Table of Contents

In [3]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
import sys
sys.path    #查看Python的搜索路径
```

1 函数

函数分类

内置函数 : 可以直接用函数名调用,如len(),type()等

模块函数 : 通过模块名进行调用,如math.sin()等 (前提是先导入(import)第三方模块)

自定义函数 : 按照用户需求随用随定义

1.1 内置函数

1.1.1 数据类型相关的函数

int() , bool() , float() , str() , list() , set() , tuple() ,dict() ,type()

1.1.2 逻辑判断相关的函数

all()

any()

1.1.2.1 all()函数

如果bool（x）对于可迭代的所有值x为True，则返回True。如果iterable为空，则返回True

0和False都为假

这里面说的可迭代指的是可以用for循环进行遍历的

In [4]:
```
all([4,5,3,0])  #0代表假,只要有一个为假则返回假
```

Out[4]: False

In [5]:
```
all([4,5,3,6])
```

Out[5]: True

In [6]:
```
all([])
```

Out[6]: True

In [7]:
```
all((2,3,4,5))
```

Out[7]: True

In [8]:
```
all({1,2,3,False})
```

Out[8]: False

1.1.2.2 any()函数

如果bool（x）对于可迭代的任何x为True，则返回True。如果iterable为空，则返回False。

In [13]:
```
any([4,5,6])  #都为真时,当然返回真
```

Out[13]: True

In [9]:
```
any([2,0,False])  #只要有一个为真即为真
```

Out[9]: True

In [10]:
```
any((0,False))  #都为假的时候则为假
```

Out[10]: False

In [14]:
```
any([0,0,False])
```

Out[14]: False

In [12]:
```
any({}) #元素为空,返回假
```

Out[12]: False

1.1.3 数学相关的函数

sum() , abs() , min() , max() , pow() , round() , len() , range() ,divmod() 同时取得商和余数

1.1.3.1 sum()函数

In [16]:
```
sum([1,2,3])
```

Out[16]: 6

In [17]:
```
sum([1,2,3],5)  #第二个参数是指从几开始进行累加,不设置默认从0开始加
```

Out[17]: 11

In [67]:
```
sum(1,2,3)  #会报错,因为sum只有两个参数,所以第一个参数需要用列表或元组或集合等的形式进行括起来
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/29046587.py in <module>
----> 1 sum(1,2,3)  #会报错,因为sum只有两个参数,所以第一个参数需要用列表或元组或集合等的形式进行括起来

TypeError: sum() takes at most 2 arguments (3 given)
```

In [65]:
```
sum((1,2,3),8)
```

Out[65]: 14

In [66]:
```
sum({3,4,5},8)
```

Out[66]: 20

1.1.3.2 abs()函数

In [18]:
```
abs(9)
```

Out[18]: 9

In [19]:
```
abs(-1)
```

Out[19]: 1

1.1.3.3 min()和max()

In [20]:
```
min(1,-1,3,7,-5,4,2)
```

Out[20]: -5

In [22]:
```
min([2,-1,4,-2])
```

Out[22]: -2

In [23]:
```
max(3,7,44,2,-8,35)
```

Out[23]: 44

In [24]:
```
max([98,37,102,-56,4])
```

Out[24]: 102

In [25]:
```
max((2,7,9,1))
```

Out[25]: 9

1.1.3.4 pow()函数

In [26]:
```
pow(2,3)  #pow进行幂的运算,第一个参数为底数,第二个参数为指数,pow(2,3)=2**3=2的3次方
```

Out[26]: 8

In [69]:
```
2**3
```

Out[69]: 8

In [27]:
```
pow(5,2)
```

Out[27]: 25

In [28]:
```
pow(5,2,4)  #第三个参数是除数,最后输出为取余  pow(5,2,4)=5**2%4=5的2次方的结果除以4后取余数
```

Out[28]: 1

1.1.3.5 round()函数

In [40]:
```
round(3.6)  #只有一位小数的时候,大于点5的时候,进位 ,即>0.5时按照四舍五入,但是=3.5时特殊,下面有例子
```

Out[40]: 4

In [37]:
```
round(3.4)  #只有一位小数的时候,小于点5的时候,退位,即四舍
```

Out[37]: 3

In [29]:
```
round(3.14159)   #不只有一位小数的时候,而且不指定第二个参数的值时,进行四舍五入取整
```

Out[29]: 3

In [30]:
```
round(3.56789)
```

Out[30]: 4

In [44]:
```
round(3.14158,2)  #保留两位小数,四舍五入
```

Out[44]: 3.14

In [32]:
```
round(3.14678,2)
```

Out[32]: 3.15

In [34]:
```
round(3.14678123,4)  #保留四位小数,四舍五入
#第二个参数是几,就保留几位小数,四舍五入
```

Out[34]: 3.1468

In [38]:
```
round(3.5)  #只有一位小数,且不指定第二个参数的时候,取距离原数据最近的偶数
```

Out[38]: 4

In [35]:
```
round(2.5)   #只有一位小数,且不指定第二个参数的时候,取距离原数据最近的偶数
```

Out[35]: 2

In [45]:
```
round(1.5)
```

Out[45]: 2

In [46]:
```
round(165.535,-1)
#精度可以取负数,代表精确位数在小数点左侧,
#比如-1代表个位要四舍五入到十位,即精确到十位,
#-2代表十位要四舍五入到百位,即精确到百位,以此类推
```

Out[46]: 170.0

In [47]:
```
round(165.535,-2)
```

Out[47]: 200.0

In [48]:
```
round(145.535,-2)
```

Out[48]: 100.0

1.1.3.6 len()函数

In [49]:
```
len('abcdef')  #计数
```

Out[49]: 6

In [70]:
```
len([2,4,24])
```

Out[70]: 3

In [51]:
```
len((2,3,1,3))
```

Out[51]: 4

In [52]:
```
len({3:'a',4:'b',6:'abc'})   #返回键值对的个数
```

Out[52]: 3

1.1.3.7 range()函数

In [53]:
```
range(20)  #从0到19的序列.  左闭右开的区间
```

Out[53]: range(0, 20)

In [55]:
```
list(range(20))
```

Out[55]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [56]:
```
for i in range(20):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

