# Python day1

## 1 井号是单行注释

#井号之后的字符不会当做python代码执行;

选中部分批量添加注释 Ctrl+ /

选中部分批量删除注释 ctrl+ \

选中部分缩进 tab

选中部分反向缩进 shift+tab

## 2 多行注释

'''

成对的 三个引号 表示多行注释！！！ 3个引号中的文本 不当做代码

note

note

print('hello')

'''

## 3 input提示语

# input('提示语')#prompt 提示语 函数为本程序提供输入， 程序执行到这句会停下等待用户输入,输入完成回车后将输入的数据传递到程序中来，字符串需要用单引号或者双引号；

# 注意input函数的返回值统一为字符串类型数据，即便输入123也返回为字符串'123',输出任意多个数据。需要输出的多个数据在后面括号中用逗号隔开；

## 4 中文格式码

# -\*- coding:utf-8 -\*- 是告诉python解释器本文件中的字符按照utf-8格式编码,才能识别本文件里的中文字符;

## 5 python中符号的使用

# 在python中当输入的字符含有单引号，则必须使用双引号将整个字符串涵盖，或者使用转义符号\’或者用’’ ； ‘let’’s go’‘let\’s go’、‘’let’s go’’

#print(a,b,10,"hello",sep='分隔符',end='\_结束符\_\n' )

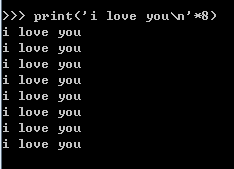
#"\n"表示换行；

例如输入：



添加换行符号后则会换行显示

Print(‘ i love you\n’ \*8）



## 6 python基本数据类型

# 数据类型,python中数据值本身是分类型的；

# number（数字型）、int（整数型）、float（浮点型，有小数点的那种）、bool（布尔型）-（bool只有两种数据true false)；

#python中bool型也当做数值类型处理，True=>1、False =<0

# print(123+True) #输出 123+1

# str（字符串）

#系统函数type(数据)返回所查询数据的类型;print (type(v1),type(v2)) #返回字符类型



#input函数的返回值都是字符串类型，如果需要输入数值数据，需要使用对应的类型函数

### 6.1变量/常量

#常量：代码中直接出现的数据内容就是常量，它和代码一样都需要占用内存空间；

#变量：Python中变量是引用"常量在内存中地址" 的一个存储空间；

#python的变量是纯引用变量，变量本身不存放数据值，而只存放'数据值'在内存中的地址；“引用“：是指变量存储数据的ID信息，而不是数据本身

#id()函数，返回变量或常量值的地址信息

a=123

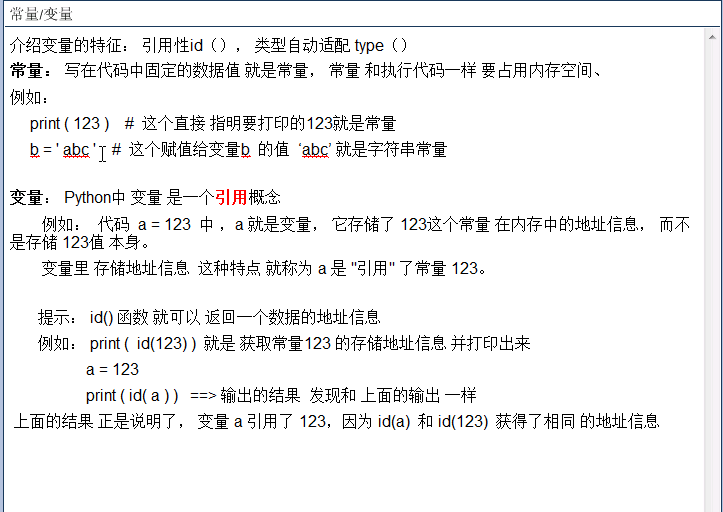
b=123

print(id(a))

print(id(b))

print(id(123))

