**Day05笔记**

day05(上午)：

1.for ... in循环：

格式：

**for 变量 in 容器对象:**

语句块

执行流程：

遇到for关键字将容器对象中的第一个元素给到变量，执行语句块，

再将容器对象中的第二个元素给到变量，执行语句块，

以此类推...

直到容器对象中最后一个元素也被获取过了，执行完语句块，整个for...in循环就结束了

两种使用场景：

场景一：配合range对象使用

**range对象：**

1).通过执行完毕range()函数返回给程序一个range对象，

此对象是一个惰性序列对象(显示不清晰)，可以使用遍历的思想去查看其内容，

也可以将其转换为list、tuple等类型再进行运算；

2).它仍然是一个容器对象，有容量/长度，有索引(index)，

可以通过变量名配合下标访问容器中的元素

格式：

**range([start],end,[step])**

解析参数：

**start：起始数据，默认从0开始**

end：结尾数据

step：步长

特点：含头不含尾 --> [start,end)

代码如下：

r = range(1,5)

print(r,type(r),len(r))

r = range(10)

**#遍历r对象**

**for i in r:**

**print(i,end=" ")**

**需求：打印1~100之间的奇数，得到个数和总和，使用for...in来实现**mySum = 0  
myCount = 0  
*# 方式一：  
# for i in range(1,100,2):  
# print(i)  
# myCount += 1  
# mySum += i  
#方式二：***for** i **in** range(1,100):  
 **if** i % 2 == 1:  
 print(i)  
 myCount += 1  
 mySum += i  
print(**'1~100之间奇数的个数为：%d'** %myCount)  
print(**'1~100之间奇数的总和为：%d'** %mySum)

也可以将range对象转换为其它的容器对象(例如：list)，然后在执行遍历操作：

代码如下：

r = range(1,11)

print(r,type(r),len(r))

lt = list(r)

print(lt,type(lt),len(lt))

#使用while循环遍历range对象：

i = 0

while i < len(r):

print(r[i])

i += 1

场景二：配合容器对象(str、list、tuple、dict、set)使用

#配合list&str类型使用：

lt = [10,20,30,40,50]

for i in lt:

print(i,end=" ")

print() 得到结果*# 10 20 30 40 50*

str1 = "abcdefg12345$%!"

for j in str1:

print(j,end=" ")

print() 得到结果*# a b c d e f g 1 2 3 4 5 $ % !*

#配合dict对象使用：

dic1 = {'name':'韩梅梅','age':25,'height':165.0}

#通过键找到值

for k in dic1:

print(k + "-->" + str(dic1.get(k)))

**items()：此函数属于dict中独有的函数，一旦执行完毕，返回给程序一个dict\_items类型的对象(容器类型，有容量)，  
可以配合forin循环一下子得到key，value...**

di = dic1.items()

print(di,type(di),len(di)) 得到结果*dict\_items([('name', '韩梅梅'),*

*('age', 25), ('height', 165.0)])*

*<class 'dict\_items'> 3*

for k,v in dic1.items():

print("k=",k,"v=",v)

dic2 = {"坦克":['程咬金','白起','项羽'],'刺客':['阿珂','李白','兰陵王'],'射手':['后裔','鲁班','狄仁杰']}

for k,v in dic2.items():

print("k=", k, "v=", v)

for vv in v:

print(vv)

*演示break&continue关键字，else语句配合forin使用***for** i **in** range(1,11):  
 **if** i == 5:  
 *#break* **continue** print(i)  
**else**:  
 print(**'没有遇到牛逼的break，所以我被执行了...'**)

2.嵌套循环：

循环中再定义循环，就是嵌套循环；

【注意】：嵌套循环不是两层循环，两层循环是嵌套循环，在之后开发过程中，99%的情况两层就够了；不应该出先3，4层

格式模板：

while 中嵌套 while

while 中嵌套 forin

forin 中嵌套 while

forin 中嵌套 forin 多

格式：

for i in seq1: 外层循环

语句块1

for j in seq2: 内层循环

语句块2 内存循环的循环体

例如：

for i in range(2):

print('我是外层循环体...')

for j in range(2):

print('我是内层循环体...')

print('i=',i,',j=',j)

