**目录**

[串讲目录 1](#_Toc22684)

[第一阶段 2](#_Toc26550)

[VIM 编辑文件 2](#_Toc31771)

[linux 目录分支结构 2](#_Toc189)

[linux 常用命令 （面试） 3](#_Toc22678)

[权限 （面试） 7](#_Toc24131)

[shell （面试） 9](#_Toc10814)

[全局变量和局部变量的修改 12](#_Toc24882)

[私有属性和私有方法 13](#_Toc5592)

[01. 应用场景及定义方式 13](#_Toc11473)

[02. 私有属性和私有方法（科普） 14](#_Toc17621)

[类属性和类方法 14](#_Toc13039)

[目标 14](#_Toc8034)

[01. 类的结构 14](#_Toc27188)

[02. 类属性和实例属性 16](#_Toc31730)

[03. 类方法和静态方法 17](#_Toc18439)

[闭包的原理以及应用 21](#_Toc6153)

[python的闭包 21](#_Toc8886)

[迭代器和生成器 23](#_Toc3142)

[Iterator 迭代器 23](#_Toc1469)

[生成器函数 25](#_Toc15889)

[python 单例模式 工厂模式 27](#_Toc4446)

[一、单例模式 27](#_Toc9037)

[二、工厂模式 28](#_Toc2027)

# 第一阶段

## VIM 编辑文件

vim的三种工作模式！  
 1、浏览模式 ：启动时进入,不能编辑文本  
 2、插入模式(a/i/o) ：编辑文本  
 3、命令行模式(ESC -> shift + :) ：保存退出..  
以下操作均在命令模式进行:  
 yy 复制  
 p 粘贴 100p 粘贴100行  
 u 撤销上次的编辑  
 shit + : 进入命令行状态 可以 w保存 或者 wq保存并退出  
 ctrl +v: 进入可视化模式,选中 I 按下建选中所有I 按D进行删除  
 dd: 删除 100dd 删除100行  
 批量选择:  
 esc > ctrl + v > 移动方向  
 批量删除:  
 esc > ctrl+v > 下键选中注释区域 > d  
 批量注释:  
 esc > ctrl+v > 下键选中注释区域 > shift+i(进入插入模式) > # > esc建键  
 查看行号：  
 esc > :set nu

## linux 目录分支结构

/: 根目录 所有文件都在根目录下.  
管理目录:  
 /boot: 用来存放内核以及系统引导文件  
 /bin: 用来存放Linux的工具,例如:在终端里面输入的 ls  
 /sbin: 大多数涉及系统管理的命令参数,是超级用户权限root的可执行命令.普通用户没有权限访问  
 /log: 存放系统的日志,数据文件,邮箱等  
 /etc: 用来存放系统的配置文件  
 /dev: 用来存放设备文件 [U盘,鼠标,硬盘]  
 /mnt: 用来存放系统挂在文件的  
用户目录:  
 /root: 超级管理员root的家目录  
 /home: 普通用户的家目录  
应用程序目录:  
 /lib: 主要用来存放系统动态库文件  
 /tmp: 临时目录,系统会定期清理  
 /usr: 存放一些不适合放在 /bin 或者 /etc 的目录里面  
 /usr/bin: 用来存放程序文件  
 /opt: 存放可选程序的文件