#以上三局输出结果一样说明a和b都'引用'同一个数据



#注意：

在使用变量之前，需要对其先赋值；

变量名可以包括字母，数字，下划线，但是变量名不能以数字开头；

字母可以是小写和大写，但大小写不同；

我们所认知的字符串就是引号内的一切东西，也把字符串叫文本，文本和数字是不同的哦。

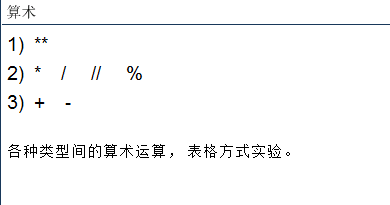
#数据之间的转换

print( bool('') ) # 空字符串‘’引号内不填任何符号包括空格，空字符--> bool False



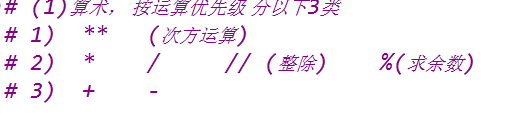
字符串 --> int 不能转换

## 7 python算数运算



#// 表示整除，%求余数，\*\* 次方运算，16\*\*（1/4）

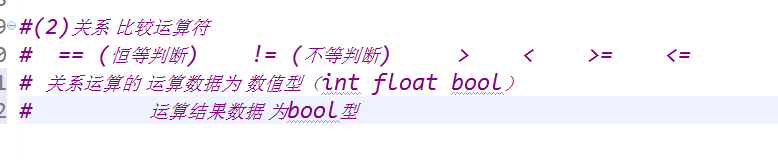
运算优先级1,2,3排序；



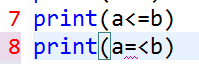
### 7.1 关系型比较运算

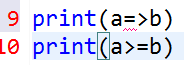
让计算机对程序中的数值大小关系的假设进行判断；

#运算数据：数值型数据运算结果是bool型数据；



### 7.2 小语法错误提示

#小于等于 <=

#大于等于 >=

#小于等于要这么写：<=,不能写成=<,同样大于等于也不能写成=>

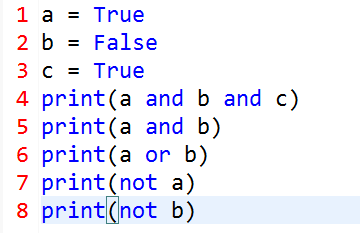
### 7.3 逻辑运算

#逻辑运算and or not

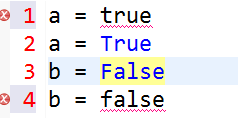
# and: 参与运算的bool数据都为真结果才为真，有一个为假结果就为假

# or:参与运算的bool数据都为假结果才为假，有一个为真结果就为真

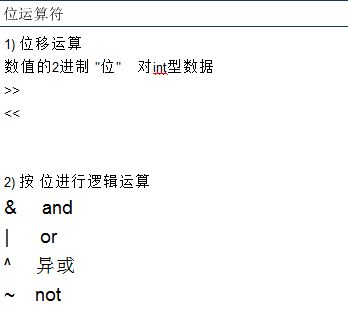
# not: 取反



# 提示一点语法错误，，python对大小写很敏感



### 7.4 位运算



#对int型数据的2进制"位"进行的按位逻辑或位移运算

# >>按位向右相邻位移动多少次

# <<按位向左相邻位移动多少次

# & 安位与

# | 按位或

# ^ 按位异或,异或规则：两个逻辑数相同则结果为'假'，不相同则结果为'真'（真1，假0）

# ~ 按位取反,规律：以0和-1之间为对称轴，按位取反就是取这个数以称轴为中心对称的另一个数

# 比如0和-1对称1和-2对称2和-3对称，所以~0 为-1,；~（-2）为1；~2为-3

a=2

b=3

c=8

print(a&b)

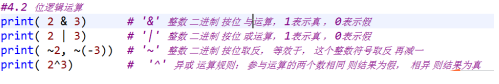
print(a|b)

print(a^b)

print(~a)

print(c>>a) #右位移相当与除以2 除多少次，比如：16 >> 3表示16除以2 除3次；

print(c<<a)#左位移相当与乘以2 乘多少次，比如：4 << 3 表示4乘以2乘3次；



举例：

Print(2&8) 结果是 0

2的二进制是1 0 ；8的二进制是 1 0 0 0 ；

长度不一样的，把短的左边补0到一样长也就是把2的二进制补齐到 0 0 1 0，然后一一对应按与计算结果是0

Print(2&10) 结果是 2

2的二进制是1 0 ；10的二进制 1 0 1 0 ；

把2的二进制补齐到10的二进制长度 0 0 1 0 ，然后一一 对应 结果是 0 0 1 0 ，去掉左边的0得1 0 ，再转化为十进制就是2

# 其他几个就不举例了，方法一样；

# Python day2

## 1 if的单分支

#[选择结构]有单分支和双分支两种结构

#[单分支]:遇到判断后只有判断为‘真’时执行一个分支的代码，分支执行完后回到主流程代码继续执行；若判断为'假' ，则不执行任何代码，直接执行主流程后面的代码；

# 主流程代码

# if (bool表达式)：

# 分支代码

# 回到主流程代码

## 2 if的双分支

#【双分支】对判断做出两种选择，判断为'真'执行一个分支判断为‘假’执行另一个分支

a=4

b=3

if(b>a):

print('b>a') # Python用行头的缩进表示下一个层次的代码；

else: # else分支是无条件的；

print('b<=a')

print('回到主流程')

