python语言有什么特点

下面是一些关键点：

1、Python是一种解释型语言。这就是说，与C语言和C的衍生语言不同，Python代码在运行之前不需要编译。其他解释型语言还包括PHP和Ruby。

2、Python是动态类型语言，指的是你在声明变量时，不需要说明变量的类型。你可以直接编写类似x=111和x="I'm a string"这样的代码，程序不会报错。

3、Python非常适合面向对象的编程（OOP），因为它支持通过组合（composition）与继承（inheritance）的方式定义类（class）。Python中没有访问说明符（access specifier，类似C++中的public和private），这么设计的依据是“大家都是成年人了”。

4、在Python语言中，函数是第一类对象（first-class objects）。这指的是它们可以被指定给变量，函数既能返回函数类型，也可以接受函数作为输入。类（class）也是第一类对象。

5、Python代码编写快，但是运行速度比编译语言通常要慢。好在Python允许加入基于C语言编写的扩展，因此我们能够优化代码，消除瓶颈，这点通常是可以实现的。numpy就是一个很好地例子，它的运行速度真的非常快，因为很多算术运算其实并不是通过Python实现的。

6、Python用途非常广泛——网络应用，自动化，科学建模，大数据应用，等等。它也常被用作“胶水语言”，帮助其他语言和组件改善运行状况。

7、Python让困难的事情变得容易，因此程序员可以专注于算法和数据结构的设计，而不用处理底层的细节。

列举几个常用的python内置模块

[Python17个常用内置模块总结 （运维）](https://blog.csdn.net/JackLiu16/article/details/79513345)

1、getpass模块详解

pwd = getpass.getpass("请输入密码：")  #输入密码不可见

yh = getpass.getuser()                #显示当前登录系统用户名;

2、os模块

os.getcwd()                           #获取当前工作目录，即当前python脚本工作的目录路径

os.chdir("dirname")                   #改变当前脚本工作目录；相当于shell下cd

os.curdir                             #返回当前目录: ('.')

os.pardir                             #获取当前目录的父目录字符串名：('..')

os.makedirs('dirname1/dirname2')      #可生成多层递归目录

os.removedirs('dirname1')             #若目录为空，则删除，并递归到上一级目录，如若也为空，则删除，依此类推

os.mkdir('dirname')                   #生成单级目录；相当于shell中mkdir dirname

os.rmdir('dirname')                   #删除单级空目录，若目录不为空则无法删除，报错；相当于shell中rmdir dirname

os.listdir('dirname')                 #列出指定目录下的所有文件和子目录，包括隐藏文件，并以列表方式打印

os.remove()                           #删除一个文件

os.rename("oldname","newname")        #重命名文件/目录

os.stat('path/filename')              #获取文件/目录信息

os.sep                                #输出操作系统特定的路径分隔符，win下为"\\",Linux下为"/"

os.linesep                            #输出当前平台使用的行终止符，win下为"\t\n",Linux下为"\n"

os.pathsep                            #输出用于分割文件路径的字符串

os.name                               #输出字符串指示当前使用平台。win->'nt'; Linux->'posix'

os.system("bash command")             #运行shell命令，直接显示

os.environ                            #获取系统环境变量

os.path.abspath(path)                 #返回path规范化的绝对路径

os.path.split(path)                   #将path分割成目录和文件名二元组返回

os.path.dirname(path)                 #返回path的目录。其实就是os.path.split(path)的第一个元素

os.path.basename(path)                #返回path最后的文件名。如何path以／或\结尾，那么就会返回空值。即os.path.split(path)的第二个元素

os.path.exists(path)                  #如果path存在，返回True；如果path不存在，返回False

os.path.isabs(path)                   #如果path是绝对路径，返回True

os.path.isfile(path)                  #如果path是一个存在的文件，返回True。否则返回False

os.path.isdir(path)                   #如果path是一个存在的目录，则返回True。否则返回False

os.path.join(path1[, path2[, ...]])   #将多个路径组合后返回，第一个绝对路径之前的参数将被忽略

os.path.getatime(path)                #返回path所指向的文件或者目录的最后存取时间

os.path.getmtime(path)                #返回path所指向的文件或者目录的最后修改时间

3、sys模块

sys.argv                              #命令行参数List，第一个元素是程序本身路径

sys.exit(n)                           #退出程序，正常退出时exit(0)

sys.version                           #获取Python解释程序的版本信息

sys.maxint                            #最大的Int值

sys.path                              #返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值

sys.platform                          #返回操作系统平台名称

sys.stdout.write('please:')

val = sys.stdin.readline()[:-1]

4、subprocess模块

执行系统命令

os.system

commands.\*      --废弃，3.x中被移除

result = commands.getoutput('cmd')

以上执行shell命令的相关的模块和函数的功能均在 subprocess 模块中实现，并提供了更丰富的功能。

call

执行命令，返回状态码

ret = subprocess.call(["ls", "-l"], shell=False)

ret = subprocess.call("ls -l", shell=True)

shell = True ，允许 shell 命令是字符串形式

check\_call

执行命令，如果执行状态码是 0 ，则返回0，否则抛异常

subprocess.check\_call(["ls", "-l"])

subprocess.check\_call("exit 1", shell=True)

check\_output（此下两条命令在2.6执行失败，要是2.7应该才可以）

执行命令，如果状态码是 0 ，则返回执行结果，否则抛异常

subprocess.check\_output(["echo", "Hello World!"])

subprocess.check\_output("exit 1", shell=True)

subprocess.Popen(...)