## linux 常用命令 （面试）

目录:  
 [root@localhost ~]#  
 root: 当前用户名  
 localhost: 主机名  
 ~ : 当前工作目录 加目录  
 # : 管理员用户  
 $ : 普通用户  
命令格式:  
 命令字 选项 参数1 参数2  
1、pwd ：显示当前所在路径  
2、ls ：显示目录的详细内容(文件、子目录)  
 命令:  
 ls --help 查看帮助信息  
 -A 显示隐藏文件(不包含.和..)  
 -a 显示隐藏文件(包含.和..)  
 -l 查看文件属性  
 -d 显示目录的详细信息  
 -R 递归显示  
 -i 查看文件节点信息  
3、路径  
 1、绝对路径: 以 / 开始的路径  
 2、相对路径: 不以 / 开始的路径  
 . 当前目录  
 .. 上一级目录  
 ~ 用户主目录/家目录(/home/tarena)  
 3、主目录/家目录  
 1、超级用户 ：root  
 2、创建用户时会自动在/home下创建1个和用户名同名的目录,为此用户主目录  
 用户名 ：tarena 主目录 ：/home/tarena  
4、cd ：切换目录(路径)  
 1、用法  
 cd 路径(绝对、相对路径都可以)  
 cd /home/tarena 、cd /home/tarena/aid1809  
 cd aid1905 #前提是现在主目录下  
 2、cd 直接回车 ：回到用户主目录/home/tarena  
 cd ~ : 回到用户主目录  
 cd - : 从最近访问的两个目录之间来回切换  
 3、TAB键自动补齐命令/路径(熟练使用!!!!!)  
 1、速度快  
 2、路径不会错,TAB不出来说明路径有问题  
 4、练习  
 1、切换到目录 ：/home/tarena/anaconda3/bin  
 cd /home/tarena/anaconda3/bin/  
 2、查看当前所在路径  
 pwd  
 3、以列表形式列出当前目录所有内容(包含隐藏)  
 ls -la  
 4、切换到当前目录的上一级目录,即  
 ：/home/tarena/anaconda3  
 cd ..  
 5、切换到/etc目录,查看/etc下有哪些内容  
 cd /etc/  
 6、直接切换到用户主目录  
 cd  
5、tab健自动补全 命令字/选项/参数/文件路径/软件名/服务名  
6、终端快捷键  
 Ctrl  
 Ctrl + l 清空当前屏幕  
 Ctrl + u 清空至行首  
 Ctrl + w 删除一个单词  
 Ctrl + c 废弃当前编辑的命令  
 Ctrl + z 终止  
7、mkdir 创建目录/文档  
 在当前目录下创建1000个目录 admin0-1000  
 mkdir admin{1..1000}  
 常用选项：  
 -p ：逐层创建目录  
 mkdir -p One/Two/Three/Go  
 # 如果中间目录不存在,直接创建  
8、touch  
 1、作用  
 1、文件不存在 -> 创建  
 2、文件存在 -> 用系统时间更新文件修改时间  
 2、格式 ：touch 文件名1 文件名2 文件名3 ...  
 3、练习  
 1、在用户主目录下创建目录 A1/B1/C1/D1  
 cd  
 mkdir -p A1/B1/C1/D1  
 2、在D1下创建文件 file1.txt file2.txt  
 cd A1/B1/C1/D1  
 touch file1.txt file2.txt  
 3、在C1下创建文件 file3.txt file4.txt  
 cd ..  
 touch file3.txt file4.txt  
 4、切换到A1目录,查看当前所在路径  
 cd ../../  
 pwd  
 5、直接切换到用户主目录  
 cd  
\* 9、rm ：删除文件/目录  
 1、rm 选项 文件/目录  
 -r ：删除全部文件/目录  
 -f ：强制删除,不给任何提示  
 -i ：删除前给提示(y代表yes,n代表no)  
 rm -rf 文件名/目录名  
 删除根目录下的/a.txt /b.txt  
 rm -rf /a.txt /b.txt  
 rm -r admin{1..1000}  
 \* 匹配所有  
 ^a 匹配以a开头的文件  
 ! 取反  
 创建a{1..1000}的文件,删除除了a50以外的所有文件  
 /log/201801.sql.log 删除除了2018年1月份意外的所有文件  
 rm -rf 201802..sql.log ... 一个一个删除  
 rm -rf !(201801.sql.log)  
 / a{1..100} 不要执行rm -rf !(a50) # 防止其他文件删除  
 删除除了a10和a20以外的文件  
 rm -rf !(a10|a20)  
10、> 重定向 # 会覆盖已有的内容  
 echo 13087508351 > day1  
 >> 追加重定向 # 不会覆盖已有的内容,将插入的内容插至末行  
 仅适用与追加一行  
 echo 13087508351 >> day1  
11、cat 查看制定文件里的内容  
 把/etc下的 passwd 重定向到 /root/pass  
 cat /etc/passwd > /root/pass  
 把/etc下的 gshadow 追加重定向到 /root/pass  
 cat /etc/gshadow > /root/pass  
  
 echo 123456 > a  
 echo 789456 > b  
 cat a >> b # 把a里面的文档内容追加到b里面,a的文档仍在  
12、cp 复制  
 cp /源文件路径/名称 /目标文件路径/[名称] # 不设置名称默认为原文件名  
 cp -p 复制的时候保证源文件权限不变  
 cp -rp /etc/\* /opt/ (递归复制目录及其子目录内的所有内容)  
13、mv 剪切  
 在相同路径下面属性重命名  
 把 /etc/gshadow 复制到 /opt  
 cp /etc/gshadow /opt/  
 把 /opt/gshadow 移动到 /root/gsha  
 mv /opt/gshadow /root/gsha  
 把/root/gsha 重命名 test  
 mv /root/gsha /root/test  
14、文件查找  
 less test 以行显示   
 more test 以页显示  
 head -5 test # 查看前5行  
 tail -5 test # 查看后5行  
 显示后10行的前5行  
 tail -10 Linux/day01.txt | head -5  
15、which 查找命令路径  
 [root@localhost ~]# which ls  
 alias ls='ls --color=auto'  
 /usr/bin/ls  
16、find 查找文件  
 find 路径 选项1 选项2  
 find -name # 按照文件名查找  
 find -iname # 按照文件名(匹配大小写)查找  
 find -type # 按类型查找  
 find -mtime # 按时间查找  
 find -user # 按用户查找  
 find -group # 按照组名查找  
 find -size # 按照大小查找 +4G -4G 4G(大于,小于,等于)  
 -a   
 -or   
 -inum 以i节点信息去查找  
 例:  
 find / -name a # 查找/目录下 a开始的文件  
 find / -name a -type f文件 | d目录   
 find -size +4G  
 find -mtime +10 # 10天前的文件  
 find / -user student  
17、grep 查找文件内容  
 -v 取反  
 -o root 只显示root  
 grep root /etc/passwd # 显示包含root的信息  
 grep ^root /etc/passwd # 显示以root开头的信息  
 grep root$ /etc/passwd # 显示以root结尾的信息  
 grep -o root /etc/passwd # 查找出来的root 一行一行显示出来  
 grep -v root /etc/passwd # 查找不包含root的信息  
 grep 5$ a.txt # 显示以5结尾的信息   
18、alias 别名: 帮助我们简化命令(默认本次有效)  
 unalias -a: 取消别名(没有-a,默认取消所有别名)  
 ssh: 远程 ssh root@192.168.4.5  
 永久性设置别名  
 vim /etc/bashrc   
 第二行  
 alias goa="ssh -X root@192.168.4.5"  
 alias hang="init 0"  
 bash # 刷新  
19、hostname 查看主机名  
 hostname name # 设置主机名  
 vim /etc/hostname里面更改  
 reboot #重启后主机名生效  
20、linux tty终端 ctrl + alt + F(1-5) # 切换终端  
 0: halt 关机  
 1: 单用户模式,单用户模式只有系统管理员才可以进入  
 2: 不具备网络文件系统功能的多用户命令行模式.  
 3: 具有网络文件系统功能的多用户命令行模式  
 4: 保留使用  
 5: 具备网络功能的xwindows模式.系统初始化直接进入图形界面.适合初学者使用  
 6: 关闭所有运行的程序并重新启动  
 halt 关机  
 init 0 关机 init (1-6)  
21、tar ：打包并压缩  
 1、格式  
 tar -zcvf 压缩包名字.tar.gz 需要压缩文件/目录  
 -z ：用gzip对包进行压缩  
 -c ：创建包(create)  
 -v ：显示明细(verbose)  
 -f ：file  
 2、解压缩  
 tar -zxvf 压缩包名字.tar.gz -C 路径  
 -C ：指定解压路径,不写默认解压到当前

