# 第一阶段面试题

## 一、Python基础

### 1. continue和 break有什么区别？

**答案：**

break和continue都是用来控制循环结构的。

1. break：提前终止循环可以使用break来完成。break完全结束一个循环，跳出循环体执行循环后面的语句。
2. continue：理解为continue是跳过当次循环中剩下的语句，执行下一次循环。
3. 区别：continue只终止本次循环，break则完全终止循环

### 2.Python 中的作用域？

**答案：**

Python 中，一个变量的作用域总是由在代码中被赋值的地方所决定的。

当 Python 遇到一个变量的话他会按照这样的顺序进行搜索：

本地作用域（Local）→当前作用域被嵌入的本地作用域（Enclosing locals）

→全局/模块作用域（Global）→内置作用域（Built-in）

1. **谈谈对闭包的理解？**

**答案：**

闭包(closure)是函数式编程的重要的语法结构。闭包也是一种组织代码的结构，

它同样提高了代码的可重复使用性。

当一个内嵌函数引用其外部作作用域的变量,我们就会得到一个闭包. 总结一下,

创建一个闭包必须满足以下几点:

必须有一个内嵌函数

内嵌函数必须引用外部函数中的变量

外部函数的返回值必须是内嵌函数

感觉闭包还是有难度的,几句话是说不明白的,还是查查相关资料.

重点是函数运行后并不会被撤销,就像16题的instance字典一样,当函数运行完

后,instance 并不被销毁,而是继续留在内存空间里.这个功能类似类里的类变

量,只不过迁移到了函数上.

闭包就像个空心球一样,你知道外面和里面,但你不知道中间是什么样.

1. **Python 里的拷贝？理解引用和 copy(),deepcopy()的区别。**

**答案：**

import copy

a = [1, 2, 3, 4, ['a', 'b']] #原始对象

b = a #赋值，传对象的引用

c = copy.copy(a) #对象拷贝，浅拷贝

d = copy.deepcopy(a) #对象拷贝，深拷贝

a.append(5) #修改对象 a

a[4].append('c') #修改对象 a 中的['a', 'b']数组对象

print 'a = ', a

print 'b = ', b

print 'c = ', c

print 'd = ', d

输出结果：

a = [1, 2, 3, 4, ['a', 'b', 'c'], 5]

b = [1, 2, 3, 4, ['a', 'b', 'c'], 5]

c = [1, 2, 3, 4, ['a', 'b', 'c']]

d = [1, 2, 3, 4, ['a', 'b']]

1. **Python 垃圾回收机制？**

**答案：**

Python GC 主要使用引用计数（reference counting）来跟踪和回收垃圾。在引

用计数的基础上，通过“标记-清除”（mark and sweep）解决容器对象可能产

生的循环引用问题，通过“分代回收”（generation collection）以空间换时

间的方法提高垃圾回

**6.什么是 lambda 函数？它有什么好处?**

**答案：**

lambda 表达式，通常是在需要一个函数，但是又不想费神去命名一个函数

的场合下使用，也就是指匿名函数

lambda 函数：首要用途是指点短小的回调函数

lambda [arguments]:expression

>>> a=lambdax,y:x+y

>>> a(3,11)

**7.请写出一段 Python 代码实现删除一个list 里面的重复元素？**

**答案：**

1. 使用 set 函数，set(list)

2. 使用字典函数，

>>>a=[1,2,4,2,4,5,6,5,7,8,9,0]

>>> b={}

>>>b=b.fromkeys(a)

>>>c=list(b.keys())

>>> c

**8.Python 里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别）**

**答案：**

赋值（=），就是创建了对象的一个新的引用，修改其中任意一个变量都会影响到另一个。

浅拷贝：创建一个新的对象，但它包含的是对原始对象中包含项的引用（如果用

引用的方式修改其中一个对象，另外一个也会修改改变）{1,完全切片方法；2，

工厂函数，如 list()；3，copy 模块的 copy()函数}

深拷贝：创建一个新的对象，并且递归的复制它所包含的对象（修改其中一个，

另外一个不会改变）{copy 模块的 deep.deepcopy()函数}

**9.介绍一下 except 的用法和作用？**

**答案：**

try…except…except…[else…][finally…]

