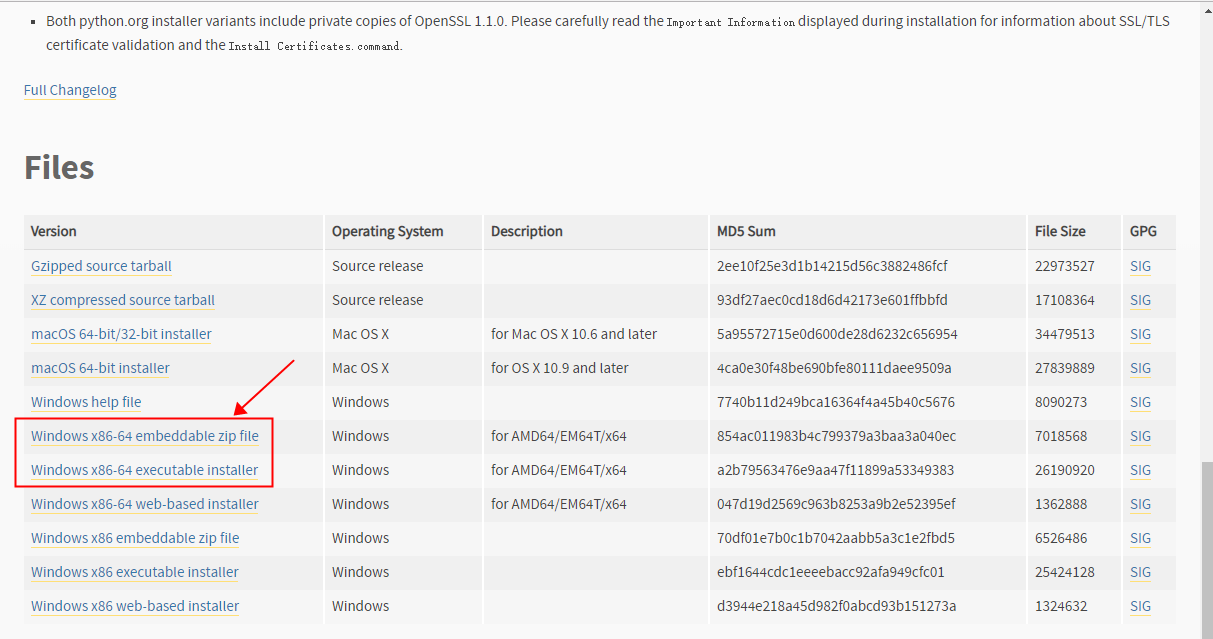
1. 关于print

print()函数也可以接受多个字符串，用逗号“,”隔开，就可以连成一串输出

逗号“,”最后并不会被打印,而是转换成空格

若打印字符串则用单引号或双引号,若打印计算结果,则不能用单引号或双引号

1. 在下载Python解释器的时候,是下载embeddable还是executable非常有讲究,必须下载executable的,否则在安装pycharm配置解释器的时候会报错



1. Python语言采用缩进方式,需严格控制缩进,冒号后面一般代表缩进块
2. Python输出的字符串包含单引号,则用双引号,若包含双引号则用单引号,若既包含单引号有包含双引号则采取转移字符”\”,其中\n表示换行,\t表示制表符,\\表示转义
3. R’’表示内部的字符不需要转义,按原样打印输出,原始字符串

但是字符串中含有单引号还是需要转移的,因为引号都是成对出现的,这里存在语法错误

’’’…’’’的格式表示多行内容,多行代码中包含单引号或双引号不需要转义

1. 布尔值,变量,常量,空值,整数和浮点数,除法,\表示浮点数的除,\\地板除,表示整数的,还有取余
2. 字符串的运算

字符串的拼接:+完全连接在一起,中间没有空格

字符串的乘法:\* 注:后面只能跟数字,字符串与字符串相乘,程序报错

字符串的提取:用[],字符串的下标从0开始,从后往前提取的时候从-1开始

举一反三: 1.前面是数字,后面是字符串的乘法可以行通吗 也可以

2.字符串的空格算一个字符吗 算,且可以打印

3.截取多个连续字符时该如何做 用[m:n]表示,是冒号

4.截取某几个不连续的字符串该如何做 用[1,7]这种方式程序会报错

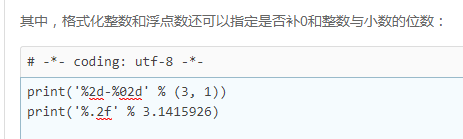
对于3问题的解决办法,Python支持截取某一连续部分的字符串,用[m:n]表示,注:只会截取到n的前一位,n也可以是负数

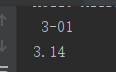
5.对于截取最后几个字符串时,用正数来表达,注:最后的n一定要大于字符串的步长,否则最后一个字母截不到,思考,如果远超过步长会怎么样,远超过步长会按照字符串最大长度计算

6.用负数该如何表示,负数-1会将最后一个字母隐去,0的话则会截取空字符串,解决办法:n不写即可 [6:]默认到字符串的结尾 ,若m省略及[:-4]从开头开始取,到倒数第四个结束

1. Python中的格式化







1. type()函数:判断类型
2. Python中的二进制:用0b表示, Python中的八进制:用0o表示,十六进制:0x

二进制的0b10表示十进制的2

1. Bin函数:将其他进制数转换为二进制

Int()函数:将任意进制数转换为十进制

Hex()函数:将任意进制数转换为十六进制

oct()函数:将任意进制数转换为八进制

1. Number包含四个子类型:int float booler complex(复数)

疑问:bool类型怎么会属于数字类型

Bool类型转换为数字

只有非0 的数字表示真,0表示假

疑问:能否将二进制,八进制,十六进制转换为bool类型 也可以转化

并不是说只有数字类型可以转换为bool类型,字符串也可以,字符串不为空时,为真,字符串为空时为false,思考:字符串为null时 ’Null’表示True Null不存在单纯的null,程序会报错

疑问:那么是否可以说bool类型也是字符串的子类型,所以将bool类型单独归为一类?