## 权限 （面试）

基本权限: r,w,x 读写执行(针对: 数主,属组,其他人)  
对于文件:  
 读取: r read ==> cat head tali less more   
 修改: w write ==> echo vim > >>   
 执行: x ==> 是否可以被执行 ./  
对于目录:  
 读取: r read ==> ls   
 修改: w write ==> mkdir touch mv cp rm  
 执行: x ==> cd 是否可以进入  
特点:  
 只要对一个目录有w权限,那就可以删除这个目录下的任何文件  
权限的指令:  
 ls -l 文档  
 drwxr-xr-x. 2 root root 6 3月 29 11:07 模板  
 drwxr-xr-x. 2 root root 6 3月 29 11:07 音乐  
 ls -lh 文档 # 查看大小  
 -rw-r--r--. 1 root root 2.4K 3月 29 11:40 /etc/passwd  
更改权限:  
 chmod -R 权限字符串 文档  
 chmod 归属关系+-= 权限类型 文档  
 chmod u+x, g+x, o+w 文档 # 用户加权限,归属加权限,其他加权限  
 chomd a+x all 文档  
 chmod 755 文档  
 所属者 所属组 其他人  
 4 2 1 ===> r w x  
 文档的归属关系:  
 属主(user): 这个文件属于哪个用户  
 属组(group): 这个文件属于哪个组  
 其他人(othor): 除了属主,属组以外的任何用户  
 所有人(all): 所有用户 chmod a+x all 文档  
 在/opt/下面创建 pass 文件权限是:700   
 mkdir /opt/pass chmod 700 pass  
 在/opt/下面创建 touch a.txt b.txt c.txt d.txt e.txt  
 a.txt 755 chmod 755 a.txt   
 b.txt 477 chmod 477 b.txt   
 c.txt 222 chmod 222 c.txt   
 d.txt 444 chmod 444 d.txt   
 e.txt 111 chmod 111 e.txt   
修改所属主,属组 chown  
 chown -R 属主 文档 chown student a.txt  
 chown -R :属组 chown :hng b.txt  
 chown -R 属主:属组 文档 chown student:hang c.txt  
练习  
 1、在用户主目录下创建文件 ：密函.txt  
 touch 密函.txt  
 2、用 vi 在文件中写入：  
 我的密函  
 回眸一笑百媚生,六宫粉黛无颜色  
 春宵苦短日高起,从此君王不早朝  
 3、查看 密函.txt 权限  
 ls -l 密函.txt  
 4、将文件权限设置为自己可读可写可执行,其他人任何权限没有  
 chmod 700 密函.txt  
 5、在4的基础上将权限设置为同组用户只读  
 chmod 740 密函.txt  
 chmod g+r 密函.txt  
 6、将文件的可执行权限给去掉  
 chmod -x 密函.txt  
 7、将 密函.txt 复制到用户主目录下的AID09中  
 cp 密函.txt AID09/