## 3 %号的运用

#字符串格式控制符%d(为整数占位置)、%f(为浮点数占位置)、%s(为字符串占位置)

#【案例】

str1= '这里用整数替代进来%d' %123

print(str1)

Str2= '这里用浮点数替代进来并且取三位小数%.3f

print(str2)

str3='这里用字符串替代进来%s' %"\_替代进来的字符\_\_"

print(str3)

## 4 分支结构：

### 4.1 单条件结构

#python条件语句是通过一条或者多条语句的执行结果（true或者false）来决定执行的代码块，任何非0和空值（null）值为true，0或者null为false ；

#IF 语句单独使用

# if 判断条件 ：

执行语句

Else：

执行语句

### 4.2 多重if条件结构(elif)

# if 判断条件1：

执行语句1

Elif 判断条件2：

执行语句2

Elif 判定条件3：

执行语句3

Else:

执行语句4

#注意：python不支持switch 语句，所以多个条件判定只能用elif来实现，如有多个条件同事判定，可以使用or / and来实现。

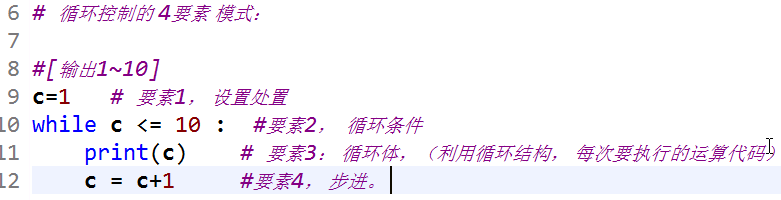
### 4.3 while(循环结构）

语法格式：

While(bool表达式）：

循环体

#循环控制的4要素模式：



#例如：

count = 1

while(count<9):

print(*'the count is:%d'*% count)

count = count+1

else:

print(*'拜拜'*)

也可以写成：

count = 1

while(count<9):

print(*'the count is:%d'*% count)

count = count+1

print(*'拜拜'*)

最后可以加 else 也可以不加直接print()

#break的作用：在循环中遇到break直接结束循环，进入循环结构之后的流程；

i = 1

while1:

print(i,end=*' '*)

i=i+1

ifi>10:

break# 跳出循环

print(*'the end'*)

#continue跳出当前循环进入下一个循环；

i=1

whilei<10:

i=i+1

ifi%2!=0: #非双数时跳过输出，进入下一个循环

Continue#先写continue再写print

print(i,end=*' '*) #输出双数

#### 4.3.1循环嵌套：

#循环嵌套：一个循环结构的循环体中又包含一个循环结构，语法上支撑无限的嵌套，我们只要求掌握2层嵌套；

#循环嵌套逻辑关键点：内层循环执行过程中对于外层循环来说还在某一个循环体执行中。

#举例：运用嵌套输出乘法表：

思维过程：里面的循环作用是输出没一个惩罚运算构成一行，外层的作用，控制输出9行，循环体代表里面的循环全部被执行完。

如下：9\*9乘法表 while嵌套函数：

a = 1  
while a <= 9:  
 c = a  
 while c <= 9:  
 s = '%s\*%s=%s'% (a,c,a\*c)  
 print(s, end='\t')  
 c += 1  
 a += 1  
 print()

#用for嵌套打印9\*9乘法表还要简单一点

for i in range(1,10):

for j in range(1,i+1):

print(*'%d\*%d=%d'* %(i,j,i\*j),end=*' \t'*)

print() #一列一列的换行输出

#print默认是打印一行，结尾加换行。end=' '意思是末尾不换行，加空格。end=' \t'表示制表。end=' \n'是换行的意思。

# sep =' 分隔符 ' 分开，分割的意思，如print(a,b,sep=' , ' 输出结果为a,b 这个好理解嘛

## 4.4 for循环结构：

#语法表达式：

For循环变量（迭代变量）in数据范围表达式：

循环体代码

### 4.4.1 for和while的区别

# for和while 的区别： for除了‘重复’功能之外，还有‘迭代’功能 ；#‘迭代’：指在重复的同时，每次带入范围内的一个新数据进入循环变量中，因此，for循环的循环体中，无论是否使用‘循环变量’循环整体被执行的次数为‘范围中的数据个数’