In [57]:
```
[i for i in range(20)]    #列表推导式
```

Out[57]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [58]:
```
[*range(20)]  #   * 代表进行分配,将0到19的数分配到列表里
```

Out[58]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [59]:
```
list(range(1,20,2))
```

Out[59]: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

1.1.3.8 divmod()函数

In [60]:
```
divmod(25,6)    #商4余1,用元组的形式输出商和余数,商写在第一个位置,余数写在第二个位置
```

Out[60]: (4, 1)

In [61]:
```
divmod(35,5)
```

Out[61]: (7, 0)

1.1.3.9 zip函数

In [62]:
```
a='12345'
b=['a','b','c','d','e']
a
b  #a,b可以是字符串,可以是列表,元组等,但是个数必须要一样,可以进行一一对应
```

Out[62]: '12345'

Out[62]: ['a', 'b', 'c', 'd', 'e']

In [63]:
```
zip(a,b)  #进行打包,将a和b的元素进行一一对应的打包到一起
```

Out[63]: <zip at 0x1a44c09fe00>

In [64]:
```
list(zip(a,b))  #想要看打包的最终结果,可以用list转换成列表的形式,或者也可以转换成元组,字典,集合
```

Out[64]: [('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e')]

In [71]:
```
tuple(zip(a,b))
```

Out[71]: (('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e'))

In [72]:
```
dict(zip(a,b))  #因为a在前面,所以a作为键,b作为值
```

Out[72]: {'1': 'a', '2': 'b', '3': 'c', '4': 'd', '5': 'e'}

In [73]:
```
set(zip(a,b))
```

Out[73]: {('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e')}

In [75]:
```
[*zip(a,b)]  #用*进行转换的话,可以是列表或者集合
```

Out[75]: [('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e')]

In [76]:
```
(*zip(a,b))  #用*号不能用元组()
```

```
  File "C:\Users\99253\AppData\Local\Temp/ipykernel_10176/809079095.py", line 1
    (*zip(a,b))
     ^
SyntaxError: can't use starred expression here
```

In [77]:
```
{*zip(a,b)}  #集合可以
```

Out[77]: {('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e')}

In [78]:
```
a='123456'
b=['a','b','c','d','e']
a
b
list(zip(a,b))  #个数对应不上的话,有多余的会被舍弃
```

Out[78]: '123456'

Out[78]: ['a', 'b', 'c', 'd', 'e']

Out[78]: [('1', 'a'), ('2', 'b'), ('3', 'c'), ('4', 'd'), ('5', 'e')]

In [80]:
```
c=('小明','大明','张三','李四')
d=[55,88,69,80]
dict(zip(c,d))  #列表,元组等,可以一一对应上的,也可以进行打包,然后可以转成字典
```

Out[80]:
```
{'小明': 55, '大明': 88, '张三': 69, '李四': 80}
```

1.2 模块函数

1.2.1 导入模块的方式

In [119]:
```
import math  #第一种导入模块的方式:import后面直接加模块的名字就可以了
```

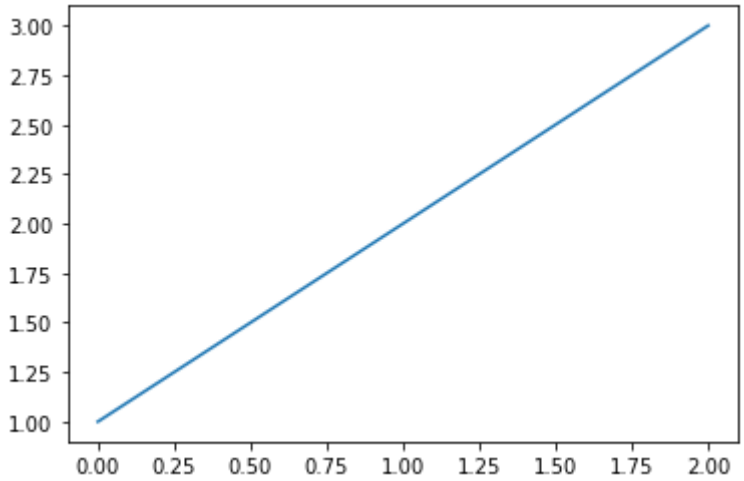
In [120]:
```
import math as mt #第二种导入模块的方式:导入之后可以给它取一个别名  (最常用的)
#如果不定义别名就会用math.来进行操作,不方便
```