用于执行复杂的系统命令

参数：

args：                      #shell命令，可以是字符串或者序列类型（如：list，元组）

bufsize：                   #指定缓冲。0 无缓冲,1 行缓冲,其他 缓冲区大小,负值 系统缓冲

stdin, stdout, stderr：     #分别表示程序的标准输入、输出、错误句柄

preexec\_fn：                #只在Unix平台下有效，用于指定一个可执行对象（callable object），它将在子进程运行之前被调用

close\_sfs：                 #在windows平台下，如果close\_fds被设置为True，则新创建的子进程将不会继承父进程的输入、输出、错误管道。所以不能将close\_fds设置为True同时重定向子进程的标准输入、输出与错误(stdin, stdout, stderr)。

shell：                     #同上

cwd：                       #用于设置子进程的当前目录

env：                       #用于指定子进程的环境变量。如果env = None，子进程的环境变量将从父进程中继承。

universal\_newlines：        #不同系统的换行符不同，True -> 同意使用 \n

startupinfo与createionflags #只在windows下有效

将被传递给底层的CreateProcess()函数，用于设置子进程的一些属性，如：主窗口的外观，进程的优先级等等

import subprocess

ret1 = subprocess.Popen(["mkdir","t1"])

ret2 = subprocess.Popen("mkdir t2", shell=True)

5、hashlib模块

用于加密相关的操作，代替了md5模块和sha模块，主要提供 SHA1, SHA224, SHA256, SHA384, SHA512 ，MD5 算法

import hashlib

# ######## md5 ########

hash = hashlib.md5()

hash.update('admin')

print hash.hexdigest()

# ######## sha1 ########

hash = hashlib.sha1()

hash.update('admin')

print hash.hexdigest()

# ######## sha256 ########

hash = hashlib.sha256()

hash.update('admin')

print hash.hexdigest()

# ######## sha384 ########

hash = hashlib.sha384()

hash.update('admin')

print hash.hexdigest()

# ######## sha512 ########

hash = hashlib.sha512()

hash.update('admin')

print hash.hexdigest()

以上加密算法虽然依然非常厉害，但时候存在缺陷，即：通过撞库可以反解。所以，有必要对加密算法中添加自定义key再来做加密。

import hashlib

# ######## md5 ########

hash = hashlib.md5('898oaFs09f')

hash.update('admin')

print hash.hexdigest()

还不够吊？python 还有一个 hmac 模块，它内部对我们创建 key 和 内容 再进行处理然后再加密

import hmac

h = hmac.new('wueiqi')

h.update('hellowo')

print h.hexdigest()

6,7、json 和 pickle

用于序列化的两个模块

json，用于字符串 和 python数据类型间进行转换

pickle，用于python特有的类型 和 python的数据类型间进行转换

Json模块提供了四个功能：dumps、dump、loads、load

pickle模块提供了四个功能：dumps、dump、loads、load

pickle

>>> import pickle

>>> data = {'k1' : 123, 'k2' : 'hello'}

>>> p\_str = pickle.dumps(data)   #序列化

>>> print p\_str

>>> loadsed = pickle.loads(p\_str)   #反序列化

>>> print loadsed

序列化到文件

>>> li = ['wsyht',11,22,'ok','yes']

>>> pickle.dump(li,open('test.txt','w'))   #序列化到文件

>>> pickle.load(open('test.txt'))          #从文件反序列化出来

json

>>> import json

>>> data = {'k1':123,'k2':'abc'}

>>> str = json.dumps(data)

>>> stt= json.loads(str)

序列化到文件

>>> li = ['wsyht',11,22,'ok','yes']

>>> json.dump(li,open('test.txt','w'))  #序列化到文件

>>> json.load(open('test.txt'))         #从文件反序化出来

8、shutil模块

shutil.make\_archive(base\_name, format,...)

创建压缩包并返回文件路径，例如：zip、tar

base\_name： 压缩包的文件名，也可以是压缩包的路径。只是文件名时，则保存至当前目录，否则保存至指定路径，

如：www                        =>保存至当前路径

如：/Users/wupeiqi/www =>保存至/Users/wupeiqi/

format： 压缩包种类，“zip”, “tar”, “bztar”，“gztar”

root\_dir： 要压缩的文件夹路径（默认当前目录）

owner： 用户，默认当前用户

group： 组，默认当前组

logger： 用于记录日志，通常是logging.Logger对象

#将 /Users/wupeiqi/Downloads/test 下的文件打包放置当前程序目录

import shutil

ret = shutil.make\_archive("wwwwwwwwww", 'gztar', root\_dir='/Users/wupeiqi/Downloads/test')

#将 /mnt下的文件打包放置 /tmp目录

import shutil

ret = shutil.make\_archive("/tmp/www", 'gztar', root\_dir='/mnt')  #2.6用不了，2.7或许可以

类似于高级API，而且主要强大之处在于其对文件的复制与删除操作更是比较支持好。

相关API介绍

copyfile(src, dst)