## shell （面试）

##################################################  
$0 当前的脚本名称  
$n 传递给脚本或者函数的参数 n 代表的是一个数字。  
$# 传递给脚本或函数参数的个数  
$? 返回上个命令的状态,或者返回函数的值  
##################################################  
判断关系  
-eq 等于  
-ne 不等于  
-gt 大于  
-ge 大于或等于  
-lt 小于  
-le 小于等于  
##################################################  
文件比较运算 若存在则为真  
-e 判断对象是否存在  
-d 判断对象是否为目录  
-f 判断对象是否为文件  
-r 判断对象是否有读的权限  
-w 判断对象是否有可写的权限  
-x 对象是否有可执行权限  
##################################################  
&& 给定的条件必须都得成立,整个测试结果为真  
|| 只有其中的一个条件成立.则结测试结果为真  
判断 /etc/ 是否存在  
[ -e "/etc/" ] && echo "存在" || echo "不存在"  
###############################################  
[root@host1 ~]#  
 [ -e "/etc/" ] && echo "存在" || echo "不存在"  
存在  
[root@host1 ~]#   
[ -e "/opt/" ] && echo "存在" || echo "不存在"  
存在  
[root@host1 ~]#   
[ -e "/chendan" ] && echo "存在" || echo "不存在"  
不存在  
判断 ls mkdir cd 这些命令是否存在  
[ -e "cd" ] && echo "存在" || echo "不存在"  
[root@host1 ~]# which mkdir   
/usr/bin/mkdir  
[root@host1 ~]# [ -e "/usr/bin/mkdir" ] && echo " 存在" || echo "不存在"  
####################################  
[root@host1 ~]# mkdir test  
[root@host1 ~]# chmod 444 test  
[root@host1 ~]# [ -w "/test" ] && echo "可读"   
|| echo "不可写"  
不可写  
#######################################  
if单分支语法组成  
if 条件测试  
then  
命令序列  
fi  
################  
if 双if分支  
if   
then  
命令序列1  
else  
命令序列2  
fi  
##################  
if 多分支  
if 条件测试1 ;then  
命令序列1  
elif 条件测试1 ;then  
命令序列2  
else  
命令序列N  
fi  
-c 可以发送的包  
-i 0.2 缩短发送测试包的间隔秒数  
-w 等待反馈的超时秒数  
ping -c 3 -i 0.2 -W 1 192.168.4.5  
检测输入IP地址的连通性  
#!/bin/bash  
a=192.168.4.  
ping -c 3 -i 0.2 -W 1 $1 > /dev/null  
if [ $? -eq 0 ]; then  
 echo "Host $1 is ip"  
else  
 echo "Host $1 is down"  
fi  
[root@room9pc01 ~]# chmod 777 ping.sh   
[root@room9pc01 ~]# ./ping.sh 192.168.4.5  
Host 192.168.4.5 is ip  
[root@room9pc01 ~]# ./ping.sh 192.168.4.8  
Host 192.168.4.8 is down  
#################################  
从键盘读取一个数,判断他的等级  
大于等于90神功盖世  
大于等于80,小于90 登峰造极  
大于等于70,小于80 炉火纯青  
大于等于60,小于70 略有小成  
小于60 初学扎到  
###############################  
for 变量名 in 值列表  
do  
 命令序列  
done  
################################  
创建20个用户,admin1-20  
密码统一为：redhat  
#!/bin/bash  
pass=redhat  
user=admin  
for i in {1..20}  
do  
 useradd $user$i >/dev/null  
 echo $pass | passwd --stdin $user$i >/dev/null  
done  
##################################  
从键盘上输入用户名  
从键盘上读取密码  
从键盘上读取创建个数  
###################################  
检测192.168.4.0网段的存活主机？  
1 思考  
2 写  
while 条件测试  
do  
 命令序列  
done  
while ：  
do  
 命令序列  
done  
################  
提示用户输入一个随机数,直到猜中为止  
使用系统自带的变量RANDOM提供系统随机数(1-100),使用while  
制作死循环  
使用取余算法使随机数变为1-100的随机数  
RANDOM为系统自带变量,值为0-16545  
显示你猜了几次  
  
#!/bin/bash  
num=$[RANDOM%100+1]  
i=0  
while :  
do  
 read -p "计算机生成了一个1-100的随机数,你猜：" cai  
 let i++  
 if [ $cai -eq $num ];then  
 echo "恭喜,猜对了"  
 echo "你猜了$i次"  
 exit  
 elif [ $cai -gt $num ];then  
 echo "Oops,猜大了"  
 else  
 echo "Oops,猜小了"  
 fi  
done  
#############################################

## 全局变量和局部变量的修改

全局变量：

局部变量：

num = 100  
nums = [123,345]

def test1():  
 global num  
 num += 100

def test2():  
 nums.append(100)

print(num) # 100

print(nums) # [123,345]

test1()

test2()

print(num) # 200

print(nums) # [123,345,100]

注意：

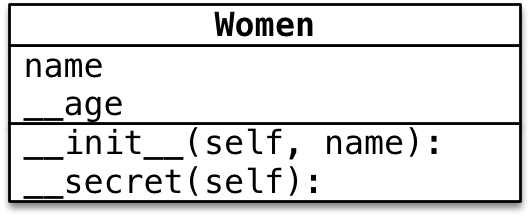
在一个函数中，对全局变量进行修改的时候，是否需要使用gloabl，要去关注是否对全局变量的地址进行修改。

1.如果修改了其内存地址，那么必须使用global。也就是即对于不可变对象的修改（数字，字符串，元祖）

2.如果只是修改了器指向空间中的数据，此时不必使用global.也就是说对于可变对象的修改(列表，字典，集合)

## 私有属性和私有方法

### 01. 应用场景及定义方式



#### 应用场景

在实际开发中，**对象** 的 **某些属性或方法** 可能只希望 **在对象的内部被使用**，而 **不希望在外部被访问到** **私有属性** 就是 **对象** 不希望公开的 **属性** **私有方法** 就是 **对象** 不希望公开的 **方法** 定义方式

在 定义属性或方法时，在 属性名或者方法名前 增加 两个下划线，定义的就是 私有 属性或方法

class Women:  
  
 def \_\_init\_\_(self, name):  
  
 self.name = name  
 # 不要问女生的年龄  
 self.\_\_age = 18  
  
 def \_\_secret(self):  
 print("我的年龄是 %d" % self.\_\_age)  
  
  
xiaofang = Women("小芳")  
# 私有属性，外部不能直接访问  
# print(xiaofang.\_\_age)  
  
# 私有方法，外部不能直接调用  
# xiaofang.\_\_secret()

### 02. 私有属性和私有方法（科普）

*提示*：在日常开发中，**不要使用这种方式，访问对象的 私有属性 或 私有方法**

Python 中，并没有 **真正意义** 的 **私有** 在给 **属性、方法** 命名时，实际是对 **名称** 做了一些特殊处理，使得外界无法访问到 **处理方式**：在 **名称** 前面加上 \_类名 => \_类名\_\_名称