### 4.4.2 range函数

#range /reigi/ 范围，range函数range()是python内置函数它能返回一系列连续增加的整数，它的工作方式类似于分片，可以生成一个列表对象，

#range(n)产生0到n-1的所有整数；

#range(n1,n2)产生n1到n2-1的所有整数

#range(n1,n2,step) 产生n1到n2-1之间步数step之间的所有整数。

#range(5)#只有一个参数，则表示会产生从0开始计数的整数列表[0, 1, 2, 3,4] # python 返回值；#range(0,6)  #当传入两个参数时，则将第一个参数做为起始位，第二个参数为结束位：

[0, 1, 2, 3, 4,5]

#range(0,10,2)#填入三个参数，第三个参数是步进值（步进值默认为1）

[0, 2, 4, 6,8]

#参数和结果也并非一定要是正数或是递增的例如：

range(-4,4)

[-4, -3, -2, -1, 0, 1, 2, 3]

range(4,-4,-1)

[4, 3, 2, 1, 0, -1, -2, -3]

#举例：输入10次分数，求总分，平均分。

s = 0

c = 0

for i in range(10):

c = float(input(*'请输入分数： '*))

s = s+c

print(*'总分是%.2f,平均分是%.2f'* %(s,s/10))

#输入任意个数（0~100）求和求平均分：

sum1 = 0

count = 0

score = float(input(*'请输入分数（0~100分有效），无效分数表示退出'*))

while0<= score <= 100 :

score = float(input(*'请输入分数（0~100有效），无效表示退出'*))

sum1=sum1+score

count = count + 1

print(*'总分数%.2f,平均分是%.2f'* %(sum1,sum1/count))

#continue 跳出循环进入下一个循环：

for i in range(10):

if(i==5):

continue

print(i)

输出结果0 1 2 3 4 6 7 8 9

# Python复合数据类型

# 分为两大类：有序数列和无序数列

## 1有序数列

元组--tuple(字符串也可以被当做一种元组）元组的元素不能改变，元组使用小括号；

列表--list，方括号内用逗号隔开[2,3,’eras’]，列表的元素可以改变;

### 1.1元组

元组数据声明初始值格式变量名=（数据1，数据2......） 注意使用小括号声明初始值

t1=（10，3.5，‘hello’,True,(1,2),’a’,’b’）

元组的使用，通过元素下标 标识具体的数据（元素下标统一使用中括号），第一个元素下标从0开始

Print(t1[0]) #输出元组的第一个元素

Print(t1[4][1]) #输出元组的第5个元素（第五个元素是元组），中的第二个子元素

#字符串也是一种元组

S1=’hello’

Print(s1[2]) #表示输出s1元组中的第三个元素L

#### 1.1.1下标其他使用形式

倒序下标用负数标识，比如t1[-1] #表示获取最后一个元素 print(t1[-1])

#获取一个区间中的元素，用[开始下标：结束下标]形式表示，结束下标的元素不包含在内

Print(t1[1:4]) #表示获取下标为1 2 3 这3个元素，不包含下标为4的元素

#获取一个区间中的元素，有间隔的情况，用[开始下标：结束下标：间隔数量]形式表示。

Print(t1[1:7:2]) #输出1号3号5号元素

#获取区间元素时，不提供结束元素下标 表示直到最后的元素

Print(t1[-3: ]) #表示获取t1元组的倒数3个元素

#元组的限定：元素不能改变

T1[0]=100 #会报错

#### 1.1.2 元组的+，\*

t3=t2+t1 # +表示元组的拼接，注意拼接的顺序t2+t1和t1+t2顺序是不一样的

t4=t2\*3 # \*表示元组元素的复制多倍，print(t4);

#### 1.1.3 显示元组排序

#print(sorted(t4)) #只显示为排序；但是元组内部可不能改变顺序，仅仅只是显示排序而已；sorted /’so，tid/ 排序；

### 1.2 列表

列表--list，方括号内用逗号隔开[2,3,’eras’]，列表的元素可以改变;

#### 1.2.1修改列表值

#修改列表list 中的某一个值 数据值

L1=[1,2,3,32.1,3.4]

L1[0]=’first’#修改列表的第一个元素值为first

Print(l1)； 通过元素下标修改。

#### 1.2.2列表增加元素

1. 列表名.append(数据) 添加一个（单个数据）数据（任何类型）至列表的最后； append--附加

l1=[1,2,3,4,5,]

l1.append(7) #要加括号哦

print(l1)