从源src复制到dst中去。当然前提是目标地址是具备可写权限。抛出的异常信息为

IOException. 如果当前的dst已存在的话就会被覆盖掉。

 copyfile( src, dst)  从源src复制到dst中去。当然前提是目标地址是具备可写权限。抛出的异常信息为IOException. 如果当前的dst已存在的话就会被覆盖掉

 copymode( src, dst)  只是会复制其权限其他的东西是不会被复制的

 copystat( src, dst)  复制权限、最后访问时间、最后修改时间

 copy( src, dst)     复制一个文件到一个文件或一个目录

 copy2( src, dst)   在copy上的基础上再复制文件最后访问时间与修改时间也复制过来了，类似于cp –p的东西

 copy2( src, dst)   如果两个位置的文件系统是一样的话相当于是rename操作，只是改名；如果是不在相同的文件系统的话就是做move操作

 copytree(olddir,newdir,True/Flase)  把olddir拷贝一份newdir，如果第3个参数是True，则复制目录时将保持文件夹下的符号连接，如果第3个参数是False，则将在复制的目录下生成物理副本来替代符号连接

 shutil.rmtree("te")      删除一个目录

import shutil

shutil.copyfile('f:/temp.txt', 'f:/os.txt') #复制文件

shutil.copytree('f:/temp', 'f:/os')  #复制目录

# ######## zip的用法 ########

shutil 对压缩包的处理是调用 ZipFile 和 TarFile 两个模块来进行的，详细：

import zipfile

# 压缩

z = zipfile.ZipFile('laxi.zip', 'w')

z.write('a.log')        #压缩包写入a.log

z.write('data.data')    #写入data文件

z.close()

# 解压

z = zipfile.ZipFile('laxi.zip', 'r')

z.extractall()

z.close()

9、time模块

三种表示主式：

1、时间戳 1970年1月1日后的秒

2、元组包含了：年、日、星期等...time.struct\_time

3、格式化的字符串 2014-11-11 11:11 print time.time()

#时间戳形式存在

print time.time()

print time.mktime(time.localtime())  #print (time.localtime())此为元组形式，这一整句意思是把元组形式转化成时间戳形式

#元组形式存在

print time.gmtime() #可加时间戳参数

print time.localtime() #可加时间戳参数

print time.strptime('2014-11-11','%Y-%m-%d') #字符串形式转换成元组形式

#字符串形式存在

print time.strftime('%Y-%m-%d')  #默认当前时间，必须记住，工作中用得最多

print time.strftime('%Y-%m-%d',time.localtime())  #默认当前时间

print time.asctime()

print time.asctime(time.localtime())

print time.ctime(time.time())

时间的三种表示方式演示

>>> import time

>>> print time.time()

1469014348.5   #秒，时间戳的方式

>>> print time.gmtime()

time.struct\_time(tm\_year=2016, tm\_mon=7, tm\_mday=20, tm\_hour=11, tm\_min=25, tm\_sec=53, tm\_wday=2, tm\_yday=202, tm\_isdst=0)

>>> print time.strftime('%Y-%m-%d %H:%M:%S')

2016-07-20 19:36:16

10、datetime模块

import datetime

'''

datetime.date：表示日期的类。常用的属性有year, month, day

datetime.time：表示时间的类。常用的属性有hour, minute, second, microsecond

datetime.datetime：表示日期时间

datetime.timedelta：表示时间间隔，即两个时间点之间的长度

timedelta([days[, seconds[, microseconds[, milliseconds[, minutes[, hours[, weeks]]]]]]])

strftime("%Y-%m-%d")

'''

import datetime

print datetime.datetime.now()

print datetime.datetime.now() - datetime.timedelta(days=5)

11、re模块

compile

match search findall

group groups

正则表达式常用格式：

　　字符：\d \w \t  .

　　次数：\* + ? {m} {m,n}

示例：

#!#/usr/bin/env python

#coding:utf-8

import re

result1 = re.match('\d+', '1afsfweasfcxvsfd123')  #在你给的字符串起始位置去匹配,\d从数字开始找，+表示一个到多个

if result1:   #当result1等于True的时候，就是匹配，如果匹配就输出里面的内容

    print result1.group()  #用group方法把他匹配的内容输出出来

else:

    print 'nothing'

result2 = re.search('\d+', 'alsfj3af') #在整个内容里面去匹配，\d从数字开始找，+表示一个到多个

if result2:

    print result2.group()    #用group方法把他匹配的内容输出出来

result3 = re.findall('\d+', 'asfaf11sf22lj33') #只要匹配全都拿出来

print result3

com = re.compile('\d+')

print com.findall('asfaf11sf22lj33')

result5 = re.search('(\d+)\w\*(\d+)','aasflsjfa12aaljsf22lj13bb')

print result5.group()   #所有匹配内容输出

print result5.groups()  #只把括号\d，也就是组里面的内容输出

result6 = re.search('a{3,5}','aaaaaa') #匹配3到5次的aaaaa输出出来

print result6.group()

Python中的pass是什么？

Pass是一个在Python中不会被执行的语句。在复杂语句中，如果一个地方需要暂时被留白，它常常被用于占位符。

Python都有哪些自带的数据结构？

Python自带的数据结构分为可变的和不可变的。可变的有：数组、集合、字典；不可变的有：字符串、元组、数。

数组和元组之间的区别：数组内容是可以被修改的，而元组内容是只读的。另外，元组可以被哈希，比如作为字典的关键字。

Python是如何进行类型转换的

函数                      描述  
int(x [,base ])         将x转换为一个整数  
long(x [,base ])        将x转换为一个长整数  
float(x )               将x转换到一个浮点数  
complex(real [,imag ])  创建一个复数  
str(x )                 将对象 x 转换为字符串  
repr(x )                将对象 x 转换为表达式字符串  
eval(str )              用来计算在字符串中的有效Python表达式,并返回一个对象  
tuple(s )               将序列 s 转换为一个元组  
list(s )                将序列 s 转换为一个列表  
chr(x )                 将一个整数转换为一个字符  
unichr(x )              将一个整数转换为Unicode字符  
ord(x )                 将一个字符转换为它的整数值  
hex(x )                 将一个整数转换为一个十六进制字符串  
oct(x )                 将一个整数转换为一个八进制字符串