In [121]:
```
import pandas as pd
import numpy as np   #做数据清洗的一些包
```

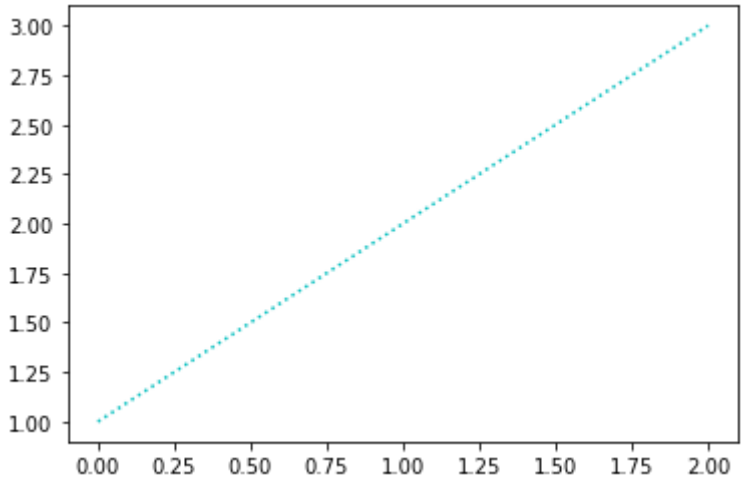
In [122]:
```
import matplotlib.pyplot as plt   #做可视化时的一些包
```

In [125]:
```
plt.plot([1,2,3])  #plt是代表matplotlib.pyplot
```

Out[125]: [<matplotlib.lines.Line2D at 0x1a44f93bd30>]



In [126]:
```
plt.plot([1,2,3],color='c',linestyle=':');
```



In [132]:
```
from math import sin #第三种导入模块的方式  要用某个模块的某个函数时用from
#这种导入方式是只导入了math这个模块的sin函数,只有sin函数可以用, math模块里的其它函数没有被导入,不能用
#参考,正课文件夹里的这个部分.我现在这里的代码执行上有一点问题
```

In [133]:
```
sin(30)
```

Out[133]: -0.9880316240928618

In [135]:
```
sin(2)  #直接通过函数名调用,只导入了sin函数,cos函数是不能用的
```

Out[135]: 0.9092974268256817

In [91]:
```
from math import *  #导入全部函数,并且调用的时候都不用加模块math这个名字,  但是不推荐这种方法
#因为后期的同名函数中分不清是Jupyter的函数还是Numpy的函数
```

In [92]:
```
cos(2)
```

Out[92]: -0.4161468365471424

1.2.2 random模块

In [137]:
```
import random   #导入用于生成随机数的模块
```

In [142]:
```
random.random()  #生成[0,1)的随机小数,()里面不需要填参数
```

Out[142]: 0.5628688113018914

In [146]:
```
random.uniform(1,19)  #生成[a,b)的随机小数,由于浮点数存储精度的原因有可能会取到b
```

Out[146]: 5.162422990278405

In [153]:
```
random.randint(5,20)  #生成[a,b]的随机整数
```

Out[153]: 15

In [188]:
```
random.randrange(1,20,3)  #指定范围和步长的随机数,包括start,不包括stop,左闭右开.
#第一个参数是起始值,第二个参数是终止值,第三个参数是步长
```

Out[188]: 13

In [190]:
```
random.randrange(50,100,2)
```

Out[190]: 74

In [193]:
```
random.seed(80)  #设置随机数种子,在该随机数种子下,执行多次,生成的随机数不会变      就是为了固定生成.....
random.randint(1,10)    #seed()括号里的数字可以是任意的,固定一个值即可.   randint()里的是取数的范围,(1,10)代表是从1到10取一个数
#生成随机整数
```

Out[193]: 5

In [196]:
```
random.seed(-2)      #seed()括号里的参数也可以是负数
random.randint(1,10)
```

Out[196]: 1

In [197]:
```
random.seed(80)  #设置随机数种子,在该随机数种子下,执行多次,生成的随机数不会变      就是为了固定生...
random.uniform(1,10)  #生成随机浮点数
```

Out[197]: 3.44342001933951

In [199]:
```
list1=[2,3,2,4,2,7,5,7]
```

In [224]:
```
random.choice(list1)  #设定取数的范围,一次只随机取一个数  #从list1里面随机取一个数
```

Out[224]: 3

In [232]:
```
random.sample(list1,3)  #设定取数的范围和取数的个数,从list1里面随机取3个数
```

Out[232]: [2, 4, 3]

In [233]:
```
set(list1)   #转换成集合,可以进行去重,这样就可以取不重复的值
```

Out[233]: {2, 3, 4, 5, 7}

In [238]:
```
random.sample(set(list1),3)
```

```
C:\Users\99253\AppData\Local\Temp/ipykernel_10176/2994876908.py:1: DeprecationWarning: Sampling from a set deprecated
since Python 3.9 and will be removed in a subsequent version.
  random.sample(set(list1),3)
```

Out[238]: [3, 2, 4]

1.2.3 math模块

函数名	函数作用
fabs(x)	返回x的绝对值，类型是浮点数
ceil(x)	取x的上入整数，如math.ceil(4.1)返回5
floor(x)	取x的下入整数，如math.floor(4.9)返回4
exp(x)	返回e的x次幂，e是自然常数
sqrt(x)	返回x的(算术)平方根，返回值是float类型
modf(x)	返回x的整数部分和小数部分，两部分的符号与x相同，整数部分以浮点型表示。例如 math.modf(4.333)，返回元组(0.3330000000000002, 4.0)
log10(x)	返回以10为基数的x的对数，返回值类型是浮点数
log(x,y)	返回以y为基数的x的对数，返回值类型是浮点数