# 私有属性，外部不能直接访问到  
print(xiaofang.\_Women\_\_age)  
  
# 私有方法，外部不能直接调用  
xiaofang.\_Women\_\_secret()

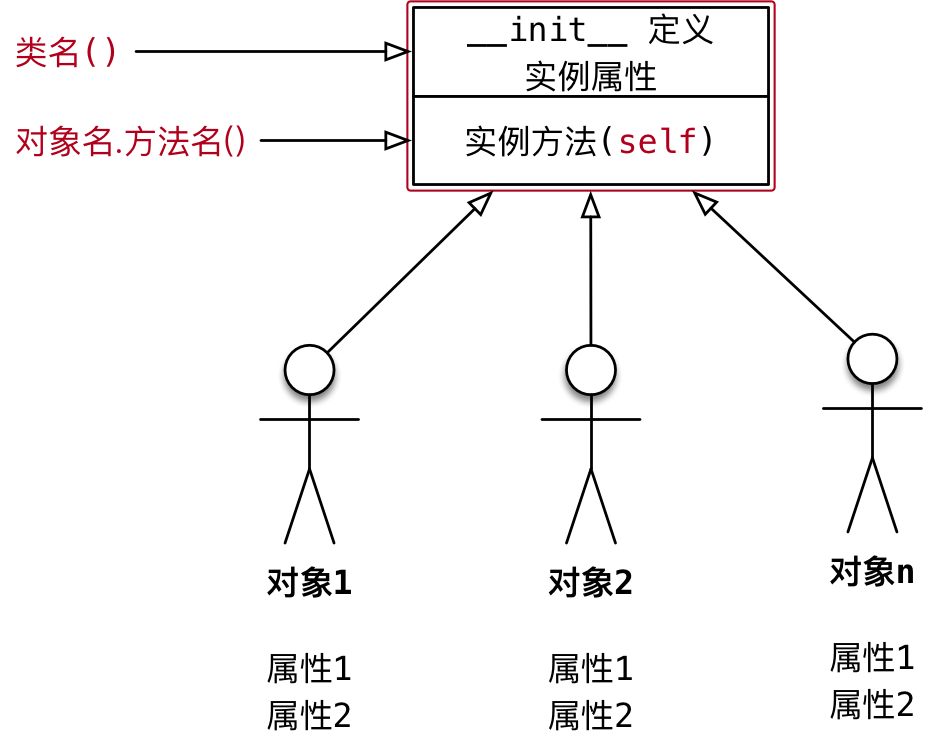
## 类属性和类方法

### 目标

* 类的结构
* 类属性和实例属性
* 类方法和静态方法

### 01. 类的结构

#### 1.1 术语 —— 实例

1. 使用面相对象开发，**第 1 步** 是设计 **类**
2. 使用 **类名()** 创建对象，**创建对象** 的动作有两步： 1) 在内存中为对象 **分配空间**
3. 2) 调用初始化方法\_\_init\_\_ 为 **对象初始化**
4. 对象创建后，**内存** 中就有了一个对象的 **实实在在** 的存在 —— **实例** 

* 因此，通常也会把：
  1. 创建出来的 **对象** 叫做 **类** 的 **实例**
  2. 创建对象的 **动作** 叫做 **实例化**
  3. **对象的属性** 叫做 **实例属性**
  4. **对象调用的方法** 叫做 **实例方法**
* 在程序执行时：
  1. 对象各自拥有自己的 **实例属性**
  2. 调用对象方法，可以通过self.
     + 访问自己的属性
     + 调用自己的方法
* **结论**
* **每一个对象** 都有自己 **独立的内存空间, 保存各自不同的属性** **多个对象的方法，在内存中只有一份，**在调用方法时，**需要把对象的引用** 传递到方法内部

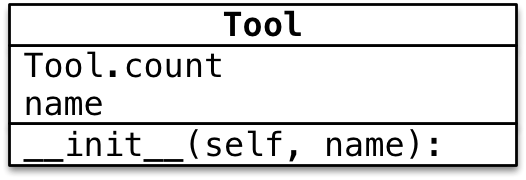
#### 1.2 类是一个特殊的对象

Python 中 一切皆对象：

* class AAA: 定义的类属于 **类对象**
* obj1 = AAA() 属于 **实例对象**
* 在程序运行时，**类** 同样 **会被加载到内存**
* 在 Python 中，**类** 是一个特殊的对象 —— **类对象**
* 在程序运行时，**类对象** 在内存中 **只有一份**，使用 **一个类** 可以创建出 **很多个对象实例**
* 除了封装 **实例** 的 **属性** 和 **方法外\***，**类对象** 还可以拥有自己的 **属性** 和 **方法** 
  1. **类属性**
  2. **类方法** 通过 **类名.** 的方式可以 **访问类的属性** 或者 **调用类的方法**