解释参数 \*args，\*\*kwargs

如果我们不确定要往函数中传入多少个参数，或者我们想往函数中以列表和元组的形式传参数时，那就使要用\*args；

如果我们不知道要往函数中传入多少个关键词参数，或者想传入字典的值作为关键词参数时，那就要使用\*\*kwargs。

args和kwargs这两个标识符是约定俗成的用法，你当然还可以用\*bob和\*\*billy，但是这样就并不太妥。

下面是具体的示例：

1. def f(\*args,\*\*kwargs): print args, kwargs
3. l = [1,2,3]
4. t = (4,5,6)
5. d = {'a':7,'b':8,'c':9}
7. f()
8. f(1,2,3) # (1, 2, 3) {}
9. f(1,2,3,"groovy") # (1, 2, 3, 'groovy') {}
10. f(a=1,b=2,c=3) # () {'a': 1, 'c': 3, 'b': 2}
11. f(a=1,b=2,c=3,zzz="hi") # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'}
12. f(1,2,3,a=1,b=2,c=3) # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2}
14. f(\*l,\*\*d) # (1, 2, 3) {'a': 7, 'c': 9, 'b': 8}
15. f(\*t,\*\*d) # (4, 5, 6) {'a': 7, 'c': 9, 'b': 8}
16. f(1,2,\*t) # (1, 2, 4, 5, 6) {}
17. f(q="winning",\*\*d) # () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}
18. f(1,2,\*t,q="winning",\*\*d) # (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}
20. def f2(arg1,arg2,\*args)

什么是lambda函数？它有什么好处?

lambda 函数是一个可以接收任意多个参数(包括可选参数)并且返回单个表达式值的函数。 lambda 函数不能包含命令，它们所包含的表达式不能超过一个。不要试图向lambda 函数中塞入太多的东西；如果你需要更复杂的东西，应该定义一个普通函数，然后想让它多长就多长。

match()和search()函数有什么区别

match()函数只检测字符串开头位置是否匹配，匹配成功才会返回结果，否则返回None

1. import re
2. print(re.match("func", "function"))
3. # 打印结果 <\_sre.SRE\_Match object; span=(0, 4), match='func'>
5. print(re.match("func", "function").span())
6. # 打印结果 (0, 4)
8. print(re.match("func1", "function"))
9. # 打印结果 None
11. 注意：print(re.match("func1", "function").span())会报错，因为取不到span

search()函数会在整个字符串内查找模式匹配,只到找到第一个匹配然后返回一个包含匹配信息的对象,该对象可以通过调用group()方法得到匹配的字符串,如果字符串没有匹配，则返回None。

1. import re
2. print(re.search("tion", "function"))
3. # 打印结果 <\_sre.SRE\_Match object; span=(4, 8), match='tion'>
5. print(re.search("tion", "function").span())
6. # 打印结果 (4, 8)
8. print(re.search("tion1", "function"))
9. # 打印结果 None
11. 注意：print(re.search("tion1", "function").span())会报错，因为取不到tion1

简述read、readline、readlines的区别

read读取整个文件

readline读取下一行数据

readlines读取整个文件到一个迭代器以供我们遍历（读取 到一个list中，以供使用，比较方便）

time,datetime,calender模块有什么区别

* time：提供了更底层的时间相关的功能，大部分函数调用了C语言的同名函数，但在不同的操作系统上表现的行为可能不一致，如：time.clock()。
* datetime：是python处理日期和时间的标准库，提供了更多实用功能，如：时间加减。
* calender：提供了日历相关的功能，如：判断是否闰年。

阅读下面的代码，写出A0，A1至An的最终值。

1. A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
2. A1 = range(10)
3. A2 = [i for i in A1 if i in A0]
4. A3 = [A0[s] for s in A0]
5. A4 = [i for i in A1 if i in A3]
6. A5 = {i:i\*i for i in A1}
7. A6 = [[i,i\*i] for i in A1]
8. A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}
9. A1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
10. A2 = []
11. A3 = [1, 3, 2, 5, 4]
12. A4 = [1, 2, 3, 4, 5]
13. A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
14. A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

请写出一段Python代码实现删除一个list里面的重复元素

1. >>> l = [1,1,2,3,4,5,4]
2. >>> list(set(l))
3. [1, 2, 3, 4, 5]
4. 或者
5. d = {}
6. for x in mylist:
7. d[x] = 1
8. mylist = list(d.keys())

Python里面如何实现tuple和list的转换？

1. >>> l = tuple(iplist)
2. >>> print l
3. ('217.169.209.2:6666', '192.227.139.106:7808', '110.4.12.170:83', '69.197.132.80:7808', '205.164.41.101:3128', '63.141.249.37:8089', '27.34.142.47:9090')
4. >>> t = list(l)
5. >>> print t
6. ['217.169.209.2:6666', '192.227.139.106:7808', '110.4.12.170:83', '69.197.132.80:7808', '205.164.41.101:3128', '63.141.249.37:8089', '27.34.142.47:9090']

举例说明创建字典的至少两种方法

1. # 1
2. dict1 = {key1:v1,key2:v2}
3. # 2
4. dict2 = {}
5. dict2[key1] = v1
6. dict2[key2] = v2
7. # 3
8. dict3 = dict(key1=v1,key2=v2)