In [239]:
```
import math
```

In [240]:
```
math.fabs(-1)    #和abs功能一样,但是输出数据类型不一样,fabs输出的是浮点数
```

Out[240]: 1.0

In [241]:
```
math.fabs(9)
```

Out[241]: 9.0

In [243]:
```
math.ceil(5.6)  #向上取整,不管小数点后是几都进行进位
```

Out[243]: 6

In [244]:
```
math.ceil(9.0001)
```

Out[244]: 10

In [248]:
```
math.ceil(6)  #如果是整数,则返回这个整数
```

Out[248]: 6

In [249]:
```
math.floor(6.933)  #向下取整,不管小数点后是几,都是舍去取整
```

Out[249]: 6

In [250]:
```
math.floor(5.9999)
```

Out[250]: 5

In [251]:
```
math.exp(2)  #返回e的几次幂,  math.exp(2)返回e的平方
```

Out[251]: 7.38905609893065

In [252]:
```
math.e  #获取e的值
```

Out[252]: 2.718281828459045

In [253]:
```
math.e**2
```

Out[253]: 7.3890560989306495

In [254]:
```
math.sqrt(25)  #开根号,返回的是浮点数
```

Out[254]: 5.0

In [255]:
```
math.sqrt(7)
```

Out[255]: 2.6457513110645907

In [256]:
```
math.sqrt(16)
```

Out[256]: 4.0

In [257]:
```
math.modf(5.7456)    #返回小数部分和整数部分
#由于计算机底层是二进制,二进制和十进制在转换的时候有一定的误差      浮点数在存储的时候有一定的精度损失
```

Out[257]: (0.7455999999999996, 5.0)

In [258]:
```
math.log10(100)  #求以10位底的对数,就是看10的几次方等于100,返回的就是几次方的几
```

Out[258]: 2.0

In [260]:
```
math.log10(1000)
```

Out[260]: 3.0

In [261]:
```
math.log(math.e**2)    #第一个参数设置真数,第二个参数设置底数,第二个参数不写默认的底数是math.e
#真数是math.e**2,底数没写是math.e ,结果为2
```

Out[261]: 2.0

In [262]:
```
math.log(math.e)  #真数是math.e,底数没写是math.e ,结果为1
```

Out[262]: 1.0

In [263]:
```
math.log(27,3)  #真数是27,底数是3 ,结果为3
```

Out[263]: 3.0

In [264]:
```
math.log(81,3)   #真数是81,底数是3 ,结果为4
```

Out[264]: 4.0

In [265]:
```
math.log(16,2)  #真数是16,底数是2 ,结果为4
```

Out[265]: 4.0

In [266]:
```
math.pi  #获取圆周率π的值
```

Out[266]: 3.141592653589793

In [267]:
```
math.e  #获取e的值
```

Out[267]: 2.718281828459045

1.3 自定义函数

自定义函数的格式:

```
def 函数名（参数1，参数2...）:
    代码块(函数要实现的功能)
    return 返回值
```

其中,函数的参数和返回值不是必须要写的

1.3.1 自定义一个函数

调用该函数可以打印出:

```
==================

人生苦短,我学Python

==================
```

In [268]:
```
def PR():
    print('==================')
    print('人生苦短,我学Python')
    print('==================')    #print就是要调用的函数的功能  定义的函数名是PR
```

In [269]:
```
PR()  #虽然没有设置参数,但是函数后面的()不能省略,如果省略了就变成了变量,而不是函数了
```

```
==================
人生苦短,我学Python
==================
```

In [270]:
```
PR()
```

```
==================
人生苦短,我学Python
==================
```

In [271]:
```
def PR1():
    print('姓名:卡卡西')
    print('年龄:20岁')
```

In [272]:
```
PR1()
```

```
姓名:卡卡西
年龄:20岁
```

1.3.2 函数的参数

In [273]:
```
def add():
    print(1234+4321)
```

In [274]:
```
add()
```

```
5555
```

In [275]:
```
def add1(a,b):
    print(a+b)  #可以往函数名add1()的括号里设置参数,参数要与print的参数一致
```

In [278]:
```
def add1(x,y):
    print(x+y)
```

In [279]:
```
add1(2,3)  #参数与参数用逗号隔开
```

```
5
```

In [280]:
```
add1(15,20)
```

```
35
```

In [281]:
```
add1(35,34)
```

```
69
```

1.3.3 参数的分类

两种分类标准:

形参和实参

位置参数和关键字参数

形参和实参:

实参是实际占用内存地址的实实在在的参数，而形参只是意义上的一种参数，在定义的时候是不占内存地址的

In [282]:
```
def add1(x,y):    #在定义的时候,x,y是形参
    print(x+y)
```

In [283]:
```
add1(2,3)  #在调用的时候,你给形参的值,比如这里的2,3是实参,是实际上参与运算的
```

```
5
```

位置参数和关键字参数

In [284]:
```
def add2(x,y,z):
    print(x+y-z)
```

In [285]:
```
add2(1,2,3)  #按照位置传参,位置参数
```

```
0
```

In [286]:
```
add2(x=5,z=3,y=2)  #按照关键字传参,关键字参数
```

```
4
```

位置参数和关键字参数混用

位置参数在前,关键字参数在后

In [287]:
```
add2(4,z=3,y=5)  #4+5-3=6    位置参数要写在前面,y和z的关键字参数要写在后面,但是y和z谁在谁的前面不影响
```

```
6
```