一系列空值都为false,空字符串,空列表[],空元组(),空集合{},None

# 慕课网

# 第四章 基本数据类型

## 组

### 1. 列表[] list

列表list  
#同一种类型的列表  
print([1,2,3,34])  
print(type([1,2,3]))  
#组合类型列表  
print([True,False,1,2,"Helo World",0xfff])  
print(type([True,False,1,2,"Helo World",0xfff]))  
#嵌套列表  
print([[1,2,True],[False,"hello"],["world"]])  
print(type([[1,2,True],[False,"hello"],["world"]]))  
#列表的基本操作,访问列表  
#注:如果用单一的数字,索引的方式来访问列表,则得到的结果是字符串,如果用冒号去访问则得到的还是一个list  
print([True,False,1,2,"Helo World",0xfff][0])  
print([True,False,1,2,"Helo World",0xfff][4])  
print(type([True,False,1,2,"Helo World",0xfff][4]))  
print([True,False,1,2,"Helo World",0xfff][4:])  
print([True,False,1,2,"Helo World",0xfff][-1:])  
print(type([True,False,1,2,"Helo World",0xfff][-1:]))  
#访问嵌套列表,不管是单一的索引还是冒号,得到的结果类型都是list  
print([[1,2,True],[False,"hello"],["world"]][0])  
print(type([[1,2,True],[False,"hello"],["world"]][0]))  
print([[1,2,True],[False,"hello"],["world"]][0:2])  
print(type([[1,2,True],[False,"hello"],["world"]][0:2]))  
#列表的加法运算  
print([True,False,1,2,"Helo World",0xfff]+['明月'])  
#列表的减法  
#列表不存在减法,程序会报错  
#列表的乘法,同字符串,不能列表与列表相乘  
#表示32个队伍的分组情况  
print([1,2,3,4],[2,2,3,4],[3,2,3,4],[4,2,3,4],[5,2,3,4],[6,2,3,4],[7,2,3,4],[8,2,3,4])

### 2. 元组

元组的语法规则与列表完全一样,此处不加赘述,元组表示方式()

元组与列表的区别自行百度

https://blog.csdn.net/Star\_SDK/article/details/80600673

#元组()  
print((1,2,3))  
print(type((1,2,3)))  
#奇怪的现象,当元组里只有一个元素时,类型会发生变化  
#思考:为什么会发生变化  
#原因是小括号在Python中不仅仅可以表示元组还可以表示数学运算,所以在Python中规定,默认当做优先级运算符来处理  
#那如何解释元组中只有一个字符串,str类型呢  
print(type((1)))  
print(type(('hello')))  
#思考:Python中如何表示只有一个元素是元组呢,用逗号隔开即可  
print((1,))  
print(type((1,)))  
#Python中的空元组,()即可  
print(())  
print(type(()))

### 3. 序列与序列共有的操作

1. #str,list,tuple叫序列  
   #序列的共有操作:1.切片[m:n] 2.+,\*  
   #切片很有意思的用法,切片中含有三个数字,意义和用户自行研究  
   print('Hello World'[0:8:2])  
   #若我想判断一个元素是否在一个序列中,此处引入in和not in  
   print(3 in [1,2,4,3])  
   print(10 in [1,2,4,3])  
   print('h' in('hello world'))  
   print('H' in('hello world'))  
   print(3 not in [1,2,4,3])  
   #如果想知道一个序列的长度,此处引入len()方法  
   print(len([1,2,3,4,5,6]))  
   print(len('Hello World'))  
   print(len(('hello','world'))) #返回结果为2  
   #获取序列中最大的元素,此处引入max()函数  
   print(max(1,2,3,4,5))  
   print(max('Hello World')) #r  
   print(max(('Hello','world'))) #world  
   #获取序列中最小的元素,此处引入min()函数  
   print(min(1,2,3,4,5))  
   print(min('Hello World')) #返回的是空格  
   print(min('HelloWorld')) #返回H  
   print(min(('Hello','world'))) #返回Hello  
   #当进行字符串字母比较时,计算机会对字符串字母所对应的的ASCII码进行比较,此处引入ord()函数,获取字母的ASCII码  
   #在使用ord函数时犯了一个错误,导致程序报错,ord函数中的字母忘记加引号,导致不是字符串,且获取空格的ASCII码时,单引号之间一定要有一个空格  
   print(ord('r'))  
   print(ord('H'))

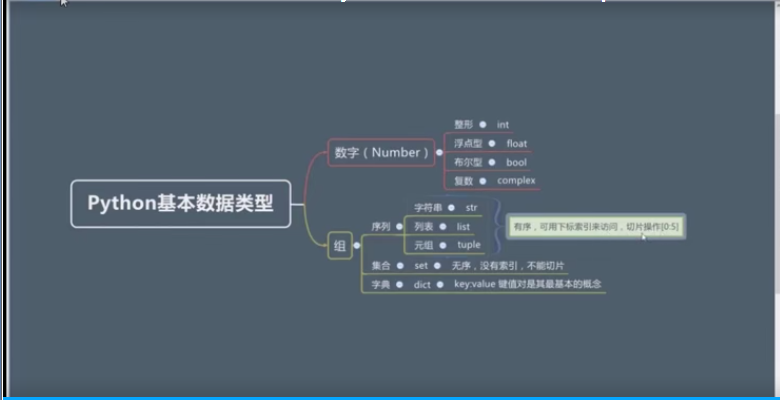
print(ord(' '))