Python生成随机数函数有哪些？

1. import random
3. random.random() 随机生成0-1之间的数字
5. random.uniform(1,10)随机生成1-10之间的数字（包括小数整数）
7. random.randint(1,10)生成1-10之间的整数
9. random.randrange(1,10,2) 生成1-10之间 步长为2的数字 所以只能是奇数。1,3,5,7,9
11. list=[1,2,3,4,5,6,7]
13. random.choice(list) 从list中随机抽取
15. random.sample(list,2) 从list中随机抽取2个元素的片段。

迭代器和生成器的区别

1）迭代器是一个更抽象的概念，任何对象，如果它的类有next方法和iter方法返回自己本身。对于string、list、dict、tuple等这类容器对象，使用for循环遍历是很方便的。在后台for语句对容器对象调用iter()函数，iter()是python的内置函数。iter()会返回一个定义了next()方法的迭代器对象，它在容器中逐个访问容器内元素，next()也是python的内置函数。在没有后续元素时，next()会抛出一个StopIteration异常

2）生成器（Generator）是创建迭代器的简单而强大的工具。它们写起来就像是正规的函数，只是在需要返回数据的时候使用yield语句。每次next()被调用时，生成器会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）

区别：生成器能做到迭代器能做的所有事,而且因为自动创建了\_\_iter\_\_()和next()方法,生成器显得特别简洁,而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法,当发生器终结时,还会自动抛出StopIteration异常

python多线程与多进程的区别

* 在UNIX平台上，当某个进程终结之后，该进程需要被其父进程调用wait，否则进程成为僵尸进程(Zombie)。所以，有必要对每个Process对象调用join()方法 (实际上等同于wait)。对于多线程来说，由于只有一个进程，所以不存在此必要性。
* 多进程应该避免共享资源。在多线程中，我们可以比较容易地共享资源，比如使用全局变量或者传递参数。在多进程情况下，由于每个进程有自己独立的内存空间，以上方法并不合适。此时我们可以通过共享内存和Manager的方法来共享资源。但这样做提高了程序的复杂度，并因为同步的需要而降低了程序的效率。

python下多线程的限制以及多进程中传递参数的方式

python多线程有个全局解释器锁（global interpreter lock），这个锁的意思是任一时间只能有一个线程使用解释器，跟单cpu跑多个程序一个意思，大家都是轮着用的，这叫“并发”，不是“并行”。  
多进程间共享数据，可以使用 multiprocessing.Value 和 multiprocessing.Array

Python是如何进行内存管理的？

一、垃圾回收：python不像C++，Java等语言一样，他们可以不用事先声明变量类型而直接对变量进行赋值。对Python语言来讲，对象的类型和内存都是在运行时确定的。这也是为什么我们称Python语言为动态类型的原因（这里我们把动态类型可以简单的归结为对变量内存地址的分配是在运行时自动判断变量类型并对变量进行赋值）。

二、引用计数：Python采用了类似Windows内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数。当变量被绑定在一个对象上的时候，该变量的引用计数就是1，(还有另外一些情况也会导致变量引用计数的增加),系统会自动维护这些标签，并定时扫描，当某标签的引用计数变为0的时候，该对就会被回收。

三、内存池机制Python的内存机制以金字塔行，-1，-2层主要有操作系统进行操作，

　　    第0层是C中的malloc，free等内存分配和释放函数进行操作；

　　    第1层和第2层是内存池，有Python的接口函数PyMem\_Malloc函数实现，当对象小于256K时有该层直接分配内存；

　　    第3层是最上层，也就是我们对Python对象的直接操作；

在 C 中如果频繁的调用 malloc 与 free 时,是会产生性能问题的.再加上频繁的分配与释放小块的内存会产生内存碎片. Python 在这里主要干的工作有:

如果请求分配的内存在1~256字节之间就使用自己的内存管理系统,否则直接使用 malloc.

这里还是会调用 malloc 分配内存,但每次会分配一块大小为256k的大块内存.

经由内存池登记的内存到最后还是会回收到内存池,并不会调用 C 的 free 释放掉.以便下次使用.对于简单的Python对象，例如数值、字符串，元组（tuple不允许被更改)采用的是复制的方式(深拷贝?)，也就是说当将另一个变量B赋值给变量A时，虽然A和B的内存空间仍然相同，但当A的值发生变化时，会重新给A分配空间，A和B的地址变得不再相同

简要描述Python的垃圾回收机制（garbage collection）

这里能说的很多。你应该提到下面几个主要的点：

1、Python在内存中存储了每个对象的引用计数（reference count）。如果计数值变成0，那么相应的对象就会小时，分配给该对象的内存就会释放出来用作他用。

2、偶尔也会出现引用循环（reference cycle）。垃圾回收器会定时寻找这个循环，并将其回收。举个例子，假设有两个对象o1和o2，而且符合o1.x == o2和o2.x == o1这两个条件。如果o1和o2没有其他代码引用，那么它们就不应该继续存在。但它们的引用计数都是1。

3、Python中使用了某些启发式算法（heuristics）来加速垃圾回收。例如，越晚创建的对象更有可能被回收。对象被创建之后，垃圾回收器会分配它们所属的代（generation）。每个对象都会被分配一个代，而被分配更年轻代的对象是优先被处理的。

python正则表达式中re.match与re.search有什么区别

re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

例如：  
print(re.match('hello', 'hello world').span())   ==>(0, 5)  
而print(re.match('world', 'hello world'))   ==>None  
  
print(re.search('hello', 'hello world').span())   ==>(0, 5)  
而print(re.search('world', 'hello world').span())   ==>(6,11)

python中类的变量：\_\_foo\_\_, \_\_foo, \_foo 有什么区别

\_\_foo\_\_: 定义的是特殊方法，一般是系统定义名字 ，类似 \_\_init\_\_() 之类的。

\_foo: 以单下划线开头的表示的是 protected 类型的变量，即保护类型只能允许其本身与子类进行访问，不能用于 from module import \*