In [289]:
```
add2(4,y=5,z=3)
```

```
6
```

In [290]:
```
add2(y=2,z=3,5)  #报错,因为位置参数要写在前面,y和z的关键字参数要写在后面,但是y和z谁在谁的前面不影响
```

```
  File "C:\Users\99253\AppData\Local\Temp/ipykernel_10176/2023192590.py", line 1
    add2(y=2,z=3,5)  #报错,因为位置参数要写在前面,y和z的关键字参数要写在后面,但是y和z谁在谁的前面不影响
                 ^
SyntaxError: positional argument follows keyword argument
```

默认参数(参数默认值)

In [291]:
```
add2()  #add2在定义的时候没有设置默认参数,所以必须要给她填上参数,否则就会报错
#想要不报错,就需要设置默认值,看下面
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/1272118690.py in <module>
----> 1 add2()

TypeError: add2() missing 3 required positional arguments: 'x', 'y', and 'z'
```

In [292]:
```
def add2(x=0,y=0,z=0):
    print(x+y-z)  #重新定义一个函数,设置默认参数的值为0 (即add2(x=0,y=0,z=0)),这样的话,不给函数传参的话,默认为0进行计算
```

In [293]:
```
add2()
```

```
0
```

In [294]:
```
add2(5,6,3)  #传了参数则按传进来的数据进行计算
```

```
8
```

1.3.4 收集和分配参数

所谓的收集参数（不定长参数），就是说只指定一个参数，然后允许调用函数时传入任意数量的参数。

收集位置参数

In [296]:
```
def suml(*args):   #习惯上定义收集位置参数的名字为*args,也可以定义为别的
    print(args)    #标*代表以元组的形式存放
```

In [297]:
```
suml(3,4,2,5)
```

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

In [298]:
```
suml(4,5)
```

```
(4, 5)
```

In [299]:
```
suml(6,4,2,5,6,2,5,5)
```

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

In [300]:
```
def sum2(*args):
    s=0
    for i in args:
        s+=i
    print(s)  #可以用定义的函数进行遍历,然后形成计算  ,能遍历是因为存放的是元组形式
#这个代码是进行了一个累计和
```

In [301]:
```
sum2(3,4,2,4)
```

```
13
```

In [303]:
```
sum2(2,3,1,4,7,6,7)
```

```
30
```

In [304]:
```
def F(x,*y,z):
    print(x,y,z)  #  *y也是一个收集位置参数
```

In [305]:
```
F(1,2,3,1,4)  #报错,因为它不知道谁是z ,它知道1是x的,但是*y和z傻傻分不清楚,然后把2,3,1,4都给了y,    z就没有了 ,就报错了
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/1367576409.py in <module>
----> 1 F(1,2,3,1,4)

TypeError: F() missing 1 required keyword-only argument: 'z'
```

In [306]:
```
F(2,4,2,4,z=5)  #要给z明确一个值,才能知道其它值是y
#如果函数有多个参数,并且需要设置收集位置参数,一般把收集位置参数写在最后
```

```
2 (4, 2, 4) 5
```

In [307]:
```
def Z(x,y,*z):
    print(x,y,z)  #把收集位置参数写在最后, z是参数,所以打印的时候z前面不需要加*
```

In [308]:
```
Z(3,4,2,5,3,5)
```

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

收集关键字参数

In [309]:
```
def canshu(**kwargs):  #收集关键字参数,可以在调用函数的时候传入任意数量的参数,是以字典的形式存放的
    print(kwargs)  #收集关键字参数的名字前面必须放两颗星星
```

In [310]:
```
canshu(m=1,n=4,y=5)  #以字典的形式存放,    m,n,y是键,      1,4,5是对应的值
```

```
{'m': 1, 'n': 4, 'y': 5}
```

分配位置参数

一颗星代表分配

In [311]:
```
tp=(1,2)
```

In [312]:
```
add1(tp)  #报错:  因为add1里的参数本身就代表分配,现在把整个元组放进来是只有一个参数的,缺少一个参数进行计算
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/3687933709.py in <module>
----> 1 add1(tp)

TypeError: add1() missing 1 required positional argument: 'y'
```

In [313]:
```
add1(*tp)
#tp前面加上*代表收集位置参数,所以会把tp这个元组里的元素拿出来进行分配作为add1的参数,这样add1里面就有2个参数了
```

```
3
```

In [314]:
```
sum2(3,2,3,4)
```

```
12
```

In [315]:
```
[*range(1,10)]
```

Out[315]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [316]:
```
sum2(*range(1,10))
```

45

In [317]:
```
sum2(*range(1,101))
```

5050

分配关键字参数

两颗星表示分配关键字参数

In [318]:
```
def fun(name,score):
    print(name,score)
```

In [319]:
```
fun(name='小明',score=90)
```

小明 90

In [321]:
```
dict1={'name':'小明','score':90}
#要想用字典作为定义的函数的参数,这个字典中的key必须和函数中的参数的名字保持一致,进行对应
```