执行流程：

先遇到外层循环，将外层循环range对象中的第一个元素0取出给到变量i，

立即进入到外层循环内部，执行'我是外层循环体'，遇到了内存循环，将内存循环range对象中的第一个元素0取出给到变量j，

立即进入到内存循环内部，执行'我是内存循环体'，输入此时的i=0,j=0；内存循环第一次结束了，立即迭代，

将内存循环range对象中的第二个元素1取出给到变量j，

立即进入到内存循环内部，执行'我是内存循环体'，输入此时的i=0,j=1；内存循环第二次结束了，立即迭代，由于没有元素了，

内存循环真正意义上就结束了；然后可以理解为外层循环的第一次结束了，迭代外层循环，

将外层循环range对象中的第二个元素1取出给到变量i，

立即进入到外层循环内部，执行'我是外层循环体'，遇到了内存循环，将内存循环range对象中的第一个元素0取出给到变量j，

立即进入到内存循环内部，执行'我是内存循环体'，输入此时的i=1,j=0；内存循环第一次结束了，立即迭代，

将内存循环range对象中的第二个元素1取出给到变量j，

立即进入到内存循环内部，执行'我是内存循环体'，输入此时的i=1,j=1；内存循环第二次结束了，立即迭代，由于没有元素了，

内存循环真正意义上就结束了；然后可以理解为外层循环的第二次结束了，尝试迭代外层循环，由于没有元素了，整个嵌套循环结束了

总结：

1).外层循环执行一次，内层循环全部执行一遍

2).如果外层循环需要执行m次，内存循环需要执行n次，那么整个嵌套循环一共会执行m \* n次

-----------------------------------------------------------------------------------------------

day05(下午)：

1.排序

分类：升序： 降序：

使用选择排序来实现对列表中的元素进行升序排列

详细思路以及代码实现参看老郭图解

**例题：选择排序(升序)代码实现**lt = [29,35,18,-11,-9,100,0,56]  
print(**'排序前：%s'** %lt)  
*#得到列表lt的长度*length = len(lt)  
*#外层循环控制轮数，i可以认为是选中空间或者点***for** i **in** range(length - 1):  
 *#内层循环控制每一轮执行的次数，j可以认为是比较空间或者箭头* **for** j **in** range(i + 1,length):  
 *#比较两个空间中的数据大小* **if** lt[i] > lt[j]:  
 *# temp = lt[i]  
 # lt[i] = lt[j]  
 # lt[j] = temp* lt[i],lt[j] = lt[j],lt[i]  
print(**'排序后：%s'** %lt)

2.**函数/方法(method、function)：**

2.1.什么是函数？

它是一段功能代码，理解为一种功能行为，在内存中有空间区域，函数需要被调用才能执行(通过函数名来调用)；

好处？

1).提高代码的复用性

2).提升代码的阅读性

3).增加代码的扩展性

4).增强了代码的维护性

函数有五要素组成：

①.函数修饰符 关键字def开头

②.函数返回值类型 有返回值(类型：int、float、str、list...)/没有返回值

③.函数名 函数的名字(标识符一种，满足规则和规范)

④.形参列表 定义在小括号内(可能有形参，也可能没有)

⑤.函数体 一段业务逻辑代码

定义格式:

① ③(④):

⑤

函数名的定义：

标识符的一种，满足规则和规范：

对于其规范，和变量名保持一致

*演示自定义函数的返回值问题*

**以下函数有两个形参，没有返回值**

**def** mySum(num1,num2):  
 print(num1 + num2)

*#调用自定义函数，完成求和操作*

mySum(100,200)

**以下函数有两个形参，有返回值**

**def** mySum(num1,num2):  
 result = num1 + num2  
 **return** result

*#调用自定义函数，完成求和操作*

result = mySum(10,20)  
print(**'两个整数的和值为：%d'** %result)

2.2.计算机的内存组成：(五块区域)