### 4. 集合{} set

#集合 set类型 用{}表示  
#集合与序列两个最大的区别 1.无序 2.不重复  
print({1,2,3,4,5,6})  
print(type({1,2,3,4,5,6}))  
print({1,True,False,"Hello"}) #此处返回结果{False, 1, 'Hello'}  
#原因:True和1都代表1,集合具有不重复的特点,所以...也证明集合是可以混合类型的  
#证明无序,无法用下标和切片的方式访问集合,程序会报错  
#print({1,2,3}[0])  
#print({1,2,3}[0:1])  
#证明不重复,会自动去掉集合中重复的元素  
print({1,1,2,3,3}) #返回结果{1, 2, 3}  
#集合也可以使用len()函数获取集合的长度,其中重复元素不重复计算长度  
print(len({1,1,2,3,3}))  
print(len({1,True,False,"Hello"}))  
#判断一个集合是否包含某一个元素 in和not in  
print(3 in {1,2,3})  
print(3 not in {1,2,3})  
#集合特有的一些操作 1.取两个集合的差集 "-"  
# 2.取两个集合的交集 "&" 3.去两个集合的并集 "|"  
print({1,2,3,4,5,6}-{3,4})  
print({1,2,3,4,5,6}&{3,4})  
print({1,2,3,4,5,6} | {3,4,7})  
#如何定义一个空的集合,下面的方法返回的类型为dict,并不是空集合  
print({ })  
print(type({ }))  
#正确定义空集合的方法,需要用到set关键字,set()  
print(set())  
print(type(set()))  
print(len(set()))

### 5. 字典 dict

### 6. 总结



# 第五章 变量和运算符

## 1. 变量

变量名:由数字,字母,下划线组成且不能以数字开头,且不能是保留关键字

#变量  
#来看一个变量的例子,注意Python非常注意缩进,此处曾多一个空格导致代码报错  
a = 1  
print(id(a)) # 140726049989264  
b = a  
print(id(b)) # 140726049989264  
a = 3  
print(id(a)) # 140726049989328  
print(b) # 1  
# 对比看以下结果:b被改变  
a = [1, 2, 3, 4, 5]  
print(id(a))  
b = a  
a[0] = '1'  
a.append(6)  
print(id(a))  
print(b) # 此处打印的结果是['1',2,3,4,5, 6]

#思考:为什么会是这样  
#原因:因为int是值类型,list是引用类型  
#值类型,a重新赋值后指向新的值,而list类型重新赋值后还是指向原来的被改变过的值  
#其中值类型包括:int str tuple(不可改变),引用类型包括list,set,dict(可改变)  
#有些非保留关键字虽然也可以作为变量名,不会报错,但是在你用到这个名字的时候会被当成变量来处理导致报错  
#比如type和print  
type = 1  
#print(type(1)) #系统报错:TypeError: 'int' object is not callable  
#既然说字符串是不可改变的,为什么a没有报错,此处不是改变,而是定义了一个新的字符串  
#可以通过id()函数来证明,获取变量的内存地址  
a = 'Hello'  
print(id(a))  
a = a +'python'  
print(a)  
print(id(a))  
#此处会报错,因为str是不可改变类型  
a = 'Hello'  
#a[0] = 'o'  
#print(a)

## List和tuple的区别

#tuple和list去区别 1.list是可变类型,tuple是不可变类型  
a = [1,2,3]  
a[0] = '1' #这是可行的  
a = (1,2,3)  
#a[0] = '1' #这会导致代码报错,因为这是不可变类型  
#此处引入新的知识点.append 向列表末尾添加一个元素,可以给列表追加元素,但是不能给元组追加元素  
a = [1,2,3]  
print(id(a))  
a.append(4)  
print(id(a))  
print(a)   
#此处引出来的新疑问,a重新赋值后打印出来的是空,如果不赋值则打印出来的是正确的,这问题百思不得其解  
a = [1,2,3]  
a.append(4)  
print(a)  
#元组不可改变类型的优势,多个人共同写代码,有些是不可以让人改变的

## 嵌套列表的访问

#如何访问到4  
a = (1,2,3,[1,2,4])  
print(a[3][2])  
#如何访问到b  
a = (1,2,3,[1,2,['a','b','c']])  
print(a[3][2][1])  
#思考下面的元组为什么可以被修改  
#猜测:修改的是元组中的列表,列表是可改变类型  
#猜测得到验证,将列表改为元组后,代码报错  
a = (1,2,3,[1,2,4])  
a[3][2] = '4'  
print(a)  
a = (1,2,3,(1,2,4))  
#a[3][2] = '4'

## 运算符

### 4.1 算数运算符

+

-

\*

/(得到的结果是float类型)

//(地板除,得到的是int类型,且取整)

%(取余)

\*\*(平方) 2\*\*5 2的5次方

### 4.2 赋值运算符

=

+=

-=

\*=

/=

//=

%=

\*\*=

注:Python语言不支持自增和自减

### 4.3 比较(关系)运算符

注:并不局限于数字的比较,操作的结果将会返回bool

==

!=

>

<

>=

<=

#关系运算符,不仅仅可以进行数字的比较,且字符串,列表,元组等等都可以进行比较  
print(1 == 1)  
print([1,2,3] <[2,3,4])  
print((1,2,3)>(2,3,4))  
print({1,2,3} >= {1,2,3})  
#print({1,2,3} >= (1,2,3)) #不同类型比较报错  
#注:关系运算符中==与!=可以进行不同类型进行比较,但是>,>=,<,<=必须比较同类型  
#print('1'>1) #报错信息:TypeError: '>' not supported between instances of 'str' and 'int'  
b = 1  
b += 1 >= 1  
print(b) #2  
  
b = 1  
b += 1 > 1  
print(b) #1  
  
b = 1  
b += 1 < 1  
print(b) #1  
  
b = 1  
b += 1 <= 1  
print(b) #2  
#验证关系运算符比较的时候,字符串比较,是按照和来比较还是按照一个一个字母去比较  
#事实证明并不是按照字符串ASCII码的和来比较,而是按照一个一个字母的ASCII码比较  
print('abc' > 'bca')  
print('abc' < 'bca')