1. 列表名.extend（列表/元组数据）把新的列表或元组的元素扩展到原来的列表中； extend/ik,stend/扩展

l1=[1,2,3,4,5,]

l2=[7,8,9]

l1.extend(l2) #要加括号哦

print(l1)

1. 列表名.insert（下标值（m），数据（n）） 在m位置前面插入数据n insert--插入值；

l1=[1,2,3,4,5,]

l1.insert(3,1)

print(l1)

输出：[1, 2, 3, 1, 4, 5]

#### 1.2.3列表元素删除

1. 列表名.remove（数据）删除列表中第一个发现的指定数据 ----删除一个数据

l2=[*'a'*,*'f'*,*'a'*,*'b'*,*'c'*]

l2.remove(*'a'*)

print(l2)

输出的数据是：['f', 'a', 'b', 'c']

2）列表名.pop（下标）删除指定下标处的元素并返回该元素，如果不指认默认 返回和删除最后一个

l2=[*'a'*,*'f'*,*'a'*,*'b'*,*'c'*]

l2.pop(3)

print(l1.pop(3)) --删除并返回删除的值

输出结果：b 返回删除的值

print(l2) --输出删除值后的列表

输出数据：['a', 'f', 'a', 'c']

## 2无序数列

### 2.1集合

#集合--set存储唯一性的多个数据（任何数据类型），元素必须唯一，数据是无序的，所以获取元素打印元素无法按顺序执行，初始数据采用大括号；集合不能修改某一个元素的值，因为无法指定获取某一个元素

s1={1,2,3,4,5,11,22,33,44}

print(s1.pop()) --随机删除一个元素并返回，也可以用着随机获取集合中的一个数据；

输出结果：33#可变，每次数据不一定一样是随机的

#### 2.1.1集合转列表

#通过列表转换为一个集合

l5=[1,1,1,2,2,3,3,4,4]

s2=set(l5) #会自动去除列表中重复的数据

print(s2)

输出结果：{1, 2, 3, 4}

#### 2.1.2增加集合数据

增加集合中的数据 add（）

s3={1,2,3,4}

s3.add(8)

print(s3)

输出结果：{8, 1, 2, 3, 4}

#### 2.1.3删除集合元素

#删除集合的一个元素 列表.remove（）

s3={1,2,3,4}

s3.remove(4) #删除集合中指定的值

print(s3)

输出结果：{1, 2, 3}

#### 2.1.4集合运算

#集合运算（交集运算符& 、并集运算| 、差集运算- 、补集运算^）

#交集运算&：取两个集合同有（相同）的部分；

t1={1,2,3,4,5,6}

t2={4,5,6,7,8,9}

print(t1&t2)

输出结果：{4, 5, 6} --取两集合相同的部分

#差集运算-：去到与另外一个集合相同的部分；

t1={1,2,3,4,5,6}

t2={4,5,6,7,8,9}

print(t1-t2)

输出结果：{1, 2, 3} --去掉相同的部分4,5,6

#补集运算^：只取两个集合特有（不同）的部分，去除重复的部分；

t1={1,2,3,4,5,6}

t2={4,5,6,7,8,9} print(t2-t1)

输出结果：{8, 9, 7} --去掉相同的部分4,5,6

print(t1^t2) --去掉重复的部分，留下不同（特有）的部分

输出结果：{1, 2, 3, 7, 8, 9}

#并集运算|：联合两个集合中的所有元素；

t1={1,2,3,4,5,6}

t2={4,5,6,7,8,9}

print(t1|t2) --还是去掉了重复的数据，集合所拥有的特征

输出结果：{1, 2, 3, 4, 5, 6, 7, 8, 9}

#### 2.1.5判断子集运算

判断子集运算（<,<=）

s1={1,2,3,4,5,6}

s3={1,2}

print(s1>s3) --判断集合s3是不是集合s1的子集合，也就是说集合s3中的元素集合是不是在s1集合中都有，且s1集合中还有s3集合没有的元素；

输出结果：True

#### 2.1.6获取集合元素个数

获取集合元素个数：len

s1={1,2,3,4,5,6}

print(len(s1)) --获取集合s1的元素个数

输出结果：6

#### 2.1.7成员运算

成员运算：

s1={1,2,3,4,5,6}

print(6ins1) --判断数据 6 是否在集合s1里

输出结果：True

等同于：

s1={1,2,3,4,5,6}

print(6in(s1)) --对于s1 加不加括号都可以

#字典--dict 每一个元素是一对数据（键值对 key--value），key（一般使用数字或字符串） 不能重复，value 可以重复

### 2.2字典

#### 2.2.1创建字典

创建字典

d1={'k1':1 , 'k1':2 , 'k3':3 } #重复出现的 key 会用最后一次出现的相同 key 对应的值 ，做为这个key的值

d1={*'k1'*:1 , *'k1'*:2 , *'k3'*:3}

print(d1)