In [322]:
```
fun(**dict1)
```

小明 90

在定义的时候,一颗星代表收集位置参数,两颗星代表收集关键字参数

在调用的时候,一颗星代表分配位置参数,两颗星代表分配关键字参数

1.3.5 函数的返回值

In [323]:
```
add1(2,3)
#print函数打印出来的东西只是给你看一下,是nonetype 没有数据类型,没有输出,结果不能参与运算
```

```
5
```

In [325]:
```
add1(1,2)+add1(3,4)
#print函数打印出来的东西只是给你看一下,是nonetype 没有数据类型,没有输出,结果不能参与运算
```

```
3
7

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/3170172262.py in <module>
----> 1 add1(1,2)+add1(3,4)

TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

In [327]:
```
type(print(2))
```

```
2
```

Out[327]:
```
NoneType
```

In [324]:
```
2+3  #这个的结果是参与运算的
```

Out[324]:
```
5
```

In [326]:
```
2+3+8
```

Out[326]:
```
13
```

In [328]:
```
def ADD1(a,b):
    return a+b
```

In [329]:
```
ADD1(3,4)
```

Out[329]: 7

In [330]:
```
ADD1(3,5)+8
```

Out[330]: 16

In [331]:
```
ADD1(3,5)+ADD1(6,7)
```

Out[331]: 21

return可以输出多个值

In [332]:
```
divmod(7,3)  #商2余1 ,以元组的形式呈现
```

Out[332]: (2, 1)

In [334]:
```
def div(x,y):
    return x//y,x%y  #返回商和余数,以元组的形式呈现
```

In [335]:
```
div(45,3)
```

Out[335]: (15, 0)

In [336]:
```
div(16,5)
```

Out[336]: (3, 1)

1.3.6 函数的说明文档

如何让其他人能看懂函数功能及用法

如何写函数帮助文档:

第一步:先写上自定义的这个函数的作用是什么

第二步:写对参数的要求

第三步:函数的返回值是什么

要求:语言简洁明了,函数各种要求尽量详细

帮助文档的位置在def关键字表达式之后，函数功能代码块之前

In [337]:
```
def ADD(a,b):
    """
    计算两个数的加和
    参数:
    a:参数一:要求为数值型数据
    b:参数二:要求为数值型数据
    return
    两个数的和
    """
    return a+b

#帮助文档如果是一行用单引号,如果是多行就用三引号
```

In [ ]:
```
ADD()
```

In [338]:
```
ADD(3,7)
```

Out[338]: 10

1.3.7 自定义模块函数

我们先自定义三个函数:

- sum1计算两个数的和
- sum2计算一系列数的和
- sum3计算一系列数的倒数和

In [341]:
```
def sum1(a,b):
    '这个函数计算两个数的和'
    print('传入的数值为:',a,b)
    print('计算的结果为:',a+b)

def sum2(*args):
    '这个函数计算一系列数的和'
    print('传入的数值为:',args)
    s=0
    for i in args:
        s+=i
    print('计算的结果为:',s)

def sum3(*args):
    '这个函数计算一系列数的倒数和'
    print('传入的数值为:',args)
    s=0
    for i in args:
        s+=1/i
    print('计算的结果为:',s)
```

In [342]:
```
sum1(2,3)
```

```
传入的数值为: 2 3
计算的结果为: 5
```

In [343]:
```
sum2(1,2,3,4,5)
```

```
传入的数值为: (1, 2, 3, 4, 5)
计算的结果为: 15
```

In [344]:
```
sum3(1,2,3)
```

```
传入的数值为: (1, 2, 3)
计算的结果为: 1.8333333333333333
```

In [346]:
```
import sys
sys.path   #查看Python的搜索路径
```

Out[346]:
```
['C:\\Users\\99253\\Desktop\\CDA数据分析\\课件\\Pyhon\\Python资料\\预习笔记',
 'C:\\ProgramData\\Anaconda3\\python39.zip',
 'C:\\ProgramData\\Anaconda3\\DLLs',
 'C:\\ProgramData\\Anaconda3\\lib',
 'C:\\ProgramData\\Anaconda3',
 '',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\locket-0.2.1-py3.9.egg',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32\\lib',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\Pythonwin',
 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\99253\\.ipython']
```

将上面定义的函数封装进一个新的空的ipynb文件里,
步骤:1.打开一个新的ipynb文件,把上面定义的函数粘贴进去,不需要进行运行
2.给新的ipynb文件重命名一个自己喜欢的名字,
3.然后保存到路径下(File->Download as->Python(.py)->保留->到路径),查找路径的方法看上面
4.用下面的import导入模块,然后就可以通过模块调用函数进行后续的操作了

In [347]:
```
import summe  #导入模块   前提是已经将函数封装进了另一空的ipynb文件(目前这个文件在当前路径下)
```

In [348]:
```
summe.sum1(3,4)  #通过模块调用函数
```