执行 try 下的语句，如果引发异常，则执行过程会跳到 except 语句。对每个except 分支顺序尝试执行，如果引发的异常与 except 中的异常组匹配，执行相应的语句。

如果所有的 except 都不匹配，则异常会传递到下一个调用本代码的最高层 try代码中。

try 下的语句正常执行，则执行 else 块代码。如果发生异常，就不会执行

如果存在 finally 语句，最后总是会执行。

**10.如何用 Python 来进行查询和替换一个文本字符串？**

**答案：**

可以使用 re 模块中的 sub()函数或者 subn()函数来进行查询和替换，

格式：sub(replacement, string[,count=0])（replacement 是被替换成的文本，

string 是需要被替换的文本，count 是一个可选参数，指最大被替换的数量）

>>> import re

>>>p=re.compile(‘blue|white|red’)

>>>print(p.sub(‘colour’,'blue socks and red shoes’))

colour socks and colourshoes

>>>print(p.sub(‘colour’,'blue socks and red shoes’,count=1))

colour socks and redshoes

subn()方法执行的效果跟 sub()一样，不过它会返回一个二维数组，包括替换后

的新的字符串和总共替换的数量

**11.Python 里面 match()和 search()的区别？**

**答案：**

re 模块中 match(pattern,string[,flags]),检查 string 的开头是否与pattern 匹配。

re 模块中 re.search(pattern,string[,flags]),在 string 搜索 pattern 的第一个匹配值。

>>>print(re.match(‘super’, ‘superstition’).span())

(0, 5)

>>>print(re.match(‘super’, ‘insuperable’))

None

>>>print(re.search(‘super’, ‘superstition’).span())

(0, 5)

>>>print(re.search(‘super’, ‘insuperable’).span())

(2, 7)

1. **Python 里面如何生成随机数？**

**答案：**

random 模块

随机整数：random.randint(a,b)：返回随机整数 x,a<=x<=b

random.randrange(start,stop,[,step])：返回一个范围在(start,stop,step)

之间的随机整数，不包括结束值。

随机实数：random.random( ):返回 0 到 1 之间的浮点数

random.uniform(a,b):返回指定范围内的浮点数

1. **如何在一个 function 里面设置一个全局的变量？**

**答案：**

解决方法是在 function 的开始插入一个 global 声明：

def f():

global x

1. **单引号，双引号，三引号的区别?**

**答案：**

单引号和双引号是等效的，如果要换行，需要符号(\),三引号则可以直接换

行，并且可以包含注释

如果要表示 Let’s go 这个字符串

单引号：s4 = ‘Let\’s go’

双引号：s5 = “Let’s go”

s6 = ‘I realy like“python”!’

这就是单引号和双引号都可以表示字符串的原因

**15.Python2 和 3 的区别？**

**答案：**

print 不再是语句，而是函数，比如原来是 print 'abc' 现在是 print('abc')

但是 python2.6+ 可以使用 from \_\_future\_\_ import print\_function 来实现相同功能

在 Python 3 中，没有旧式类，只有新式类，也就是说不用再像这样 class

Foobar(object): pass 显式地子类化 object

但是最好还是加上. 主要区别在于 old-style 是 classtype 类型而 new-style 是type 类型

原来 1/2（两个整数相除）结果是 0，现在是 0.5 了

python 2.2+ 以上都可以使用 from \_\_future\_\_ import division 实现改特性, 同时注

意 // 取代了之前的 / 运算

新的字符串格式化方法 format 取代%

错误, 从 python2.6+ 开始已经在 str 和 unicode 中有该方法, 同时 python3 依然支持 % 算符

xrange 重命名为 range

同时更改的还有一系列内置函数及方法, 都返回迭代器对象, 而不是列表或者

元组, 比如 filter, map, dict.items 等

!=取代 < >

python2 也很少有人用 < > 所以不算什么修改

long 重命名为 int

不完全对, python3 彻底废弃了 long+int 双整数实现的方法, 统一为 int , 支持高精度整数运算.

except Exception, e 变成 except (Exception) as e

只有 python2.5 及以下版本不支持该语法. python2.6 是支持的. 不算新东西

exec 变成函数

类似 print() 的变化, 之前是语句.