### 4.4 逻辑运算符

逻辑运算符操作的是bool类型,返回的也是bool类型

And

Or

Not

### 4.5 成员运算符

特点1.判断的是一个元素是否在另外一组与元素里 2.返回的值还是bool

可以是str,list,tuple,dict

In

Not in

\#成员运算符 in not in  
#可以是str,list,tuple,set,dict,在字典类型判断的key  
a='h'  
print(a in 'hello')  
a = 1  
print(a in [1,2,3])  
print(a in(1,2,3))  
print(a in {1,2,3})  
print(a in {1:2,2:3}) #True  
print(a in {2:1,2:3}) #False  
print(a in {'c':2})

### 4.6 身份运算符

#身份运算符 is和not is  
#此处需要引入对象的概念,后期详细介绍  
#最终的返回结果也是bool  
#is和==的区别,关系运算符==比较的是两个变量的值,is比较的是两个变量的内存地址

#==相当于java中的equal()和==  
a = 1  
b = 1.0  
print(a == b) #True  
print(a is b) #False  
print(id(a))  
print(id(b))  
#思考题:以下 a==b ,a is b ,c ==d, c is d  
a = {1,2,3}  
b = {2,1,3}  
print(a == b) #True 猜测set是无序的  
print(a is b) #False,要完全相等才会是True  
print(id(a))  
print(id(b))  
c = (1,2,3)  
d = (2,1,3)  
print(c == d) #False 猜测tuple是有序的  
print(c is d) #False 要完全相等才会是True  
print(id(c))  
print(id(d))

### 4.7 类型判断 isinstance()

对象三个特征

值 身份 类型

#判断类型,推荐方法,引入函数isinstance()  
#用法:isinstance(变量,变量类型),且第二个参数可以接受元组,元组中传入多个类型  
a = 1  
print(isinstance(a,str))  
print(isinstance(a,int))  
print(isinstance(a,set))  
print(isinstance(a,list))  
print(isinstance(a,tuple))  
print(isinstance(a,(str,int,list)))

### 4.8 位运算符

# 第六章 分支 循环与枚举

## 1. 表达式

表达式定义: 表达式是由一系列运算符和操作符所组成的序列

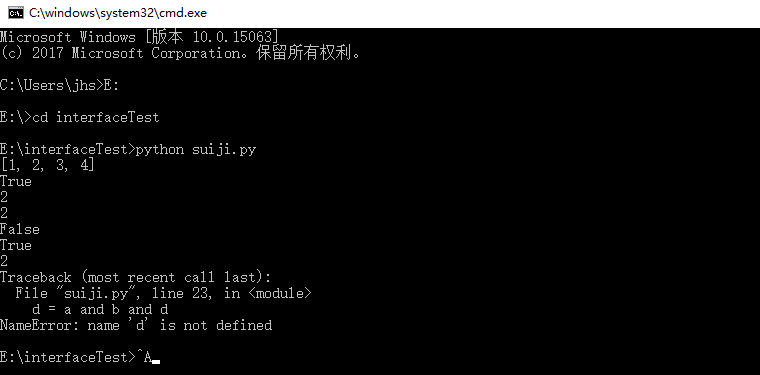
## 2. 表达式的优先级



#思考题:计算 a + b \* c 和 a or b and c  
a = 1  
b = 2  
c = 3  
print(a+b \*c) #7  
print(a or b and c) #3 按从左到右的顺序计算,若and的优先级高于or则结果1  
#返回的结果为1,实际证明and的优先级高于or,由此引出表达式的优先级  
#表达式的优先级  
#同级表达式之间的优先级自己研究,当not和and及or在一起运算时，优先级为是not>and>or  
a = 1  
b = 2  
c = 2  
print(not a or b + 2 == c)  
#此表达式的优先级排序为((not a) or ((b + 2) == c)) #False

## 3. Windows下运行python脚本

#注:用cmd打开,进入相应命令运行脚本,之前一直失败是因为当时直接python解释器打开,所以运行失败



## 4条件语句

#条件语句  
#判断用户输入用户名与密码是否一致,若一致则输出登录成功,否则提示请输入正确的用户名和密码  
account = '15555877052'  
password = 'hu123456'  
  
user\_account = input('please input account:')  
user\_password = input('please input password:')  
if account == user\_account and password == user\_password:  
 print('登录成功')  
else :  
 print('密码错误')

### 4.1 此处引入了is和==的区别

#此处重点解释一下python中is和==(is not和!=)的区别  
'''  
is用于判断两个变量的引用对象是否为同一个,==用于判断引用变量的值是否相等,类似于java的equal()和==  
反之,is not用于判断两个变量是否引用不同的对象,而!=判断两个引用变量的值是否不等  
'''  
#整数的比较  
x = 5  
y = 5  
print(x == y) #True  
print(x is y) #True  
print(id(x))  
print(id(y))  
#字符串的比较  
x = 'abc'  
y = 'abc'  
print(x == y) #True  
print(x is y) #True  
print(id(x))  
print(id(y))  
#list列表的比较  
x = [1,2,3]  
y = [1,2,3]  
print(x == y) #True  
print(x is y) #False  
print(id(x))  
print(id(y))  
#tuple元组的比较  
x = (1,2,3)  
y = (1,2,3)  
print(x == y) #True  
print(x is y) #False  
print(id(x))  
print(id(y))  
#dict字典的比较  
x = {'id':1,'name':'Hu','age':26}  
y = {'id':1,'name':'Hu','age':26}  
print(x == y) #True  
print(x is y) #False  
print(id(x))  
print(id(y))  
#set集合的比较  
x = {1,2,3}  
y = {1,2,3}  
print(x == y) #True  
print(x is y) #False  
print(id(x))  
print(id(y))  
#赋值后比较  
x = {1,2,3}  
y = x  
print(x == y) #True  
print(x is y) #True  
print(id(x))  
print(id(y))  
#总结:如果引用变量为int或str,且值相同,则x is y是True,否则为False  
#当字符串中含有特殊字符时,也是True  
x = 'abc$'  
y = 'abc$'  
print(x == y) #True  
print(x is y) #True  
print(id(x))  
print(id(y))

