用时的行为（需要返回恰当的值）
__round__(self[, n])	定义当被 round() 调用时的行为（需要返回恰当的值）
__index__(self)	1. 当对象是被应用在切片表达式中时，实现整形强制转换 2. 如果你定义了一个可能在切片时用到的定制的数值型,你应该定义 __index__ 3. 如果 __index__ 被定义，则 __int__ 也需要被定义，且返回相同的值
	上下文管理（with 语句）
__enter__(self)	1. 定义当使用 with 语句时的初始化行为 2. __enter__ 的返回值被 with 语句的目标或者 as 后的名字绑定
__exit__(self, exc_type, exc_value, traceback)	1. 定义当一个代码块被执行或者终止后上下文管理器应该做什么 2. 一般被用来处理异常，清除工作或者做一些代码块执行完毕之后的日常工作

简单定制

```
import time as t
class MyTimer():
    def __init__(self):
        self.unit=['年','月','天','小时','分钟','秒']
        self.prompt="未开始计时！ "
        self.lasted=[]
        self.begin=0
        self.end=0
    def __str__(self):
        return self.prompt
    __repr__=__str__
    def __add__(self,other):
        prompt="总共运行了:"
        result=[]
```

```
        for index in range(6):
            result.append(self.lasted[index]+other.lasted[index])
            if result[index]:
                prompt+=(str(result[index])+self.unit[index])
        return prompt
    #开始计时
    def start(self):
        self.begin=t.localtime()
        self.prompt='提示：请先调用 stop()停止计时！'
        print("计时开始...")
    #停止计时
    def stop(self):
        if not self.begin:
            print("提示：请先调用 start()开始计时！")
        else:
            self.end=t.localtime()
            self._calc()
            print("计时结束!")
    #计算运行时间
    def _calc(self):
        self.lasted=[]
        self.prompt="总共运行了"
        for index in range(6):
            self.lasted.append(self.end[index]-self.begin[index])
            if self.lasted[index]:
                self.prompt+=(str(self.lasted[index])+self.unit[index])
    #为下一轮计时进行初始化
        self.begin=0
        self.end=0
```

```
>>> t1 = MyTimer()
>>> t1
未开始计时！
>>> t1.stop()
提示：请先调用 start() 进行计时！
>>> t1.start()
计时开始...
>>> t1.stop()
计时结束！
>>> t1
总共运行了4秒
>>> t2 = MyTimer()
>>> t2.start()
计时开始...
>>> t2.stop()
计时结束！
>>> t2
总共运行了3秒
>>> t1 + t2
'总共运行了7秒'
```

属性访问

```
class C:
    def __getattribute__(self,name):
        print("getattribute")
        return super( ).__getattribute__(name)
    def __getattr__(self,name):
        print("getattr")
    def __setattr__(self,name,value):
        print("setattr")
        super().__setattr__(name,value)
    def __delattr__(self,name):
        print("delattr")
        super().__delattr__(name)
c=C()
c.x
```

```
getattribute
getattr
```

```
c.x=1
```

```
setattr
```

```
c.x
```

```
getattribute
1
```

```
del c.x
```

```
delattr
```

描述符

```
class MD:
    def __get__(self,instance,owner):
        print("getting....",self,instance,owner)
    def __set__(self,instance,value):
        print("setting....",self,instance,value)
    def __delet__(self,instance):
        print("deleting....",self,instance)
class Test:
    x=MD()
test=Test()
test.x
```

```
getting.... <__main__.MD object at 0x000001436902BCA0> <__main__.Test object at 0x00000143690299C0> <class '__main__.Test'>
```

```
test
```

```
<__main__.Test object at 0x00000143690299C0>
```

```
Test
```

```
<class '__main__.Test'>
```

test.x='CYM'

```
setting.... <__main__.MD object at 0x000001436902BCA0> <__main__.Test object at 0x00000143690299C0> CYM
```

定制序列

	容器类型
__len__(self)	定义当被 len() 调用时的行为（返回容器中元素的个数）
__getitem__(self, key)	定义获取容器中指定元素的行为，相当于 self[key]
__setitem__(self, key, value)	定义设置容器中指定元素的行为，相当于 self[key] = value
__delitem__(self, key)	定义删除容器中指定元素的行为，相当于 del self[key]
__iter__(self)	定义当迭代容器中的元素的行为
__reversed__(self)	定义当被 reversed() 调用时的行为
__contains__(self, item)	定义当使用成员测试运算符（in 或 not in）时的行为