**16.下面代码会输出什么：**

def f(x,l=[]):

for i in range(x):

l.append(i\*i)

print l

f(2)

f(3,[3,2,1])

f(3)

**答案：**

[0, 1]

[3, 2, 1, 0, 1, 4]

[0, 1, 0, 1, 4]

**17.这两个参数是什么意思：\*args，\*\*kwargs？我们为什么要使用它们？**

**答案：**

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：

def f(\*args,\*\*kwargs): print args, kwargs

l = [1,2,3]

t = (4,5,6)

d = {'a':7,'b':8,'c':9}

f()

f(1,2,3) # (1, 2, 3) {}

f(1,2,3,"groovy") # (1, 2, 3, 'groovy') {}

f(a=1,b=2,c=3) # () {'a': 1, 'c': 3, 'b': 2}

f(a=1,b=2,c=3,zzz="hi") # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'}

f(1,2,3,a=1,b=2,c=3) # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2}

f(\*l,\*\*d) # (1, 2, 3) {'a': 7, 'c': 9, 'b': 8}

f(\*t,\*\*d) # (4, 5, 6) {'a': 7, 'c': 9, 'b': 8}

f(1,2,\*t) # (1, 2, 4, 5, 6) {}

f(q="winning",\*\*d) # () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}

f(1,2,\*t,q="winning",\*\*d) # (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}

def f2(arg1,arg2,\*args,\*\*kwargs): print arg1,arg2, args, kwargs

f2(1,2,3) # 1 2 (3,) {}

f2(1,2,3,"groovy") # 1 2 (3, 'groovy') {}

f2(arg1=1,arg2=2,c=3) # 1 2 () {'c': 3}

f2(arg1=1,arg2=2,c=3,zzz="hi") # 1 2 () {'c': 3, 'zzz': 'hi'}

f2(1,2,3,a=1,b=2,c=3) # 1 2 (3,) {'a': 1, 'c': 3, 'b': 2}

f2(\*l,\*\*d) # 1 2 (3,) {'a': 7, 'c': 9, 'b': 8}

f2(\*t,\*\*d) # 4 5 (6,) {'a': 7, 'c': 9, 'b': 8}

f2(1,2,\*t) # 1 2 (4, 5, 6) {}

f2(1,1,q="winning",\*\*d) # 1 1 () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}

f2(1,2,\*t,q="winning",\*\*d) # 1 2 (4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}

**19.简要描述 Python 的垃圾回收机制（garbage collection）。**

**答案：**

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。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。

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

**20.简述函数式编程**

**答案：**

在函数式编程中，函数是基本单位，变量只是一个名称，而不是一个存储单元。

除了匿名函数外，Python 还使用 fliter(),map(),reduce(),apply()函数来支持函数式编程。

**21.什么是匿名函数，匿名函数有什么局限性？**

**答案：**

匿名函数，也就是 lambda 函数，通常用在函数体比较简单的函数上。匿名函数

顾名思义就是函数没有名字，因此不用担心函数名冲突。不过 Python 对匿名函

数的支持有限，只有一些简单的情况下可以使用匿名函数。

**22.如何捕获异常，常用的异常机制有哪些？**

**答案：**

如果我们没有对异常进行任何预防，那么在程序执行的过程中发生异常，就会中断程序，调用 python 默认的异常处理器，并在终端输出异常信息。try...except...finally 语句:当 try 语句执行时发生异常，回到 try 语句层，寻找后面是否有 except 语句。找到 except 语句后，会调用这个自定义的异常处理器。except 将异常处理完毕后，程序继续往下执行。finally 语句表示，无论异常发生与否，finally 中的语句都要执行。

assert 语句：判断 assert 后面紧跟的语句是 True 还是 False，如果是 True 则继续执行 print，如果是 False 则中断程序，调用默认的异常处理器，同时输出assert 语句逗号后面的提示信息。

with 语句：如果 with 语句或语句块中发生异常，会调用默认的异常处理器处理，但文件还是会正常关闭。

**23.去除列表中的重复元素？**

**答案：**

#用集合

list(set(l))