### 4.2 Python基本编程规范

看波浪线的提示对代码进行修改

此处引入snipace小插曲

1. 自动补全函数,类等等 2.tab键进入下一个

3.0中还行不能自动补全了

Pass占位符

### 4.3 嵌套if else 封装

# 嵌套if else  
a = input('请输入a的值:')  
print(type(a))  
if a == 1:  
 print('apple')  
else:  
 if a == 2:  
 print('orange')  
 else:  
 if a == 3:  
 print('banana')  
 else:  
 print('go shopping')  
# 解决为什么输入什么,返回的结果都是go shopping  
# 原因是因为你输入的虽然是1,但是类型是str,原因是input()函数,其接受任意输入, 将所有输入默认为字符串处理,并返回字符串类型  
# 思考如何输入int类型的1,使用int()函数(将变量转换为十进制数,所以此处只能输入数字,否则代码报错)  
# 引入elif  
a = input('请输入a的值:')  
print(type(a))  
a = int(a)  
print(type(a))  
if a == 1:  
 print('apple')  
elif a == 2:  
 print('orange')  
elif a == 3:  
 print('bnana')  
else:  
 print('go shopping')  
  
# 写一段代码要求用户只能输入数字类型  
a = input('请输入数字类型的值:')  
try:  
 a = int(a)  
 if a == 1:  
 print('apple')  
 elif a == 2:  
 print('orange')  
 elif a == 3:  
 print('bnana')  
 else:  
 print('go shopping')  
except ValueError:  
 a = input('只能输入数字,请重新输入:')

# 第七章 包、模块、函数与变量作用域

## 1.循环

### 1.1 while循环

While expression:

语句块

While还可以与else结合使用

While expression:

语句块

Else：

语句块

# while循环  
counter = 0  
while counter <= 10:  
 counter += 1  
 print(counter)  
# 注：counter要先定义，否则代码会报错  
# while与else的结合  
counter = 0  
while counter <= 10:  
 counter += 1  
 print(counter)  
else:  
 print('EOF')

一般情况下，for循环用的比较多，但是在递归的时候会用到for

### 1.2 for循环

for循环主要是用来遍历/循环 序列或者集合、字典

for循环基本结构

for target\_list in expression\_list:

pass

target\_list:我们自己定义的任意一个符合python规则的变量

expression\_list:需要遍历的list，set,tuple,dict

# for循环  
a = [1, 2, 3]  
for x in a:  
 print(x)  
# 代码块有一个特性，代码块内部还可以嵌套代码块  
# 要求打印出列表的所有元素  
a = [['apple', 'orange', 'banana', 'grape'], (1, 2, 3)]  
for x in a:  
 for y in x:  
 print(y)  
# 此处引入小知识点，如果我们希望打印出列表的所有元素在一行打印出来，该怎么做  
# 解决办法：在打印的后面加“，end = ‘’”即可,在python中默认是end = '\n',以什么为结尾  
a = [['apple', 'orange', 'banana', 'grape'], (1, 2, 3)]  
for x in a:  
 for y in x:  
 print(y, end=',')  
# 实验打印列表会怎么样  
a = [['apple', 'orange', 'banana', 'grape'], (1, 2, 3)]  
for x in a:  
 print(x)  
# ['apple', 'orange', 'banana', 'grape']  
# (1, 2, 3)  
# for循环也有和else结合使用的，代表的意思是当列表里的所有元素都被遍历完之后执行该代码块，但在实际情况下很少用到for与else结合使用  
a = [['apple', 'orange', 'banana', 'grape'], (1, 2, 3)]  
for x in a:  
 for y in x:  
 print(y, end=',')  
else:  
 print('EOF') # 最后的EOF被打印  
# 强行终止某一段代码 引入break和continue  
# 当遍历到2的时候终止循环，break终止当前循环，并且break后的代码不会被执行，continue终止当前条件的循环，继续下面的循环  
a = [1, 2, 3]  
for x in a:  
 if x == 2:  
 break  
 print(x) # 只打印1  
  
a = [1, 2, 3]  
for x in a:  
 if x == 2:  
 continue  
 print(x) # 1 3  
# for与break和else结合使用，else中的代码块不会被执行,若for循环不是正常退出，而是通过强制退出则不会执行else语句的  
a = [1, 2, 3]  
for x in a:  
 if x == 2:  
 break  
 print(x) # 只打印1  
else:  
 print('EOF') # 不会被打印  
# 思考：如果换成continue，那么else语句是否会被打印出来  
# 结果：是会被打印出来的，因为continue相当于正常的将所有元素遍历完  
a = [1, 2, 3]  
for x in a:  
 if x == 2:  
 continue  
 print(x) # 只打印1  
else:  
 print('EOF') # 会被打印  
# 思考：加入break后为什么打印结果是这个  
a = [['apple', 'orange', 'banana', 'grape'], (1, 2, 3)]  
for x in a:  
 for y in x:  
 if y == 'orange':  
 break  
 print(y)  
else:  
 print('EOF')  
# 原因：这是一个嵌套循环，break跳出的是内层循环，外层循环还执行，所以打印出1，2，3，相当于正常遍历完所有元素，所以打印出else的代码块  
  