```
传入的数值为: 3 4
计算的结果为: 7
```

In [349]:
```
summe.sum1(1,2)
```

```
传入的数值为: 1 2
计算的结果为: 3
```

In [350]:
```
summe.sum2(3,4,5)
```

```
传入的数值为: (3, 4, 5)
计算的结果为: 12
```

In [351]:
```
summe.sum2(1,2,3)
```

```
传入的数值为: (1, 2, 3)
计算的结果为: 6
```

In [ ]:
```
1.编写想要封装进模块的函数
2.把这些函数放到一个空的ipynb文件里
3.重命名(给文件取名),另存为.py格式
4.查看Python的搜索路径,把.py文件放到任意一个搜索路径的文件夹里(只需要放一个就可以)
5.导入模块,通过模块调用里面的函数
```

1.3.8 匿名函数lambda

Lambda函数是自定义函数的一种,专指用关键字" lambda"定义的无名短函数。

这种函数得名于省略了 用def声明函数的标准步骤

lambda函数的语法只包含一个语句，如下:

```
lambda [arg1 [,arg2,.....argn]]:expression  (表达式)
```

简单表达一下:

lambda 形式参数(数量不限) ：函数表达式

(lambda表达式汇总的参数不需要用括号括起来)

In [352]:
```
def A(x,y):
    return x+y
```

In [353]:
```
lambda x,y:x+y
```

Out[353]: <function __main__.<lambda>(x, y)>

In [354]:
```
B=lambda x,y:x+y
B(1,2)    #调用的第一种方法,定义一个变量,用变量来调用
```

Out[354]: 3

In [355]:
```
(lambda x,y:x+y)(2,3)  #第二种调用方式,也可以直接把lambda函数用括号括起来,在后面加上要传的参数
```

Out[355]: 5

In [356]:
```
list1=[[1,9],[2,8],[3,7]]
list1
```

Out[356]: [[1, 9], [2, 8], [3, 7]]

In [357]:
```
sorted(list1)  #默认升序,第二个参数reverse=False  以每个嵌套列表里索引位置为0的元素作为排序的依据
```

Out[357]: [[1, 9], [2, 8], [3, 7]]

In [358]:
```
sorted(list1,reverse=True)
```

Out[358]: [[3, 7], [2, 8], [1, 9]]

In [ ]:
```
sorted(list1,key=lambda x:x[1])  #升序  以每个嵌套列表里索引位置为1的元素作为排序的依据
#lambda x:x[1]这里面的x表示的是子列表
```

In [359]:
```
list1
```

Out[359]: [[1, 9], [2, 8], [3, 7]]

In [360]:
```
min(list1)  #以每个嵌套列表索引位置为0的元素进行比较,返回1,2,3中最小的元素所在的列表  ,返回的是子列表
```

Out[360]: [1, 9]

In [361]:
```
min(list1,key=lambda x:x[1])  #以每个嵌套列表索引位置为1的元素进行比较,返回9,8,7中最小的元素所在的列表
#lambda x:x[1]这里面的x代表的是子列表
```

Out[361]: [3, 7]

In [362]:
```
max(list1)  #以每个嵌套列表索引位置为0的元素进行比较,返回1,2,3中最大的元素所在的列表
```

Out[362]: [3, 7]

In [363]:
```
max(list1,key=lambda x:x[1])  #以每个嵌套列表索引位置为1的元素进行比较,返回9,8,7中最大的元素所在的列表
#lambda x:x[1]这里面的x代表的是子列表
```

Out[363]: [1, 9]

- lambda函数的应用场景:
  - python编写一些程序的脚本时，使用lambda就可以省下来定义函数的过程。比如写一个简单的脚本管理服务器，就没有必要定义函数，再去调用它。直接使用lambda函数，可以使程序更加简明。
  - 一些只需要调用一两次的函数，就**没有必要为了想个合适的函数名字而费精力**了，直接使用lambda函数就可以省去取名的过程。
  - 阅读普通函数，通常需要跳到开头def定义的位置，使用lambda函数可以省去这样的步骤。

1.3.8.1 匿名函数和其他函数连用

lambda函数和filter函数连用

```
filter的作用是过滤,依据给定的规则对数据进行筛选,把满足条件的原数据保留,不满足条件的数据删除
```

我们研究第一个内建函数是一个过滤器。我们每天都会接触到大量的数据，过滤器的作用就显得格外重要，通过过滤器，就可以保留我们关注的信息，把其他不感兴趣的东西直接丢掉。

python对filter（）的解释大致意思是：

- filter（）有两个参数。第一个参数可以是一个函数，也可以是一个None。
- 如果是第一个参数是函数的话，则将第二个迭代数据里的每一个元素作为函数的参数进行计算，把返回True的值筛选出来；
- 如果第一个参数为None，则直接将第二个参数中为True的值筛选出来。下面举个例子：

In [364]:
```
list(filter(None,[1,2,0,False,True]))

[*filter(None,[1,2,0,False,True])]   #用list的形式将filter转换成列表,或者也可以用[*  ]的方式进行转换成列表
```

Out[364]: [1, 2, True]

Out[364]: [1, 2, True]

In [365]:
```
temp = filter(None,[1,2,0,False,True])

list(temp)
```

Out[365]: [1, 2, True]

利用filter(),尝试写一个筛选奇数的过滤器：

In [366]:
```
def is_odd(n):
    return n%2  #如果n是偶数,n%2为0,0代表假,所以原数据会被过滤掉