#用字典

l1 = ['b','c','d','b','c','a','a']

l2 = {}.fromkeys(l1).keys()

print l2

1

2

3

l1 = ['b','c','d','b','c','a','a']

l2 = {}.fromkeys(l1).keys()

print l2

#用字典并保持顺序

l1 = ['b','c','d','b','c','a','a']

l2 = list(set(l1))

l2.sort(key=l1.index)

print l2

#列表推导式

l1 = ['b','c','d','b','c','a','a']

l2 = []

[l2.append(i) for i in l1 if not i in l2]

**24.变态台阶问题**

**一只青蛙一次可以跳上 1 级台阶，也可以跳上 2 级……它也可以跳上 n 级。求该青蛙跳上一个 n 级的台阶总共有多少种跳法**

**答案：**

第一种方法：台阶问题/斐波纳挈

fib = lambda n: n if n <= 2 else fib(n - 1) + fib(n - 2)

第二种记忆方法

def memo(func):

cache = {}

def wrap(\*args):

if args not in cache:

cache[args] = func(\*args)

return cache[args]

return wrap

@ memo

def fib(i):

if i < 2:

return 1

return fib(i-1) + fib(i-2)

第三种方法

def fib(n):

a, b = 0, 1

for \_ in xrange(n):

a, b = b, a + b

return b

**25.Python中list和tuples的差别是什么？**

**答案：**

列表里的内容是可以改变的，增删改都可以，tuple则不行；

python中字典的key必须是可哈希的，不可变的所以tuple可以作为字典的键，而list不行；

对于使用场景tuple适合一些只读的数据，如python链接mysql得到的结果就是用tuple，而list则在列表长度不固定或者需要有变动的数据中使用tuple的性能比list好一些，tuple比list更省内存

**26.代码举例什么是decorator。**

**答案：**

decorator是一个装饰器函数，作用是用来包装另一个函数。

装饰器函数出入的是一个函数返回的也是一个函数。

def fn(fc):

def f():

print('<a>'+fc()+'</a>')

return f

@fn

def f2():

return '我是一个超链接'

f2() #<a>我是一个超链接</a>

**27.单引号，双引号，三引号的区别**

**答案：**

单引号和双引号是等效的，如果要换行，需要符号(\),三引号则可以直接换行，并且可 以包含注释

如果要表示Let’s go 这个字符串

单引号：s4 = ‘Let\’s go’

双引号：s5 = “Let’s go”

s6 = ‘I realylike“python”!’

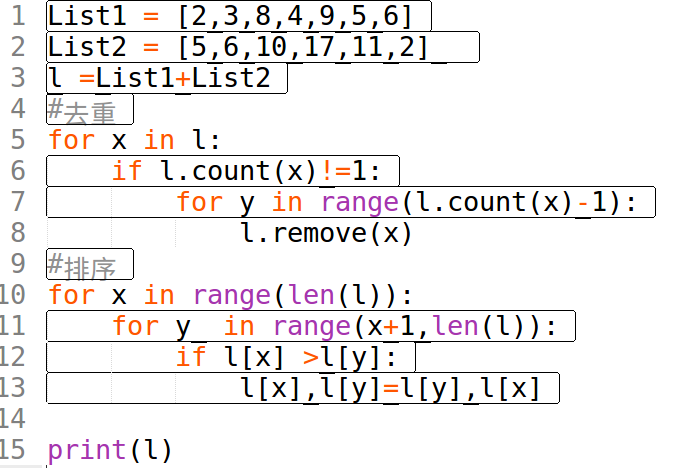
这就是单引号和双引号都可以表示字符串的原因了

**28.请用自己的算法,按升序合并如下两个List , 并去除重复的元素**

List1 = [2,3,8,4,9,5,6]