# python中如何像其他语言一样  
# for(i= 1,i< 10,i ++){  
# }  
# python提供range()函数,range(m, n, l)左闭右开,m代表从m开始,n代表结束的元素,l代表步长,其中l可以省略  
for x in range(1, 10):  
 print(x) # 从1打印到9  
for x in range(1, 10, 2):  
 print(x) # 1,3,5,7,9  
# 思考:想打印一个递增的等差数列  
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
for x in range(0, len(a), 2):  
 print(x, end=' | ') # 0|2|4|6|8|  
# 由此处可以看出range的第一个元素与序列的起始下标并无关系,仅代表从m开始  
# 思考:想打印一个递减的等差数列  
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
for x in range(len(a), 0, -2):  
 print(x, end=' | ')  
# 打印出a列表相同间隔的元素  
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
for i in range(0, len(a), 2):  
 print(a[i], end=' | ') # 1 | 3 | 5 | 7 | 9 |  
# 更好的办法,利用切片  
b = a[0:len(a):2]  
print(b)

## 组织结构

# 组织结构  
# 暂时理解,包(一个文件夹) -> 模块(一个一个的文件) -> 类 -> 函数,变量(并不是一个组织结构)  
# python中如何区分一个普通的文件夹和包呢  
# 如果你想让一个文件夹成为包的话,你必须在这个文件夹下面包含一个特定的文件 \_\_init\_\_.py，否则它就是一个普通的文件夹  
# 强调\_\_init\_\_.py本身也是一个模块,文件可以为空也可以有代码  
# 注:对于一个模块来说命名空间就是包名.模块名，但\_\_init\_\_.py比较特殊，它的命名空间就是folder，就是他的包名  
# 从左图结构中可以看出folder，folder2是包，而folder3就是一个普通的文件夹  
  
# 命名空间:为了区分不同包名下相同的模块,一个包下可以有多个模块  
# folder.7.py folder2.7.py  
# 一个包下面还可以有子包，包下面还可以建文件夹，如folder -> folder1-1,子包的概念  
# 并不是说包下面只能包含包,和包平级的也可以是模块，如：4.py和folder1-1是平级，4.py是模块

## Import

### 3.1 导入模块

# import导入模块  
# 在某个模块中定义的变量或函数等,想在其它模块中引用,可用导入,所以通常我们将一些公用的东西,重复的东西放到同一个模块下,使代码具有复用性  
# 导入模块有两种方法,方法一:import 包名.模块名(同一个包下的引用,包名可以省略)  
# 引用的时候 命名空间.变量名,其中命名空间也可以使用别名,用as即可  
# 同一个包下导入，在folder下4.py中定义变量a ,在7.py中引用  
# import 4  
# print(4.o)  
# 遇到问题，代码一直提示错误identifier expected，原因：查看脚本命名是否是纯数字，纯数字的脚本导入会报错（我就是败在这一步）  
# import suiji  
# print(suiji.o) 这段代码还是报错，原因2019.1.3版本中即使是同一个包下面的模块导入也需要加上包名  
print('分割线1----------------')  
import folder.hello  
print(folder.hello.o)  
# 返回结果：(1, 2, 3) (1, 2, 3)  
# 思考：为什么o变量被打印两次，原因import导入的是hello这个模块，会将hello模块所有东西打印一遍  
# 有时候会打印两遍，有时候又不会，不知道什么原因，好像是第一次导入是会将整个模块打印，第二次就不会了  
# 原因:因为在同一个模块中,包不会重复导入  
# 当命名空间过长时，可以使用别名  
print('分割线2----------------')  
import folder.hello as b  
print(b.o)  
# import只能导入模块，不能直接导入变量

### 3.2 from import

1. # 导入方式2 from module import 变量或函数等  
   # 两者的区别：import导入的是一个模块，from ...import导入的时变量或函数  
   print('分割线3----------------')  
   from folder.hello import o  
   print(o) # 这个地方就不用再写模块了，可直接引用  
   # 思考：from .. import的import后面是否仅仅只可以接变量和函数，能否接模块，答案是可以的  
   print('分割线4----------------')  
   from folder import hello  
   print(o)  
   print(hello.o) # 这个地方加不加模块名都是可以的  
   # 思考：当被导入的模块中存在大量的变量，有无简便的方法  
   print('分割线5----------------')  
   from folder.hello import \*  
   print(o)  
   print(h) # 单独跑这一段代码程序会报错，百思不得其解  
   # 这个方法并不推荐，推荐方法，在导入模块的首行添加\_\_all\_\_ = [‘’，‘’]里面存放导入的变量名称或函数名称，  
   # 用这种方法添加的叫内置变量或内置属性  
   print('分割线6----------------')  
   from folder import hello  
   print(o)  
   print(u)  
   print(h)  
   # 在\_\_all\_\_ = [‘’，‘’]中定义的会被打印，若未定义则会报错  
     
   # \_\_init\_\_.py作用:当导入包时,python会自动首先执行这个文件  
   # 包和模块不会被重复导入的,且操作时避免循环导入

# 第八章 python函数

## 1.认识函数

# 函数:分为自定义函数和内置函数 内置函数如:print(),round(),如何快速查看内置函数  
# 的功能可使用help(round)查看帮助文档  
# 其中,round(a,2)表示对变量a保留2位小数,并且会四舍五入  
a = 1.1314527  
result1 = round(a, 2)  
result2 = round(a, 4) # 会四舍五入  
print(result1, result2) # print打印多个变量时可采用此种方法,在同一行打印  
help(print) # 查看帮助文档  
# 函数的好处:1.功能性 2.隐藏细节 3.避免编写重复写代码