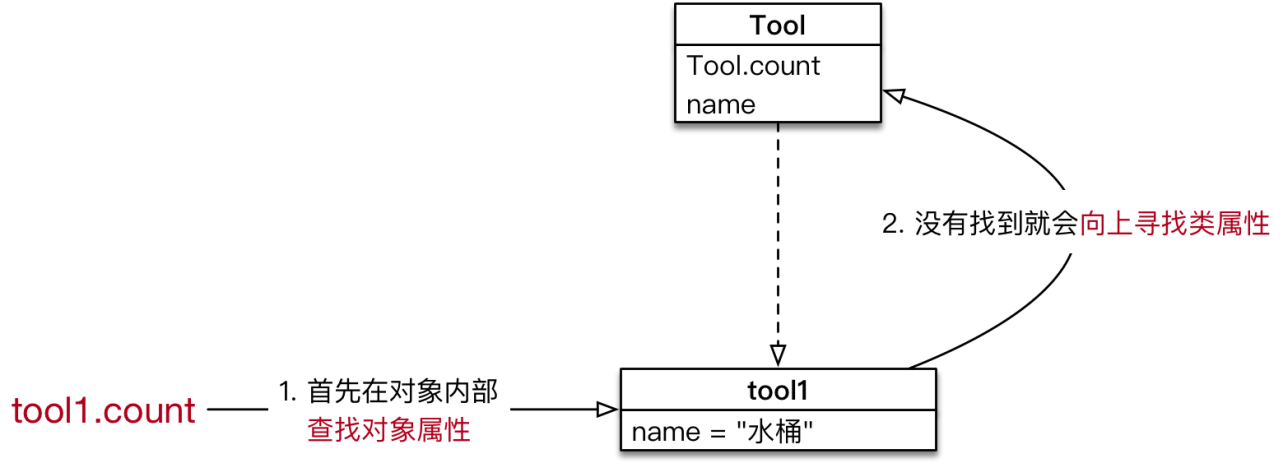
### 02. 类属性和实例属性

#### 2.1 概念和使用

* **类属性** 就是给 **类对象** 中定义的 **属性**
* 通常用来记录 **与这个类相关** 的特征
* 类属性 **不会用于记录** 具体对象的特征
* **示例需求**
  + 定义一个 **工具类**
  + 每件工具都有自己的 name
  + **需求** —— 知道使用这个类，创建了多少个工具对象？
  + 
  + class Tool(object):  
      
     # 使用赋值语句，定义类属性，记录创建工具对象的总数  
     count = 0  
      
     def \_\_init\_\_(self, name):  
     self.name = name  
      
     # 针对类属性做一个计数+1  
     Tool.count += 1  
      
      
    # 创建工具对象  
    tool1 = Tool("斧头")  
    tool2 = Tool("榔头")  
    tool3 = Tool("铁锹")  
      
    # 知道使用 Tool 类到底创建了多少个对象?  
    print("现在创建了 %d 个工具" % Tool.count)

#### 2.2 属性的获取机制

* 在 Python 中 **属性的获取** 存在一个 **向上查找机制**



* 因此，要访问类属性有两种方式： **类名.类属性** **对象.类属性 （不推荐）**

**注意**

* 如果使用 **对象.类属性** = **值** 赋值语句，只会 **给对象添加一个属性，**而不会影响到 **类属性的值**

### 03. 类方法和静态方法

#### 3.1 类方法

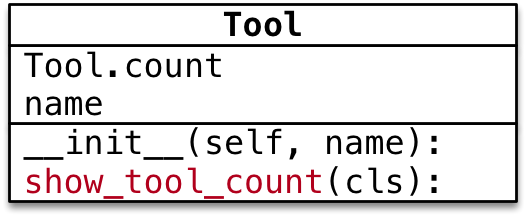
* **类属性** 就是针对 **类对象** 定义的属性
  + 使用 **赋值语句** 在 class 关键字下方可以定义 **类属性**
  + 在 **类方法** 内部可以直接访问 **类属性** 或者调用其他的 **类方法**
* **类方法** 就是针对 **类对象** 定义的方法
  + 在 **类方法** 内部可以直接访问 **类属性** 或者调用其他的 **类方法**

**语法如下**

@classmethod  
def 类方法名(cls):  
 pass

* 类方法需要用 **修饰器** @classmethod 来标识，告诉解释器这是一个类方法
* 类方法的 **第一个参数** 应该是 cls
  + 由 **哪一个类** 调用的方法，方法内的 cls 就是 哪一个类的引用
  + 这个参数和 **实例方法** 的第一个参数是 self 类似
  + 提示 **使用其他名称也可以，不过习惯使用 cls**
* 通过**类名. 调用** 类方法，调用方法时，不需要传递 cls 参数
* 在方法内部
  + 可以通过 cls. 访问类的属性
  + 也可以通过cls. 调用其他的类方法

**示例需求**

* 定义一个 **工具类**
* 每件工具都有自己的 name
* **需求** —— 在 **类** 封装一个 show\_tool\_count 的类方法，输出使用当前这个类，创建的对象个数
* ￼
* @classmethod  
  def show\_tool\_count(cls):  
   """显示工具对象的总数"""  
   print("工具对象的总数 %d" % cls.count)
* 在类方法内部，可以直接使用 cls 访问 **类属性** 或者 **调用类方法**

#### 3.2 静态方法

* 在开发时，如果需要在 **类** 中封装一个方法，这个方法：
  + 既 **不需要** 访问 **实例属性** 或者调用 **实例方法**
  + 也 **不需要** 访问 **类属性** 或者调用 **类方法**
* 这个时候，可以把这个方法封装成一个 **静态方法**

**语法如下**

@staticmethod  
def 静态方法名():  
 pass

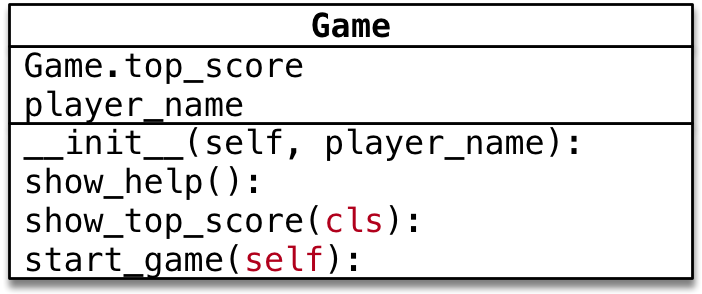
* **静态方法** 需要用 **修饰器** @staticmethod 来标识，**告诉解释器这是一个静态方法**
* 通过 **类名.** 调用 **静态方法**

class Dog(object):  
   
