# 第一阶段面试题

## 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 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']]

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

**答案：**

1、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。

2、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}

3、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}

### 4.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

### 5.下面代码会输出什么：为什么？

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：**

l=[0, 1]

[3,2,1]覆盖l,结果为：[3, 2, 1, 0, 1, 4]

l=[0,1],append.[0,1,4],结果为：[0, 1, 0, 1, 4]

### 6.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':

### 7. 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消失，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

### 8.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

#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]

### 9.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

### 10.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

### 11.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 = [2,3,8,2,3,8,4,9,5,6]

List2 = [5,6,10,17,11,2,8,4,9]

*# 合并*

List1+=List2

*# 升序*

**for** i **in** range(len(List1)):

**for** j **in** range(len(List1)-i-1):

**if** List1[j]>List1[j+1]:

List1[j] , List1[j + 1]=List1[j+1] , List1[j]

print(List1)

print(**''**)

*# 去重*

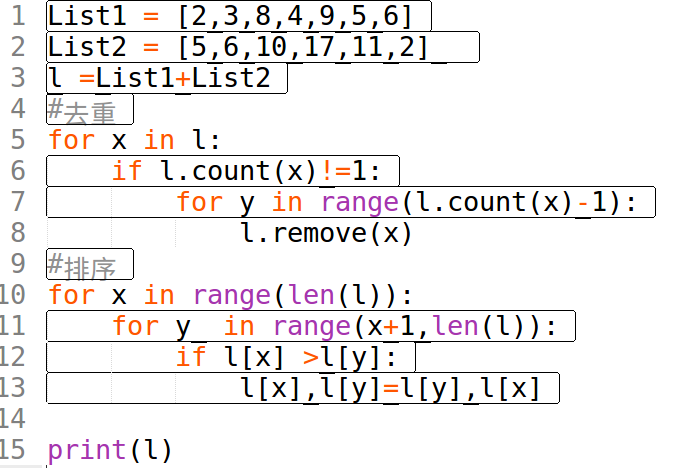
l01=list(set(List1))

print(l01)

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

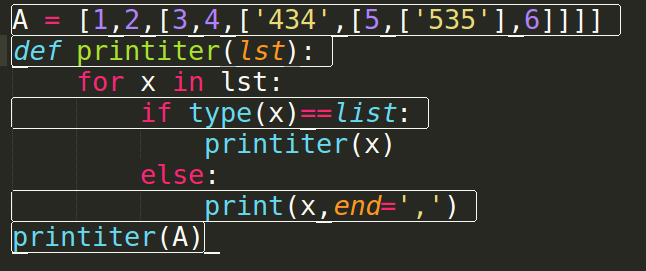
List1 = [2,3,8,4,9,5,6]

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



### 12.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中的每个元素并打印出来.

**答案：**



## Python - OOP

### 1.游戏编程：

按以下要求定义一个乌龟类和鱼类并尝试编写游戏

假设游戏场景为范围（x,y）为0<=x<=10,0<=y<=10

游戏生成1只乌龟和10条鱼

它们的移动方向均随机

乌龟的最大移动能力为2（它可以随机选择1还是2移动），鱼儿的最大移动能力是1

当移动到场景边缘，自动向反方向移动(场景暂定 10\*10)

乌龟初始化体力为100（上限）

乌龟每移动一次，体力消耗1

当乌龟和鱼坐标重叠，乌龟吃掉鱼，乌龟体力增加20

鱼暂不计算体力

当乌龟体力值为0（挂掉）或者鱼儿的数量为0游戏结束

分析：

1.定义两个类

乌龟类：move() , eat() , 属性 self.x self.y self.power

鱼类：move(),属性 self.x self.y

1个实例化龟类，10个实例化鱼类

move() x,y 随机 1 or 2 如果等于那么 eat()

import random

#乌龟类

class Turtle:

def \_\_init\_\_(self):

self.power=100 #体力

#乌龟坐标

self.x=random.randint(0,10)

self.y=random.randint(0,10)

#乌龟移动的方法：移动方向均随机 第四条

def move(self):

#计算移动后的新位置（只有四种可能）

new\_x=self.x+random.choice([1,2,-1,-2])

new\_y=self.y+random.choice([1,2,-1,-2])

#判断移动后是否超出边界

if new\_x<0:

self.x=0-new\_x

elif new\_x>10:

self.x=10-(new\_x-10)

else:

#不越界则移动乌龟的位置

self.x=new\_x

if new\_y<0:

self.y=0-new\_y

elif new\_y>10:

self.y=10-(new\_y-10)

else:

#不越界则移动乌龟的位置

self.y=new\_y

self.power-=1 #乌龟每移动一次，体力消耗1

def eat(self):

self.power+=20 #乌龟吃掉鱼，乌龟体力增加20

if self.power>100:

self.power=100 #乌龟体力100（上限）

#鱼类

class Fish:

def \_\_init\_\_(self):

#鱼坐标

self.x=random.randint(0,10)

self.y=random.randint(0,10)

def move(self):

#计算移动后的新位置（只有四种可能）

new\_x=self.x+random.choice([1,-1])

new\_y=self.y+random.choice([1,-1])

#判断移动后是否超出边界

if new\_x<0:

self.x=0-new\_x

elif new\_x>10:

self.x=10-(new\_x-10)

else:

#不越界则移动鱼的位置

self.x=new\_x

if new\_y<0:

self.y=0-new\_y

elif new\_y>10:

self.y=10-(new\_y-10)

else:

#不越界则移动鱼的位置

self.y=new\_y

#开始测试数据

tur=Turtle() #生成1只乌龟

fish=[] #生成10条鱼

for item in range(10):

fish.append(Fish()) #把生成的鱼放到鱼缸里

#判断游戏是否结束：当乌龟体力值为0（挂掉）或者鱼儿的数量为0游戏结束

if tur.power<0 or len(fish)==0:

print("Game Over ~")

#游戏开始

i=1

while True:

if tur.power<=0:

print('乌龟体力不支')

break

if len(fish)==0:

print('乌龟吃完了所有的鱼')

break

#首先乌龟迈出第一步

tur.move()

print('乌龟迈出第%d步,体力值为%d'%(i,tur.power))

print(tur.x,tur.y) #乌龟移动后

for item in fish:

item.move()

if item.x==tur.x and item.y==tur.y:

tur.eat()

fish.remove(item)

print("死了一只鱼")

print("乌龟最新体力值为 %d"%tur.power)

i+=1