输出结果：{'k3': 3, 'k1': 2}

#### 2.2.2获取键对值

获取某个键值对的值

通过 key来映射value，采用‘字典名[key值]’获取key值对应的value；

d1={*'k1'*:1 , *'k1'*:2 , *'k3'*:3}

print(d1[*'k1'*])

输出结果：2

#### 2.2.3增加键对值

增加一个键对值

集合名[键]=值

d1={*'k1'*:1,*'k2'*:2,*'k3'*:3}

d1[*'k4'*]=4 #增加一个键对值：k4’

print(d1)

输出：{'k3': 3, 'k1': 1, 'k2': 2, 'k4': 4}

#### 2.2.4更新键对值

更新一个键对值：

d1={*'k1'*:1,*'k2'*:2,*'k3'*:3}

d1[*'k3'*]=8 #更新键对值‘k3’为8

print(d1)

输出结果：{'k3': 8, 'k1': 1, 'k2': 2}

#### 2.2.5删除键对值

删除键对值

1）字典名.pop（‘key值’）

d1={*'k1'*:1,*'k2'*:2,*'k3'*:3}

d1.pop(*'k1'*) #删除键对值‘k1’和对应的值

print(d1.pop(*'k2'*)) #返回删除的键对值，并删除这个值

输出结果2

print(d1)

输出结果：{'k3': 3, 'k2': 2}

1. 字典名.popitem（）

无参数随机返回一对键对值，返回的数据是有2个元素的元组。

d1={*'k1'*:1,*'k2'*:2,*'k3'*:3}

d1.popitem()

print(d1.popitem()) --随机返回一对键对值

输出结果：('k1', 1) --随机的会变化

print(d1)

输出结果：{'k2': 2, 'k3': 3} --会变化，因为删除的是随机的键对值

a=d1.popitem()

Print(type(a))

输出：<class 'tuple'> -Tuple/to,peil/元组

#### 2.2.6字典的其他创建方式

补充：字典的其他创建方式：

1) 通过多个 2个元素的元组创建字典

t1=('k1',1) # 作为字典的键值对 1

t2=('k1',2) # 作为字典的键值对 2

t3=('k3',3) # 作为字典的键值对 3

d1= dict((t1,t2,t3)) --通过（t1,t2,t3）整个元组创建字典， 这个元组中每个元素 t1 t2 t3 又是一个包含两个数的元组；

整体表达：

t1=(*'k1'*,1)

t2=(*'k2'*,2)

t3=(*'k3'*,3)

d1=dict((t1,t2,t3))

print(d1)

输出：{'k1': 1, 'k3': 3, 'k2': 2}

1. 通过两个列表转换为字典

l1=[1,2,3]

l2=[*'a'*,*'b'*,*'c'*]

d3=dict(zip(l1,l2)) --zip链接

print(d3)

输出：{1: 'a', 2: 'b', 3: 'c'}

若l1,l2换个位置结果输出为：

{'a': 1, 'b': 2, 'c': 3}

若列表l1 ，l2的元素长度不相等，则按照少的计算

例如：

l1=[1,2,3]

l2=[*'a'*,*'b'*,*'c'*,*'d'*,*'e'*]

d3=dict(zip(l2,l1))

print(d3)--按照少的l1计算

输出结果:{'a': 1, 'c': 3, 'b': 2}

#获取字典里所有的key和所有的value

获取字典里所有的key ; 字典名.keys()

d1={*'a'*:1,*'b'*:2,*'c'*:3}

a=d1.keys()

print(a)

输出结果：dict\_keys(['a', 'c', 'b'])

获取value; 字典名.values()

d1={*'a'*:1,*'b'*:2,*'c'*:3}

a=d1.values()

print(a)

输出结果：dict\_values([3, 1, 2])

# 函数

函数的代码块以def关键词开头，后接函数标识符名称和圆括号（）括号内是参数，变量，函数内容以冒号起始，并且缩进，return[表达式]结束函数，选择性的返回一个值给调用方，（意思就是函数如个运算完成后需要返回结果必须使用 return语句，，#函数体里没有写 return 语句时，却要使用函数的返回值， 此时函数会返回以None 值,函数中任何地方的 return 被执行到的时候，这个函数就会

结束）不带表达式的return相当于返回None，，# 函数必须先定义才能使用

print( f2() )