 # 狗对象计数  
 dog\_count = 0  
   
 @staticmethod  
 def run():  
   
 # 不需要访问实例属性也不需要访问类属性的方法  
 print("狗在跑...")  
  
 def \_\_init\_\_(self, name):  
 self.name = name

#### 3.3 方法综合案例

**需求**

1. 设计一个 Game 类
2. 属性：
   * 定义一个 **类属性** top\_score 记录游戏的 **历史最高分**
   * 定义一个 **实例属性**player\_name 记录 **当前游戏的玩家姓名**
3. 方法：
   * **静态方法** show\_help 显示游戏帮助信息
   * **类方法** show\_top\_score 显示历史最高分
   * **实例方法** start\_game 开始当前玩家的游戏
4. 主程序步骤
   * 1) 查看帮助信息
   * 2) 查看历史最高分
   * 3) 创建游戏对象，开始游戏

* 

**案例小结** **1. 实例方法** —— 方法内部需要访问 **实例属性**

* **实例方法** 内部可以使用 **类名.** 访问类属性

**2. 类方法** —— 方法内部 **只** 需要访问 **类属性** **3. 静态方法** —— 方法内部，不需要访问 **实例属性** 和 **类属性**

**提问** 如果方法内部 即需要访问 **实例属性，**又需要访问 **类属性**，应该定义成什么方法？

**答案**

* 应该定义 **实例方法**
* 因为，**类只有一个**，在 **实例方法** 内部可以使用 **类名.** 访问类属性

class Game(object):  
  
 # 游戏最高分，类属性  
 top\_score = 0  
  
 @staticmethod  
 def show\_help():  
 print("帮助信息：让僵尸走进房间")  
   
 @classmethod  
 def show\_top\_score(cls):  
 print("游戏最高分是 %d" % cls.top\_score)  
  
 def \_\_init\_\_(self, player\_name):  
 self.player\_name = player\_name  
  
 def start\_game(self):  
 print("[%s] 开始游戏..." % self.player\_name)  
   
 # 使用类名.修改历史最高分  
 Game.top\_score = 999  
  
# 1. 查看游戏帮助  
Game.show\_help()  
  
# 2. 查看游戏最高分  
Game.show\_top\_score()  
  
# 3. 创建游戏对象，开始游戏  
game = Game("小明")  
  
game.start\_game()  
  
# 4. 游戏结束，查看游戏最高分  
Game.show\_top\_score()

## 闭包的原理以及应用

### python的闭包

#### 定义：

如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包(closure)。

#### 示例：

def f1():  
 name1 = 'tom'  
 name2 = 'bob'  
 def f2():  
 print('hello,%s,%s' %(name1,name2))  
 return f2   
   
   
if \_\_name\_\_ == '\_\_main\_\_':  
 func = f1()  
 func()  
 print(func.\_\_closure\_\_)  
 print(func.\_\_closure\_\_[0].cell\_contents)  
 print(func.\_\_closure\_\_[1].cell\_contents)  
   
# 输出结果  
hello,tom,bob  
(<cell at 0x03A55CF0: str object at 0x03CBEA20>, <cell at 0x03CBEAD0: str object at 0x03CBEAA0>)  
tom  
bob  
  
#所有函数都有一个 \_\_closure\_\_属性，如果这个函数是一个闭包的话，那么它返回的是一个由 cell 对象 组成的元组对象。cell 对象的cell\_contents 属性就是闭包中的自由变量。

总结：

Python中，闭包就是一个包含了函数和引用环境变量的代码块，我倾向于把闭包看做一种代码结构，而不是函数，是因为第一次调用它的时候，只是把闭包的引用环境和内部的函数包装在一起，以一个函数对象来返回，而没有执行内部的函数，当调用这个函数对象时，才会执行内部的函数功能。

#### 示例2

函数传参闭包示例

def f1(x,y):  
 def f2(z):  
 return (x + y) % z  
 return f2  
  
func = f1(10,5)  
print(func(2)) # 1

#### 闭包的简单应用

import logging  
  
def log\_header(logger\_name):  
 logging.basicConfig(level=logging.DEBUG, format='%(asctime)s [%(name)s] %(levelname)s %(message)s', datefmt='%Y-%m-%d %H:%M:%S')  
 logger = logging.getLogger(logger\_name)   
 def \_logging(something,level):  
 if level == 'debug':  
 logger.debug(something)  
 elif level == 'warning':  
 logger.warning(something)  
 elif level == 'error':  
 logger.error(something)  
 else:  
 raise Exception("I dont know what you want to do?" )  
 return \_logging  
   
project\_1\_logging = log\_header('project\_1')  
   
project\_2\_logging = log\_header('project\_2')  
   
def project\_1():  
 #do something  
 project\_1\_logging('this is a debug info','debug')  
 #do something  
 project\_1\_logging('this is a warning info','warning')  
 # do something  
 project\_1\_logging('this is a error info','error')  
   
def project\_2():  
  
 # do something  
 project\_2\_logging('this is a debug info','debug')  
 # do something  
 project\_2\_logging('this is a warning info','warning')  
 # do something  
 project\_2\_logging('this is a critical info','error')  
   
project\_1()  
project\_2()  
  
  
###输出结果：  
2019-05-26 22:56:23 [project\_1] DEBUG  this is a debug info  
2019-05-26 22:56:23 [project\_1] WARNING  this is a warning info  
2019-05-26 22:56:23 [project\_1] ERROR  this is a error info  
2019-05-26 22:56:23 [project\_2] DEBUG  this is a debug info  
2019-05-26 22:56:23 [project\_2] WARNING  this is a warning info  
2019-05-26 22:56:23 [project\_2] ERROR  this is a critical info