\_\_foo: 双下划线的表示的是私有类型(private)的变量, 只能是允许这个类本身进行访问了。

你用过哪些python机器学习相关的框架？

[常用python机器学习库总结](https://www.cnblogs.com/Fighting365/p/6132707.html)

**1. Python网页爬虫工具集**

一个真实的项目，一定是从获取数据开始的。无论文本处理，机器学习和数据挖掘，都需要数据，除了通过一些渠道购买或者下载的专业数据外，常常需要大家自己动手爬数据，这个时候，爬虫就显得格外重要了，幸好，Python提供了一批很不错的网页爬虫工具框架，既能爬取数据，也能获取和清洗数据，也就从这里开始了：

**1.1 Scrapy**

Scrapy, a fast high-level screen scraping and web crawling framework for Python.

鼎鼎大名的Scrapy，相信不少同学都有耳闻，课程图谱中的很多课程都是依靠Scrapy抓去的，这方面的介绍文章有很多，推荐大牛pluskid早年的一篇文章：《Scrapy 轻松定制网络爬虫》，历久弥新。

官方主页：http://scrapy.org/  
Github代码页: https://github.com/scrapy/scrapy

**1.2 Beautiful Soup**

You didn’t write that awful page. You’re just trying to get some data out of it. Beautiful Soup is here to help. Since 2004, it’s been saving programmers hours or days of work on quick-turnaround screen scraping projects.

客观的说，Beautifu Soup不完全是一套爬虫工具，需要配合urllib使用，而是一套HTML / XML数据分析，清洗和获取工具。

官方主页：http://www.crummy.com/software/BeautifulSoup/

**1.3 Python-Goose**

Html Content / Article Extractor, web scrapping lib in Python

Goose最早是用Java写得，后来用Scala重写，是一个Scala项目。Python-Goose用Python重写，依赖了Beautiful Soup。前段时间用过，感觉很不错，给定一个文章的URL, 获取文章的标题和内容很方便。

Github主页：https://github.com/grangier/python-goose

**2. Python文本处理工具集**

从网页上获取文本数据之后，依据任务的不同，就需要进行基本的文本处理了，譬如对于英文来说，需要基本的tokenize，对于中文，则需要常见的中文分词，进一步的话，无论英文中文，还可以词性标注，句法分析，关键词提取，文本分类，情感分析等等。这个方面，特别是面向英文领域，有很多优秀的工具包，我们一一道来。

**2.1 NLTK — Natural Language Toolkit**

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and an active discussion forum.

搞自然语言处理的同学应该没有人不知道NLTK吧，这里也就不多说了。不过推荐两本书籍给刚刚接触NLTK或者需要详细了解NLTK的同学: 一个是官方的《Natural Language Processing with Python》，以介绍NLTK里的功能用法为主，同时附带一些Python知识，同时国内陈涛同学友情翻译了一个中文版，这里可以看到：推荐《用Python进行自然语言处理》中文翻译-NLTK配套书；另外一本是《Python Text Processing with NLTK 2.0 Cookbook》，这本书要深入一些，会涉及到NLTK的代码结构，同时会介绍如何定制自己的语料和模型等，相当不错。

官方主页：http://www.nltk.org/  
Github代码页：https://github.com/nltk/nltk

**2.2 Pattern**

Pattern is a web mining module for the Python programming language.

It has tools for data mining (Google, Twitter and Wikipedia API, a web crawler, a HTML DOM parser), natural language processing (part-of-speech taggers, n-gram search, sentiment analysis, WordNet), machine learning (vector space model, clustering, SVM), network analysis and canvas visualization.

Pattern由比利时安特卫普大学CLiPS实验室出品，客观的说，Pattern不仅仅是一套文本处理工具，它更是一套web数据挖掘工具，囊括了数据抓取模块（包括Google, Twitter, 维基百科的API，以及爬虫和HTML分析器），文本处理模块（词性标注，情感分析等），机器学习模块(VSM, 聚类，SVM）以及可视化模块等，可以说，Pattern的这一整套逻辑也是这篇文章的组织逻辑，不过这里我们暂且把Pattern放到文本处理部分。我个人主要使用的是它的英文处理模块Pattern.en, 有很多很不错的文本处理功能，包括基础的tokenize, 词性标注，句子切分，语法检查，拼写纠错，情感分析，句法分析等，相当不错。

官方主页：http://www.clips.ua.ac.be/pattern

**2.3 TextBlob: Simplified Text Processing**

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

TextBlob是一个很有意思的Python文本处理工具包，它其实是基于上面两个Python工具包NLKT和Pattern做了封装（TextBlob stands on the giant shoulders of NLTK and pattern, and plays nicely with both），同时提供了很多文本处理功能的接口，包括词性标注，名词短语提取，情感分析，文本分类，拼写检查等，甚至包括翻译和语言检测，不过这个是基于Google的API的，有调用次数限制。TextBlob相对比较年轻，有兴趣的同学可以关注。

官方主页：http://textblob.readthedocs.org/en/dev/

Github代码页：https://github.com/sloria/textblob

**2.4 MBSP for Python**

MBSP is a text analysis system based on the TiMBL and MBT memory based learning applications developed at CLiPS and ILK. It provides tools for Tokenization and Sentence Splitting, Part of Speech Tagging, Chunking, Lemmatization, Relation Finding and Prepositional Phrase Attachment.