## 2.函数的定义及运行特点

# 自定义函数,函数需要先定义再调用  
# 函数基本结构,注:def是定义函数关键字,parameter\_list可省略,代码块可以使用return返回结果,若不加return则代表返回的是None  
# def funcname(parameter\_list):  
# pass  
# 1.实现两个数字的相加 2.打印输入参数  
# 遇到问题:代码报提示PEP:8 expected 2 blank lines ，found 1"  
# 解决办法:这句话的意思是“有两个空白行，但是没有发现。”,在声明函数的那一行的上方必须有两行的空行，否则便出现这个情况  
  
  
def add(x, y):  
 result = x + y  
 return result # 此处必须有return语句,否则这段代码没有意义  
# 调用函数  
  
  
# add(1, 2) # python传入参数的方式按照入参的顺序来,不会打乱  
# print(result) #这段代码会报错  
# 解决办法  
a = add(1, 2)  
print(a)  
  
  
# def print(code):  
# print(code) # 此处不需要return,因为本身就是打印,没必要使用return返回结果  
  
  
# print('Python') # 这段代码运行是会报错的,因为自己调用自己,递归次数最大987次,根本原因是定义了一个与内置函数一样的自定义函数  
# 解决办法,修改函数名即可  
def print\_code(code):  
 print(code)  
  
  
print\_code('Python')  
# 引入小知识点,如何设置最大的递归次数  
# import sys  
# sys.setrecursionlimit(10000)  
  
  
# 两段代码结合使用  
def add(x, y):  
 result = x + y  
 return result  
  
  
def print\_code(code):  
 print(code)  
  
  
a = add(1, 2)  
b = print\_code('Python')  
print(a, b)  
# 返回结果是Python  
# 3 None 思考为什么返回这个结果,先调用print\_code方法,里面有一个打印,所以先返回Python,随后a = 3,b没有return所以返回None  
# 以前print都是打印一个结果,其实print可以传多个参数,不换行打印,  
# 如:print(a,b,c ,d,.....)  
  
  
# 实验能否使用input()函数接收用户输入的变量,可以,用以下函数即可  
def add(x, y):  
 result = x + y  
 return result  
  
  
c = int(input('please input num x:'))  
d = int(input('please input num y:'))  
  
  
a = add(c, d)  
print(a)  
  
  
# return语句的补充说明,当函数代码内部一旦遇到return,return后面的语句将不会被执行  
def print\_code(code1, code2):  
 print(code1)  
 return  
 print(code2)  
  
  
print\_code('hello', 'world') # world不会被打印  
  
  
# return多个参数时,用逗号隔开即可  
def damage(skill1, skill2):  
 damage1 = skill1 \* 3  
 damage2 = skill2 \* 3 + 10  
 return damage1, damage2  
  
  
# 当你用一个变量接收到元组之后,如何使用结果  
def damage(skill1, skill2):  
 damage1 = skill1 \* 3  
 damage2 = skill2 \* 3 + 10  
 return damage1, damage2  
  
  
damages = damage(2, 3)  
print(type(damages)) # <class 'tuple'>  
print(damages[0], damages[1]) # 这种依靠序号来访问变量的方法是非常非常不推荐的  
  
  
# 可以采用序列解包的方式  
def damage(skill1, skill2):  
 damage1 = skill1 \* 3  
 damage2 = skill2 \* 3 + 10  
 return damage1, damage2  
  
  
damages\_sill1, damages\_skill2 = damage(2, 3)  
print(damages\_sill1, damages\_skill2)  
  
  
# 序列解包  
a = 1  
b = 2  
c = 3  
# 以上代码可以简化成如下代码  
a, b, c = 1, 2, 3  
# 以上代码是否可以简化成一行,是可以的  
d = 1, 2, 3  
print(type(d)) # tuple  
a, b, c = d # 这就叫一个序列解包  
# 注:序列解包时元素要相等  
a = 1  
b = 1  
c = 1  
# 简化 a,b,c = 1 ,还可以简化成以下代码  
a = b = c = 1  
  
"""  
函数的参数列表种类  
1.必须参数:在参数列表中定义的参数,在调用时必须传入参数  
2.关键字参数:可以告诉函数明确指定参数值给哪个参数,无序,意义在于方便,增强代码的可读性  
3.默认参数:  
必须参数与关键字参数的区别在于函数的调用而不在于函数的定义上  
函数在定义时传入的参数叫做形参,在调用时传入的参数叫做实参  
"""  
  
  
def add(x, y): # x,y叫做形参  
 result = x + y  
 return result  
  
  
a = add(1, 2) # 1,2叫做实参  
print(a)  
b = add(y=3, x=4) # 关键字参数  
print(b)  
  
  
# 若参数列表需要传递15个参数,引入默认参数(实际操作中不可取,可将15个参数封装成对象,此处仅举例)引出默认参数  
def print\_student\_file(name, age=7, sex='男', college='人民小学'):  
 print('我叫'+name)  
 print('我今年'+str(age)+'岁')  
 print('我是'+sex+'生')  
 print('我在'+college+'上学')  
  
  
print\_student\_file('小张')  
print('------------------')  
# 只改变年龄  
print\_student\_file('小胡', age=6)  
# 改变年龄和性别,并且不按照默认顺序传递,是否可行,实验证明不可行  
# print\_student\_file('小胡', '女', 7)  
print('------------------')  
print\_student\_file('小胡', 7, '女',)  
print('------------------')  
print\_student\_file('小胡', 7, sex='女',)  
# print\_student\_file('小胡', age=7, '女', college = '光明小学') 代码报错,因为在调用时同样不能非默认参数在默认参数后面  
"""  
默认参数用法总结  
1.在定义的时候不能将非默认参数放在默认参数后面  
比如:def print\_student\_file(name, age=7, sex='男', college='人民小学',teacher)这种写法是错误的  
2.在调用的时候必须参数必须要传,默认参数可以不传  
3.在调用时若不使用关键字参数,必须按照默认参数顺序传递参数,是按照形参和实参一一对应的  
4.在调用时可以使用关键字参数,可以不遵守顺序  
5.在调用时同样不能非默认参数在默认参数后面  
比如:print\_student\_file('小胡', age=7, '女', college = '光明小学')  
"""