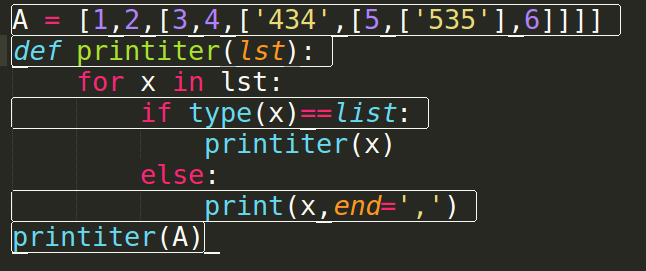
List2 = [5,6,10,17,11,2]

**答案：**



**29.有一个多层嵌套的列表 A = [1,2,[3,4,[‘434’,[…]]]],请写一段代码遍历A中的美个元素并打印出来.**

**答案：**



**30.用自己的话说明迭代器和生成器，它们之间的关系？**

**答案：**

迭代器：指的是一个重复的过程，每一次重复称为一次迭代，并且每一次重复的结果是下一次重复的初始值

生成器：只要在函数体内出现yield关键字，那么再执行函数就不会执行函数代码，会得到一个结果，该结果就是生成器

**31.Python是如何进行内存管理的?**

**答案：**

python内部使用引用计数，来保持追踪内存中的对象，Python内部记录了对象有多少个引用，即引用计数，当对象被创建时就创建了一个引用计数，当对象不再需要时，

这个对象的引用计数为0时，它被垃圾回收。

**32.字符串格式化:% 和 format的区别**

**答案：**

符串的format函数非常灵活，很强大,可以接受的参数不限个数，并且位置可以不按顺序，而且有较为强大的格式限定符(比如:填充，对齐,精度等)

## 二、OOP

### **1.新式类和旧式类的区别,如何确保使用的类是新式类?**

**答案：**

为了统一类(class)和类型(type)，python 在 2.2 版本引进来新式类。在 2.1 版

本中，类和类型是不同的。

为了确保使用的是新式类，有以下方法：

放在类模块代码的最前面 \_\_metaclass\_\_ = type

从内建类 object 直接或者间接地继承

在 python3 版本中，默认所有的类都是新式类。

1. **举例说明Python中的继承类型.**

**答案：**

python中的继承包括单继承和多继承。在单继承中子类无\_\_init\_\_时，会直接继承父类的\_\_init\_\_，多继承时，子类无\_\_init\_\_则从父类列表中逐个判断，是否有\_\_init\_\_直到有时则继承。

class A:

pass

class B:

pass

单继承:

class C(A):

pass

多继承：

class D(A,B):

Pass

1. **谈一下类方法、实例方法和静态方法的区别？**

**答案：**

1. 类方法只能访问'类变量'的方法

2. 类方法需要使用@classmethod装饰器定义

3. 类方法的第一个参数是类实例，约定写为cls

说明：

　　 类(实例)和对象(实例)都可以调用类方法

类方法不能访问实例变量

静态方法：@staticmethod

　　 静态方法是普通函数，

　　 静态方法定义在类的内部，只能凭借该类或实例调用

静态方法需要使用@staticmethod装饰器定义

静态方法写普通函数定义相同，不需要传入self和cls 参数

说明：

　　 类和实例都可以调用静态方法

静态方法不能访问类变量和实例变量

1. **请阐述\_\_new\_\_ 和 \_\_init\_\_ 的区别？**

**答案：**

1、\_\_new\_\_是一个静态方法,而\_\_init\_\_是一个实例方法.

2、\_\_new\_\_方法会返回一个创建的实例,而\_\_init\_\_什么都不返回.

3、只有在\_\_new\_\_返回一个cls的实例时后面的\_\_init\_\_才能被调用.

4、当创建一个新实例时调用\_\_new\_\_,初始化一个实例时用\_\_init\_\_.