MBSP与Pattern同源，同出自比利时安特卫普大学CLiPS实验室，提供了Word Tokenization, 句子切分，词性标注，Chunking, Lemmatization，句法分析等基本的文本处理功能，感兴趣的同学可以关注。

官方主页：http://www.clips.ua.ac.be/pages/MBSP

**2.5 Gensim: Topic modeling for humans**

Gensim是一个相当专业的主题模型Python工具包，无论是代码还是文档，我们曾经用《如何计算两个文档的相似度》介绍过Gensim的安装和使用过程，这里就不多说了。

官方主页：http://radimrehurek.com/gensim/index.html

github代码页：https://github.com/piskvorky/gensim

**2.6 langid.py: Stand-alone language identification system**

语言检测是一个很有意思的话题，不过相对比较成熟，这方面的解决方案很多，也有很多不错的开源工具包，不过对于Python来说，我使用过langid这个工具包，也非常愿意推荐它。langid目前支持97种语言的检测，提供了很多易用的功能，包括可以启动一个建议的server，通过json调用其API，可定制训练自己的语言检测模型等，可以说是“麻雀虽小，五脏俱全”。

Github主页：https://github.com/saffsd/langid.py

**2.7 Jieba: 结巴中文分词**

“结巴”中文分词：做最好的Python中文分词组件 “Jieba” (Chinese for “to stutter”) Chinese text segmentation: built to be the best Python Chinese word segmentation module.

好了，终于可以说一个国内的Python文本处理工具包了：结巴分词，其功能包括支持三种分词模式（精确模式、全模式、搜索引擎模式），支持繁体分词，支持自定义词典等，是目前一个非常不错的Python中文分词解决方案。

Github主页：https://github.com/fxsjy/jieba

**3. Python科学计算工具包**

说起科学计算，大家首先想起的是Matlab，集数值计算，可视化工具及交互于一身，不过可惜是一个商业产品。开源方面除了GNU Octave在尝试做一个类似Matlab的工具包外，Python的这几个工具包集合到一起也可以替代Matlab的相应功能：NumPy+SciPy+Matplotlib+iPython。同时，这几个工具包，特别是NumPy和SciPy，也是很多Python文本处理 & 机器学习 & 数据挖掘工具包的基础，非常重要。最后再推荐一个系列《用Python做科学计算》，将会涉及到NumPy, SciPy, Matplotlib，可以做参考。

**3.1 NumPy**

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

1）a powerful N-dimensional array object

2）sophisticated (broadcasting) functions

3）tools for integrating C/C++ and Fortran code

4） useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy几乎是一个无法回避的科学计算工具包，最常用的也许是它的N维数组对象，其他还包括一些成熟的函数库，用于整合C/C++和Fortran代码的工具包，线性代数、傅里叶变换和随机数生成函数等。NumPy提供了两种基本的对象：ndarray（N-dimensional array object）和 ufunc（universal function object）。ndarray是存储单一数据类型的多维数组，而ufunc则是能够对数组进行处理的函数。

官方主页：http://www.numpy.org/

**3.2 SciPy：Scientific Computing Tools for Python**

SciPy refers to several related but distinct entities:

1）The SciPy Stack, a collection of open source software for scientific computing in Python, and particularly a specified set of core packages.

2）The community of people who use and develop this stack.

3）Several conferences dedicated to scientific computing in Python – SciPy, EuroSciPy and SciPy.in.

4）The SciPy library, one component of the SciPy stack, providing many numerical routines.

“SciPy是一个开源的Python算法库和数学工具包，SciPy包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。其功能与软件MATLAB、Scilab和GNU Octave类似。 Numpy和Scipy常常结合着使用，Python大多数机器学习库都依赖于这两个模块。”—-引用自“Python机器学习库”

官方主页：http://www.scipy.org/

**3.3 Matplotlib**

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®\* or Mathematica®†), web application servers, and six graphical user interface toolkits.

matplotlib 是python最著名的绘图库，它提供了一整套和matlab相似的命令API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入GUI应用程序中。Matplotlib可以配合ipython shell使用，提供不亚于Matlab的绘图体验，总之用过了都说好。

官方主页：http://matplotlib.org/

**4. Python 机器学习 & 数据挖掘 工具包**

机器学习和数据挖掘这两个概念不太好区分，这里就放到一起了。这方面的开源Python工具包有很多，这里先从熟悉的讲起，再补充其他来源的资料，也欢迎大家补充。

**4.1 scikit-learn: Machine Learning in Python**

scikit-learn (formerly scikits.learn) is an open source machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, logistic regression, naive Bayes, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

首先推荐大名鼎鼎的scikit-learn，scikit-learn是一个基于NumPy, SciPy, Matplotlib的开源机器学习工具包，主要涵盖分类，回归和聚类算法，例如SVM， 逻辑回归，朴素贝叶斯，随机森林，k-means等算法，代码和文档都非常不错，在许多Python项目中都有应用。例如在我们熟悉的NLTK中，分类器方面就有专门针对scikit-learn的接口，可以调用scikit-learn的分类算法以及训练数据来训练分类器模型。

官方主页：http://scikit-learn.org/

**4.2 Pandas: Python Data Analysis Library**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Pandas也是基于NumPy和Matplotlib开发的，主要用于数据分析和数据可视化，它的数据结构DataFrame和R语言里的data.frame很像，特别是对于时间序列数据有自己的一套分析机制，非常不错。这里推荐一本书《Python for Data Analysis》，作者是Pandas的主力开发，依次介绍了iPython, NumPy, Pandas里的相关功能，数据可视化，数据清洗和加工，时间数据处理等，案例包括金融股票数据挖掘等，相当不错。

官方主页：http://pandas.pydata.org/

**4.3 mlpy – Machine Learning Python**

mlpy is a Python module for Machine Learning built on top of NumPy/SciPy and the GNU Scientific Libraries.mlpy provides a wide range of state-of-the-art machine learning methods for supervised and unsupervised problems and it is aimed at finding a reasonable compromise among modularity, maintainability, reproducibility, usability and efficiency. mlpy is multiplatform, it works with Python 2 and 3 and it is Open Source, distributed under the GNU General Public License version 3.