第九章

*"""  
第1课  
面向对象最核心的概念 1.类 2对象  
类名的命名规则:与变量命名稍微有点不同,1.类名的第一个字母最好大写 2.两个组成的首字母大写  
变量命名的建议最好是小写,两个单词时建议使用下划线连接  
在类的内部我们可以做哪些事情 1.定义若干个变量 2.定义函数  
"""*# 定义一个最基本的类,类的关键字class  
class Student():  
 name = ''  
 age = 0  
  
 def print\_file(self): # 此处代码不加self程序会报错  
 pass  
  
  
student = Student() # 变量student接收实例化的结果  
student.print\_file()  
  
  
# 思考:如何调用这个类  
# 解决办法:给类实例化 student = Student()  
# 思考:如何调用类下面的方法  
# 解决办法:实例化变量.方法名 student.print\_file()  
class Student():  
 age = 0  
 name = '西'  
  
 def student\_information(self):  
 print(self.name)  
 print(str(self.age))  
  
  
student = Student()  
student.student\_information() # 注:此处一定要用实例化后的变量来调用方法  
  
  
  
"""  
第2课  
总结:类的最基本概念就是封装一些变量和函数  
在一个模块中可以定义很多个类,自己举例  
类下面的函数叫方法,类下面的方法和变量一定要加self,否则程序会报错  
不要在类的内部调用,类的调用一定要放在类的外部,类负责定义,描述,在做项目的时候推荐类的定义和调用放在不同的模块  
如何在另一个模块中调用类,先import后实例化成对象再调用方法  
from 9c import Student  
student = Student()  
student.student\_information()  
函数 VS 方法 在模块中定义的函数叫函数,在类中定义的函数叫方法  
方法和函数的区别:方法设计层面,函数:程序运行,过程式的一种称谓  
变量定义在模块中叫变量,在类中定义叫数据成员  
  
"""  
  
"""  
第3课  
类和对象的关系  
类和对象的关系通过实例化关联在一起的  
类:行为和特征  
学生类 name,age等  
name和age就是特征 数据成员  
行为 dohomework 方法  
当类被实例化后变成一个具体的对象  
"""  
  
  
"""  
第4课 构造函数  
  
"""  
  
  
# 思考:以下三个实例化对象是否相同  
# 解答:从我们的角度或正常的角度来看,这三个对象是一样的,但从机器内部来看是不  
# 相同的,可通过验证这三个对象的内存地址来区分  
class Student():  
 age = 0  
 name = ''  
  
 def do\_homework(self):  
 print('homework')  
  
  
student1 = Student()  
student2 = Student()  
student3 = Student()  
print(id(student1))  
print(id(student2))  
print(id(student3))  
  
  
# 思考:怎样实例化几个从我们的角度来看不相同的对象,此处引入构造函数概念  
class Student():  
 name = ''  
 age = 0  
  
 def \_\_init\_\_(self, name, age): # 这个函数的名字是固定的 构造函数  
 print('我的名字是:'+name)  
 print('我的年龄是:' + str(age))  
  
 def do\_homework(self):  
 print('homework')  
  
  
student1 = Student('小白', 12)  
student2 = Student('小红', 45)  
student3 = Student('小黄', 34)  
print(id(student1))  
print(id(student2))  
print(id(student3))  
# 注:构造函数的调用是自动执行的,所以不需要再次通过实例化对象.构造函数来调用  
  
  
# 假如我们通过构造函数来调用,那么返回结果又是什么  
class Student():  
  
 def \_\_init\_\_(self): # 这个函数的名字是固定的 构造函数  
 print('我的名字是:')  
 print('我的年龄是:')  
  
 def do\_homework(self):  
 print('homework')  
  
  
student1 = Student()  
a = student1.\_\_init\_\_()  
print(a) # None  
print(type(a)) # <class 'NoneType'>  
# 返回结果的原因:和之前说的一样,没有return语句所以返回为None  
  
  
# 思考:我们是否可以为构造函数返回一个值  
class Student():  
  
 def \_\_init\_\_(self): # 这个函数的名字是固定的 构造函数  
 print('我的名字是:')  
 return 'result'  
  
 def do\_homework(self):  
 print('homework')  
  
  
student1 = Student()  
a = student1.\_\_init\_\_()  
print(a)  
print(type(a)) # TypeError: \_\_init\_\_() should return None, not 'str'  
# 结论:不能为构造函数指定非None的返回值,与普通函数的区别  
  
  
class Student():  
 name = ''  
 age = 0  
  
 def \_\_init\_\_(self, name, age): # 这个函数的名字是固定的 构造函数  
 # 在函数内部初始化对象的属性  
 name = name # 左边的name代表对象的属性,右边的name代表函数的入参  
 age = age  
 print('我的名字是:'+name)  
 print('我的年龄是:' + str(age))  
  
 def do\_homework(self):  
 print('homework')  
  
  
student1 = Student('小白', 12)  
print(student1.name) # 这句并没有被打印,思考为什么  
# 说明我们在构造函数中的赋值并没有改变name变量的取值,原因:当局部变量与全局变量同名时,局部变量并不会覆盖全局变量  
# 如果这样理解就错了,我们决不能将模块中的全局变量与局部变量之间的关系等同于类下面的变量  
# 打印空值真正的原因:类变量和实例变量  
"""  
类变量和实例变量  
"""