栈内存：

特点：满足先进后出，分为栈顶和栈底两部分，函数要被执行一定要在栈顶开辟空间；

一旦一个函数(func1)被调用了，那么就会进栈(入栈)，如果运行到一半又调用了别的函数(func2)，

那么func1就会被压栈，而func2会被进栈，此时func2一通执行直至结束，那么func2就被弹栈(出栈)，

而func1就升栈了继续执行，直至也运行完毕，然后弹栈；

【注意事项】：

1).形式参数(形参)：它可以认为是一个入口(在函数名后面的小括号中定义)，并不存在实际数据，在代码的体现上就是定义变量，等着接受实参；

2).实际参数(实参)：实实在在存在的数据(可以是各种类型)，将实际参数传递给形式参数(过程：称为参数传递)，

之后在函数体中参数运算的都是实际参数

3).返回值：在整个函数执行完毕后需不需要返回给程序一些数据(分类：有返回值/没有返回值)

如果有返回值：需要使用关键字return进行数据的返回

return关键字的作用：

1).将需要返回的数据给到函数的调用处(调用者)

2).表示函数的结束

**4).和return关键字在同一作用域，在其后面不要去定义任何的代码，因为永远不可能被执行到**

**5).return关键字后面也可以不定义任何的内容，它的作用仅仅意味着函数结束了...**

2.3.自定义函数的四种定义场景：

1).无参无返回值

2).无参有返回值

3).有参无返回值

4).有参有返回值 *演示自定义函数的四种场景：  
#无参无返回值***def** func1():  
 print(**'hello method...'**)  
*#无参有返回值***def** func2():  
 **return True***#有参无返回值***def** func3(a,b):  
 print(a + b)  
*#有参有返回值***def** func4(a,b):  
 *#在自定义函数中调用本模块(.py文件)别的自定义函数* func1()  
 **return** a **if** a > b **else** b  
*#调用自定义函数*print(func2()）  
func3(50,30)  
myMax = func4(10,-20)  
print(**'较大值为：%d'** %myMax)

2.4.形参列表的多种定义情况：

*演示形参的多种定义情况：  
#默认参数的使用：***def** func1(name=**"张三"**,age=18,height=175.5):  
 print(name,age,height)  
*#调用自定义函数(不传递参数)：会使用默认参数参与之后的运算*func1()  
*#调用自定义函数(传递部分参数)：传递的部分使用传递内容，其余部分使用默认参数参与运算*func1(**"李四"**)  
*#一般参数的使用：(最多)***def** func2(name,age,address):  
 print(name,age,address)  
*#调用自定义函数：必须传递参数，而且位置和个数都要对的上*func2(**"李雷"**,21,**"北京"**)  
*#关键字参数的使用：  
#好处：不用记住每个变量的位置***def** func3(name,age,address):  
 print(name,age,address)  
*#体现在调用自定义函数上：传递类型键值对的组合*func3(address=**"上海"**,name=**"韩梅梅"**,age=20) **可变参数的使用：  
在定义一项功能时，对于参数层面(到底几个不确定)，此时可以使用可变参数来实现  
参数解释：  
\*args：接受0~无穷多个单值，存入到一个元祖中返回  
\*\*kwargs：接受0~无穷多个键值对组合，存入到一个字典中返回  
def** func4(\*args,\*\*kwargs):  
 print(args,type(args))  
 print(kwargs,type(kwargs))  
func4() **需求：实现多个整数求和的操作  
自定义函数：  
两个原则：  
1).有没有形参？  
有，但是个数不确定，使用可变参数  
2).有没有返回值？  
可有可无；  
def** mySum(\*args):  
 *# i = 0* sum1 = 0  
 *# while i < len(args):  
 # sum1 += args[i]  
 # i += 1* **for** i **in** args:  
 sum1 += i  
 **return** sum1  
mySum = mySum(1,2,3,4,5)  
print(**'和值为：%d'** %mySum)  
**def** func1(\*args,\*\*kwargs):  
 print(args)  
 print(kwargs)  
func1(0,100,3.14,**True**,**"abcd"**,[10,20,30],name=**"班长"**,age=25)  
**以下代码会报错：  
因为\*args定义在前，\*\*kwargs定义在后，由于在传递参数的过程中已经出现了类似  
name="班长",age=25这样的代码，那么其后就不能出现单值数据的传递了；***# func1(0,100,3.14,True,"abcd",[10,20,30],name="班长",age=25,-100,3.14)*