第三方库的使用

一.Matplotlib 和 Pandas 库的使用

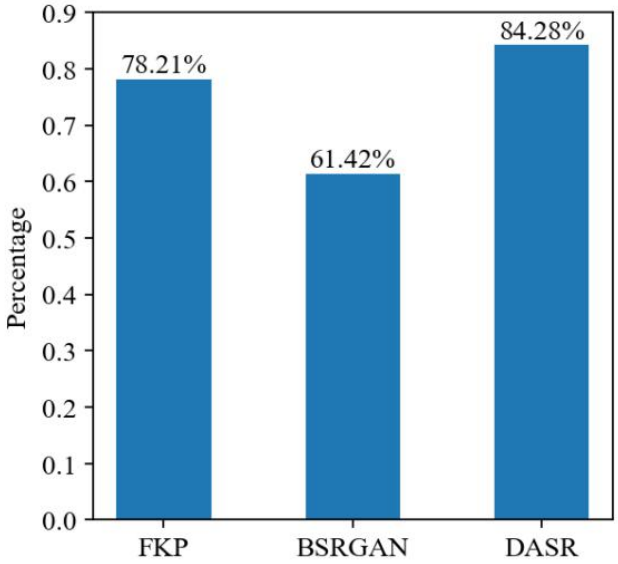
1.Matplotlib.pyplot 基础语法

方法名称	方法作用
plt. figure(figsize, facecolor)	创建一个空白画布，figsize参数可以指定画布大小，像素，单位为英寸。
plt.plot(x,y,ls,lw,lable,color)	根据x，y数据绘制直线、曲线、标记点，ls为线型linestyle，lw为线宽linewidth，lable为标签文本内容，color为颜色。
plt. scatter(x, y, c, marker, label, color)	绘制散点图：x、y为相同长度的序列，c为单个颜色字符或颜色序列，marker为标记的样式，默认的是'o'，label为标签文本内容，color为颜色

plt. bar(x, height, width, bottom)	绘制条形图
plt.title(string)	在当前图形中添加标题，可以指定标题的名称、位置、颜色、字体大小等参数。
plt.xlabel(string)	在当前图形中添加x轴名称，可以指定位置、颜色、字体大小等参数。
plt.ylabel(string)	在当前图形中添加y轴名称，可以指定位置、颜色、字体大小等参数。
plt.xlim(xmin,xmax)	指定当前图形x轴的范围，只能确定一个数值区间，而无法使用字符串标识。
plt.ylim(ymin,ymax)	指定当前图形y轴的范围，只能确定一个数值区间，而无法使用字符串标识。
lt.legend()	指定当前图形的图例，可以指定图例的大小、位置、标签。
plt.show()	在本机显示图形。

例 1:

```
import matplotlib.pyplot as plt
algorithm = ["FKP", "BSRGAN", "DASR"]
results = [0.7821, 0.6142, 0.8428]  # 创建画布，指定画布的宽高，以及像素点的数量
plt.figure('label',figsize=(4,4), dpi = 160)
plt.bar(algorithm, results, alpha=1, width=0.5)  # width 用于控制条形图之间的距离
for x,y in zip(algorithm, results):  # 将数据显示在柱状图上
    plt.text(x, y, '%.2f%%' % (y*100), ha='center', va='bottom', fontdict={'family':'Times New Roman', 'size':12})
plt.ylabel('Percentage', fontdict={'family':'Times New Roman', 'size':12}) #设置纵坐标轴label
plt.ylim([0, 0.9]) # 设置纵坐标坐标轴取值
```



2.Pandas 基础语法

①Series 是一维标记数组。它可以容纳任何类型的数据。

```
mySeries = pd.Series([3,-5,7,4], index=['a','b','c','d'])
```

②type(mySeries)dataframe 是一个二维数据结构，它包含列。

```
data = {'Country' : ['Belgium', 'India', 'Brazil' ], 'Capital': ['Brussels', 'New Delhi', 'Brassilia'],
'Population': [1234,1234,1234]}
datas = pd.DataFrame(data, columns=['Country','Capital','Population'])
print(type(data))
print(type(datas))
```

③读取数据

```
df = pd.read_csv('data.csv')
df = pd.read_excel('filename.xlsx')
```

④统计数据常用函数

方法名称	方法作用
df.info()	函数提供有关数据信息:
df.shape	显示数据的行数和列数
df.index	显示找到的索引总数
df.columns	给出数据框的所有列
df.count	给出每一列中有多少数据
len(df)	计算数据框的长度
df.min()	给出每一列中的最小值
df.max()	给出每一列中的最大值
df.sum()	给出每一列中的求和

例：使用 pandas 遍历 csv 文件，csv 文件中数据如下：

```
import pandas as pd
df = pd.read_csv("./data.scv") #  读取 csv 文件中数据
for index,row in df.iterrows(): #  遍历读取的 csv 文件数据
    print("name=",row["name"], "age=",row["age"],"score=",row["score"])
```

```
name= jay age= 21 score= 90
name= zhangsan age= 21 score= 65
```