list(filter(is_odd,range(10)))   #用来帅选出0到10的奇数,左闭右开
```

Out[366]: [1, 3, 5, 7, 9]

现在，我们使用lambda，简化上述的函数代码：

In [367]:
```
list(filter(lambda n:n%2,range(10)))
```

Out[367]: [1, 3, 5, 7, 9]

思考：如何写一个筛选偶数的过滤器？

In [368]:
```
list(filter(lambda n:n%2==0,range(10)))
```

Out[368]: [0, 2, 4, 6, 8]

In [369]:
```
list(filter(lambda n:n%2-1,range(10)))
```

Out[369]: [0, 2, 4, 6, 8]

lambda函数和map函数连用

map在这里不是地图的意思，在编程领域，map一般做“映射”来解释。

map()也有两个参数，仍然一个是函数，一个是可迭代序列，将序列的每一个个元素作为函数的参数进行运算加工，直到可迭代序列每个元素加工完毕，返回所有加工后的元素构成的新序列。

有了刚才filter()的经验，在这里举个map()函数的例子：

In [375]:
```
list2=[1,2,3,4,5]
list2
```

Out[375]: [1, 2, 3, 4, 5]

In [371]:
```
str(list2)
```

Out[371]: '[1, 2, 3, 4, 5]'

In [378]:
```
list(map(str,list2))    #map可以将list2转换成字符串形式,然后再用list转换成列表的形式
```

Out[378]: ['1', '2', '3', '4', '5']

filter进行过滤

In [381]:
```
[*filter(lambda x:x%2,range(10))]
```

Out[381]: [1, 3, 5, 7, 9]

map进行映射计算

In [382]:
```
[*map(lambda x:x%2,range(10))]
```

Out[382]: [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

filter()过滤后，返回的是结果为真（即不等于0）的对象，而map()遍历，返回的是计算结果

1.3.9 函数可以嵌套使用

In [383]:
```
def print_1():
    print('这是print1函数的内部')
```

In [384]:
```
print_1()
```

这是print1函数的内部

In [385]:
```
def print_2():
    print('这是print2函数的内部')
    print_1()    #在定义新函数的时候,也可以调用之前定义过的函数
```

In [386]:
```
print_2()
```

这是print2函数的内部
这是print1函数的内部

1.3.10 递归函数

5的阶乘 1 * 2 * 3 * 4 * 5 = 5 * 4!

4的阶乘 1 * 2 * 3 * 4 = 4 * 3!

In [387]:
```
def caljie(n):
    if n>1:
        r=n*caljie(n-1)
    elif n==1:
        r=1
    else:
        print('请重新输入')
    return r          #递归函数比较耗内存,不建议采用
```

In [388]:
```
caljie(5)
```

Out[388]: 120

我们这个例子实际上是满足了两个条件:
1）调用函数的本身
2）设置了正常的返回条件
具体的我们看成是以下分析步骤:

In [ ]:
```
caljie(5) = 5*caljie(4)      120
caljie(4) = 4*caljie(3)      24
caljie(3) = 3*caljie(2)      6
caljie(2) = 2*caljie(1)      2
caljie(1) = 1
```

1.3.11 局部变量和全局变量

In [389]:
```
def test():
    h=100
    return h
```

In [390]:
```
test()
```

Out[390]: 100

In [391]:
```
h   #会报错,因为在函数外部调用这个h,这个h是没有被定义的,因此报错
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/2324051631.py in <module>
----> 1 h

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

In [392]:
```
a=1  #全局变量
```

In [393]:
```
def test1():
    h=100
    return a+h
```

In [394]:
```
test1()
```

Out[394]: 101

In [395]:
```
def test2():
    a=50  #这里的a是局部变量   ,函数外面的a=1这个全局变量还是不变的
    h=100
    return a+h
```

In [396]:
```
test2()  #全局变量名字和局部变量名字相同时,函数优先使用局部变量
```

Out[396]: 150

In [398]:
```
def  test3():
    h=100
    return h+b  #建议函数内部要使用什么变量,在函数内部去定义
```

In [399]:
```
test3()   #b在函数内部\外部都没有被定义,所以会报错.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10176/1791326256.py in <module>
----> 1 test3()

~\AppData\Local\Temp/ipykernel_10176/822138559.py in test3()
      1 def  test3():
      2     h=100
----> 3     return h+b  #建议函数内部要使用什么变量,在函数内部去定义

TypeError: unsupported operand type(s) for +: 'int' and 'list'
```

In [400]:
```
a   #全局变量不会被局部变量影响
```

Out[400]: 1

In [411]:
```
def test4():
    global a  #global用来对全局变量进行声明,当进行了声明后,全局变量就被进行了更改
    a=50
    h=100
    return a+h
#使用global关键字声明全局变量,一定要调用函数后才会修改全局变量
```

In [412]:
```
a=1
a    #这里的a没有进行改变,是因为必须使用函数之后才能被修改
```

Out[412]: 1

In [413]:
```
test4()  #这里使用了函数,所以后面的a这个变量就会被从1修改成50
```

Out[413]: 150

In [414]:
```
a  #上面使用了函数,所以这里的全局变量a就被修改了
```

Out[414]: 50

全局变量名字和局部变量名字相同

1. 在自定义函数外边定义的变量叫做全局变量
2. 全局变量能够在所有的自定义函数中进行访问
3. 如果在自定义函数中修改全局变量，那么就需要使用global进行声明，否则出错
4. 如果全局变量的名字和局部变量的名字相同，那么自定义函数内使用的是局部变量的
5. 自定义函数内部有局部变量的用局部变量,没有局部变量的,用全局变量

In [ ]:
```

```