## 迭代器和生成器

### Iterator **迭代器**

#### 什么是迭代器

能用next(it) 函数取值的对象（实例）

#### 说明：

用 iter 函数可返回一个可迭代对象的迭代器。

迭代器是访问可迭代对象的一种方式。

迭代器只能向前，不能后退。

##### 示例1

L = [1，1,2,3,5,8,13]   
it = iter(L) # 返回可迭代对象的迭代器   
next(it) #1   
next(it) #1   
next(it) #2   
next(it) #3   
......   
next(it) #13   
next(it) # StopIteration 异常

##### 示例2

it = iter(range(1,10,3print(next(it))  
print(next(it))  
print(next(it))  
try:  
print(next(it))  
except StopIteration:  
print('迭代器取值结束range函数：  
a = range(1,10,2)  
it = iter(a)  
try:  
while True:  
print(next(it)except StopIteration:  
print('打印结束')

**函数：**

iter(x)

* 从一个对象x找那个返回一个迭代器，x必须是能提供一个迭代器的对象

next(it)

* 从迭代器it中获取下一个记录，如果无法获取下一条记录则会触发StopIteration异常

**Generator** **生成器\***\*( Python2.5\*\* **及以后的版本中才有\***\*)\*\*

* 生成器能够提供一个迭代器对象（实例）

### 生成器函数

含有 yield 语句的函数是生成器函数 (yield 翻译为产生或者生成)

#### yield 语句：

**语法：**

* yield 表达式

**说明：**

* yield 用于def 函数中，目的是将次函数作为生成器函数使用
* yield用来生成数据，供迭代器next(it)函数使用

**示例：**

\#以下是生成器函数   
def myyield():   
 print('yield 2之前')   
 yield 2   
 print('yield 3之前')   
 yield 3   
 print('yield 5之前')   
 yield 5   
 print('yield 7之前')   
 yield 7   
 print('生成器函数调用结束')   
  
myyield() #返回myyield生成器对象   
it = iter(myyield())   
print(next(it)) # 2   
print(next(it)) # 3   
print(next(it)) # 5   
print(next(it)) # 7   
  
  
def myyield()：   
 yield 2   
 yield 3   
 return '我是生成器函数中的return语句'  
 # raise StopIteration('我是生成器函数中的return语句')  
 yield 5  
 yield 7

**说明**

生成器函数的调用将返回一个生成器对象，生成器对象是可迭代对象

在生成器函数内调用 return 会产生一个 StopIteration 异常

**实际应用：**

写一个生成器函数：

传入一个终止值n，可以生成1-n之间奇数的生成器。

用for语句打印这些奇数。

def mymodd(n):  
 x = 1  
 while x <= n:  
 yield x  
 x += 2  
for y in mymod(n):  
 print(y)

**练习**

[myxrange.py](深圳民治串讲\第一阶段串讲\myxrange.py) \# 1. 写一个生成器函数 myxrange(start, stop, step),

\# 来生成一系列整数,规则与range函数完全相同

\# (不允许调用range）

def myxrange(start, stop=None, step=None):

if stop is None:

stop = start

start = 0

if step is None:

step = 1

if step > 0: # 正向生成

i = start

while i < stop:

yield i

i += step

elif step < 0: # 反向生成

i = start

while i > stop:

yield i

i += step

# for x in myxrange(10, 0, -3):

# print(x)

s = sum( (x \*\* 2 for x in myxrange(1, 10) if x % 2 == 1) )

print(s)

## python 单例模式 工厂模式

设计模式来源于建筑行业,中西方建筑各有自己的风格;每种风格都有自己的建筑方法(套 路);只要按照不同风格的建筑方法（套路）就能建成不同风格的建筑;简单的说设计模式就是一种(方法)套路;再比如肯德基麦当劳快餐店为什么全球的口味都一样呢？就是做食品的套路相同(炸鸡块多长时间都是固定的)。

程序中设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。使用设计模式是为了重用代码、让代码更容易被他人理解、保证代码可靠性。

### 一、单例模式

保证只有一个对象。

注意：在python中导入模块就是一种 **单例模式。**

**练习1：创建Sun类**

class People(object):  
 instance = None  
   
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if not cls.instance:  
 cls.instance = super().\_\_new\_\_(cls)  
 return cls.instance  
 else:  
 return cls.instance  
   
per1 = People()  
per2 = People()  
# 没有重写new前，id不同  
print(id(per1)) # 35113504  
print(id(per2)) # 35113504

### 二、工厂模式

**工厂模式** 是一个在软件开发中用来 **创建对象** 的设计模式。

当程序运行输入一个“类型”的时候，需要 **创建于此相应的对象** 。这就用到了 **工厂模式**。在如此情形中，实现代码 **基于工厂模式，** 可以达到可扩展，可维护的代码。当增加一个新的类型，**不在需要修改已存在的类，只增加能够产生新类型的子类。**

使用工厂模式 **应用场景：**

不知道用户想要创建什么样的对象

**练习1：创建CarFactory工厂类。**

class Car(object):  
 def run(self):  
 print('so~so~de')  
   
 def stop(self):  
 print('ci~~~~')  
   
   
class BMW(Car):  
 pass  
   
class Benz(Car):  
 pass  
   
   
class CarFactory(object):  
 def make\_car(self,name):  
 if name == 'BMW':  
 return BMW()  
 if name == 'Benz':  
 return Benz()  
   
factory = CarFactory()  
car = factory.make\_car('BMW')  
print(type(car)) # <class '\_\_main\_\_.BMW'>