语法：def 函数名（参数列表）：

函数体

#默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配起来的。

案例：定义加法函数，并调用：

def**add**(a1,a2): # a1,a2是参数，也是变量，多变性，可以取很多值。

if a1>0and a2>0:

a3=a1+a2

print(*'可执行函数'*)

return a3#函数的返回值，用指定的数据替代调用函数的代码 同时函数执行‘终止’

else:

print(*'a1 a2参数无效！'*)

print(*'本函数执行完成'*)

print(add(1,2)) #调用函数（必须先声明（导入）函数，才能完成），也就是说需要给a1,a2赋值,专业术语调用，调用add(1,2)，先定义才能使用。

输出结果：可执行函数 3 #return a3的返回值，如参数无效则返回None

#函数实现：对一个整数是否是质数的处理

def**iszhishu**(n):

c=2

while c <= n-1:

if n % c == 0:

returnFalse

c+=1

else:

returnTrue

print(iszhishu(7)) #为n赋值哈，专业术语调用，调用iszhishu(7)，先定义才能使用。

# 函数的参数形式

# 1）参数的多态性一个函数多个功能

def**add**(a,b):

return a+b

print(add(1,2)) # 参数的多样性

print(add(*'abc'*,*'def'*)) #参数的多样性

print(add((1,2,3),(*'a'*,*'b'*,*'c'*))) #参数的多样性

1. 传递可变参数和不可变参数（所谓可变或不可变实际上就是指向的内存地址可不可变）

# 不可变参数：

Int(字符串）、float、数值型（number）、元组（tuple）

当不可变实参被传递到函数中进行了改变，函数调用完成后原来的实际参数"不会"受到改变；

b = 10

def**f2**(c):

print(c)

c = 11 # 传递进来的是 不可变数据如果重新复制实际是引用"新的数据"而不是改变原来的数据

print(c)

f2(b)

print(b)

输出：11 10

实际上参数b是没有发生改变的；

# 可变参数:

字典型（dict）、列表型（list）、set（集合）

当可变实参被传递到函数中进行了改变，函数调用完成后原来的实际参数也会受到改变

a = [1,2,3]

def**f1**(l):

l[0]=100

print(l)

f1(a)

print(a) #函数执行完成后 再次 输出原始数据a

输出：[100, 2, 3][100, 2, 3]

# 参数的默认值

有默认值的参数必须写参数位置的最后面

def**add1** (a,b,c=100,d=200):默认值参数写在最后面，在函数调用时，若没有为参数提供指定值，则会使用默认值，如果为参数提供了指定值则会使用指定值；

#print( add1(a=1,2,3,4)) # 这句会报错 。注意规则； 传递参数时指定名称的参数必须写在不指定名称的参数之后

print(add1(20,b=3,c=4,d=1)) #这样写没问题

print(add1(20,3,d=1,c=4))#这样写也没问题

print(add1(20,b=3,c=4,a=1))#但是这样传递参数就会报错，应该是传递参数时候如果不指定名参数传递的顺序就和定义的参数变量位置想对应，默认a=20,所有后面再加一个a=1就会报错，位置函数必须在被调用函数定义中的准确顺序来进行传递

print( add1(1,c=150,d=100,b=20)) 这样写就对了指定名称的参数必须写在不指定名称的参数之后

#可变形参的函数：可变形参指的是参数个数不受限制，你传多少，它接收多少，但格式有要求

1. 元组类型的可变形参函数，形参为1个\*号元组类型

def**f**(\*gg):

print(type(gg))

print(gg)

f(1,2,3,4)

输出：<class 'tuple'>

(1, 2, 3, 4)

再来一个：

def**f3** (a, \*b):

print(type(b))

print(a,b)

f3(1,2,3,4,5,6,7,8)

输出：<class 'tuple'>

1 (2, 3, 4, 5, 6, 7, 8)

2）字典类型的可变形参函数，形参为2个\*号

def**f**(\*\*gg):

print(type(gg))

print(gg)

f(a=1,b=2,c=3)

输出：<class 'dict'>

{'b': 2, 'c': 3, 'a': 1}

#全局变量和局部变量

Global 使用global关键字就是告诉python编译器这个变量不是局部变量而是全局变量

# Python的面向对象

# python面向对象语法规则：类（class）和对象（object）

#(一)类的定义格式：

class 类名称 ():

class**Person**(): 创建一个类，类名为person，关键字 class 加上类名用来创建一个类；类名加圆括号()的形式可以创建一个类的实例，也就是被称作对象的东西。所谓对象#“笔”作为一个抽象的概念，可以被看成是一个类。而一支实实在在的笔，