官方主页：http://mlpy.sourceforge.net/

**4.4 PyBrain**

PyBrain is a modular Machine Learning Library for Python. Its goal is to offer flexible, easy-to-use yet still powerful algorithms for Machine Learning Tasks and a variety of predefined environments to test and compare your algorithms.

PyBrain is short for Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library. In fact, we came up with the name first and later reverse-engineered this quite descriptive “Backronym”.

“PyBrain(Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network)是Python的一个机器学习模块，它的目标是为机器学习任务提供灵活、易应、强大的机器学习算法。（这名字很霸气）

PyBrain正如其名，包括神经网络、强化学习(及二者结合)、无监督学习、进化算法。因为目前的许多问题需要处理连续态和行为空间，必须使用函数逼近(如神经网络)以应对高维数据。PyBrain以神经网络为核心，所有的训练方法都以神经网络为一个实例。”

**4.5 Theano**

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

1）tight integration with NumPy – Use numpy.ndarray in Theano-compiled functions.

2）transparent use of a GPU – Perform data-intensive calculations up to 140x faster than with CPU.(float32 only)

3）efficient symbolic differentiation – Theano does your derivatives for function with one or many inputs.

4）speed and stability optimizations – Get the right answer for log(1+x) even when x is really tiny.

5）dynamic C code generation – Evaluate expressions faster.

6） extensive unit-testing and self-verification – Detect and diagnose many types of mistake.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. But it is also approachable enough to be used in the classroom (IFT6266 at the University of Montreal).

“Theano 是一个 Python 库，用来定义、优化和模拟数学表达式计算，用于高效的解决多维数组的计算问题。Theano的特点：紧密集成Numpy；高效的数据密集型GPU计算；高效的符号微分运算；高速和稳定的优化；动态生成c代码；广泛的单元测试和自我验证。自2007年以来，Theano已被广泛应用于科学运算。theano使得构建深度学习模型更加容易，可以快速实现多种模型。PS：Theano，一位希腊美女，Croton最有权势的Milo的女儿，后来成为了毕达哥拉斯的老婆。”

**4.6  Pylearn2**

Pylearn2 is a machine learning library. Most of its functionality is built on top of Theano. This means you can write Pylearn2 plugins (new models, algorithms, etc) using mathematical expressions, and theano will optimize and stabilize those expressions for you, and compile them to a backend of your choice (CPU or GPU).

“Pylearn2建立在theano上，部分依赖scikit-learn上，目前Pylearn2正处于开发中，将可以处理向量、图像、视频等数据，提供MLP、RBM、SDA等深度学习模型。”

你用过哪些python web开发相关的框架？

[Python web 目前较火的三大框架](https://blog.csdn.net/chenming_python/article/details/79126243)

1、Django

 Django是一个开放源代码的Web应用框架，由Python写成。采用了MTV的框架模式，即模型M，模板T和视图V。它最初是被开发来用于管理劳伦斯出版集团旗下的一些以新闻内容为主的网站的，即是CMS（内容管理系统）软件。

2、Flask

 Flask是一个使用 Python 编写的轻量级 Web 应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2 。Flask使用 BSD 授权。

 Flask也被称为 “microframework” ，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。

 Flask 很轻，花很少的成本就能够开发一个简单的网站。非常适合初学者学习。Flask 框架学会以后，可以考虑学习插件的使用。例如使用 WTForm + Flask-WTForm 来验证表单数据，用 SQLAlchemy + Flask-SQLAlchemy 来对你的数据库进行控制。

3、Tornado

 Tornado是一种 Web 服务器软件的开源版本。Tornado 和现在的主流 Web 服务器框架（包括大多数 Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。

 得利于其 非阻塞的方式和对epoll的运用，Tornado 每秒可以处理数以千计的连接，因此 Tornado 是实时 Web 服务的一个 理想框架。

如何提高python的运行效率

使用生成器；关键代码使用外部功能包（Cython，pylnlne，pypy，pyrex）；针对循环的优化--尽量避免在循环中访问变量的属性

简单谈下GIL

Python代码的执行由Python 虚拟机(也叫解释器主循环，CPython版本)来控制，Python 在设计之初就考虑到要在解释器的主循环中，同时只有一个线程在执行，即在任意时刻，只有一个线程在解释器中运行。对Python 虚拟机的访问由全局解释器锁（GIL）来控制，正是这个锁能保证同一时刻只有一个线程在运行。

在多线程环境中，Python 虚拟机按以下方式执行：

1. 设置GIL  
2. 切换到一个线程去运行  
3. 运行：  
    a. 指定数量的字节码指令，或者

b. 线程主动让出控制（可以调用time.sleep(0)）  
4. 把线程设置为睡眠状态  
5. 解锁GIL  
6. 再次重复以上所有步骤

 在调用外部代码（如C/C++扩展函数）的时候，GIL 将会被锁定，直到这个函数结束为止（由于在这期间没有Python 的字节码被运行，所以不会做线程切换）。