则是“笔”这种类型的对象

(1)类成员方法(函数)

1.1) 属于整个类的成员方法：

def 方法名():

def**genAttr**(*self*): #类方法和我们之前定义的函数区别在于，第一个参数必须为self；一个类可以有属于它的函数，这种函数被称为类的“方法”；一个笔有书写的功能，所以“书写”就是笔这个类的一种方法

方法体

*self*.name = None

*self*.age = 1

*self*.gender = *"M"*

1.2） 属于类的实例(对象) 的成员方法 #类是一种抽象的类型，而对象是这种类型的实例。“笔”作为一个抽象的概念，可以被看成是一个类。而一支实实在在的笔，则是“笔”这种类型的对象

def 方法名(self) # self代表实例自己，代表实例，而非类

def**genAttr**(*self*):

*self*.name = None

*self*.age = 1 #方法体

*self*.gender = *"M"*

(2)类的成员属性(变量)

2.1） 属于类的成员属性

变量名 = 初始值

2.2) 属于实例的成员属性，需要在成员方法中

为实例设置属性，参数

def**\_\_init\_\_（self,a,b,c....） #用它来为每一个实例制定自己的特征，初始化新创建对象的状态，为实例设置自己的属性**

class**person**(): def**\_\_init\_\_**(*self*,name=None,age=1,gender=*'M'*):

*self*.name = None#创建属性并赋值

*self*.age = 1#创建属性并赋值

*self*.gender = *"M"*#创建属性并赋值

定义了属性才可以嘛 哈哈哈哈IMG_256补充说明：通常类中设计的成员更多是实例成员

def 实例方法名(self):

self.实例属性名 = 初值 #上面已经阐述了，为实例设置属性和初始值；

#(二)创建类的实例语法格式：

实例名或对象名 = 类名()

class**Person**():

......

p1 = Person() # 创建好第一个实例 p1为实例名

p2 = Person() # 创建好第二个实例 p2为实例名

#(三) 调用类或实例中的成员语法格式

类名.类成员

实例名.实例成员

class **Person**(): #创建类，类名为person

.......

def **genAttr**(*self*): #类的方法genAtter(self)

.......

def **talk**(*self*): #类的方法talk(self)

.......

p1 = Person() # 创建好第一个实例 p1为实例名

p2 = Person() # 创建好第二个实例 p2为实例名

p1.genAttr() # 调用P1实例的genAttr方法

p1.name=*'张三'*#调用P1实例的name属性并设置为新值

p1.age = 20#调用P1实例的age属性并设置为20

#调用类方法，要直接通过类名字加点号调用

#调用类变量的方法是“对象.变量名”。你可以得到它的值，也可以改变它的值

p1.talk() #默认调用P1实例的talk方法，全部属性，且属性为默认值；

*@classmethod*

class**Person**():

mingzu = *"汉族"*

*@classmethod*

def**f1**(cls):#类方法中的cls参数代表可以引用类的其他成员

print(*'这个是整个类的成员方法，不能被类的实例调用，只能被类名字加点号调用'*)

print(cls.mingzu) # cls.mingzu 引用类的成员属性

#类方法应该由类名字调用不要通过"类的实例"调用

# "@classmethod" 称为方法的装饰器。这个装饰器表示声明该f1方法是整个类的方法。

#classmethod装饰器用于装饰类方法，怎么说呢？，就是通常情况下想要调用类中的某个方法时，需要先将其实例化，然后利用实例调用该方法，但是如果我们想要用类（而不是实例）直接调用方法呢？那么就需要使用classmethod装饰器了，具体看看下面的举例解释

我们举个例子来说明@classmethod

现在我们声明一个类：foo，其中包含两个方法，f1,f2

其中f1为实例方法，调用时需要先将类实例化；f2为类函数，调用前不需要实例化，注意f2的参数为

Cls，也就是类本身，f1实例为self，如下：

1）class**foo**(object): #创建类foo

def**f1**(*self*): #f1为实例方法，用self

print(*'normal'*) #方法体

*@classmethod #classmethod装饰器*

def**f2**(cls): #f2为类方法，用cls

print(*'class'*) #方法体

1. 加入我们想要调用f1，所以需要先实例化，foo\_sl是foo的一个实例

foo\_s1 = foo()

1. 调用f1

foo\_s1.f1()

输出结果：normal

1. 假如我们使用类直接调用f1，就会提示错误

foo.f2()

会报错啦！！！

1. 而用类方法就可以直接调用

Foo.f2()

输出：class

1. 当然，实例也可以直接调用类方法：

Foo\_s1.f2()

输出：class