**1、Python如何实现单例模式？**

Python有两种方式可以实现单例模式，下面两个例子使用了不同的方式实现单例模式：

1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | class Singleton(type):  def \_\_init\_\_(cls, name, bases, dict):  super(Singleton, cls).\_\_init\_\_(name, bases, dict)  cls.instance = None  def \_\_call\_\_(cls, \*args, \*\*kw):  if cls.instance is None:  cls.instance = super(Singleton, cls).\_\_call\_\_(\*args, \*\*kw)  return cls.instance  class MyClass(object):  \_\_metaclass\_\_ = Singleton  print MyClass()  print MyClass()  使用decorator来实现单例模式  def singleton(cls):  instances = {}  def getinstance():  if cls not in instances:  instances[cls] = cls()  return instances[cls]  return getinstance  @singleton  class MyClass:  … |

**2：什么是lambda函数？**

Python允许你定义一种单行的小函数。定义lambda函数的形式如下：labmda 参数：表达式lambda函数默认返回表达式的值。你也可以将其赋值给一个变量。lambda函数可以接受任意个参数，包括可选参数，但是表达式只有一个：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> g = lambda x, y: x\*y  >>> g(3,4)  12  >>> g = lambda x, y=0, z=0: x+y+z  >>> g(1)  1  >>> g(3, 4, 7)  14 |

也能够直接使用lambda函数，不把它赋值给变量：

>>> (lambda x,y=0,z=0:x+y+z)(3,5,6)  
14

如果你的函数非常简单，只有一个表达式，不包含命令，可以考虑lambda函数。否则，你还是定义函数才对，毕竟函数没有这么多限制。

**3：Python是如何进行类型转换的？**

Python提供了将变量或值从一种类型转换成另一种类型的内置函数。int函数能够将符合数学格式数字型字符串转换成整数。否则，返回错误信息。

|  |  |
| --- | --- |
| 1  2  3  4 | >>> int(”34″)  34  >>> int(”1234ab”) #不能转换成整数  ValueError: invalid literal for int(): 1234ab |

函数int也能够把浮点数转换成整数，但浮点数的小数部分被截去。

|  |  |
| --- | --- |
| 1  2  3  4 | >>> int(34.1234)  34  >>> int(-2.46)  -2 |

函数°oat将整数和字符串转换成浮点数：

|  |  |
| --- | --- |
| 1  2  3  4 | >>> float(”12″)  12.0  >>> float(”1.111111″)  1.111111 |

函数str将数字转换成字符：

|  |  |
| --- | --- |
| 1  2  3  4 | >>> str(98)  ‘98′  >>> str(”76.765″)  ‘76.765′ |

整数1和浮点数1.0在python中是不同的。虽然它们的值相等的，但却属于不同的类型。这两个数在计算机的存储形式也是不一样。

**4：Python如何定义一个函数**

函数的定义形式如下：

|  |  |
| --- | --- |
| 1  2 | def <name>(arg1, arg2,… argN):  <statements> |

函数的名字也必须以字母开头，可以包括下划线“ ”,但不能把Python的关键字定义成函数的名字。函数内的语句数量是任意的，每个语句至少有一个空格的缩进，以表示此语句属于这个函数的。缩进结束的地方，函数自然结束。  
下面定义了一个两个数相加的函数：

|  |  |
| --- | --- |
| 1  2  3  4 | >>> def add(p1, p2):  print p1, “+”, p2, “=”, p1+p2  >>> add(1, 2)  1 + 2 = 3 |

函数的目的是把一些复杂的操作隐藏，来简化程序的结构，使其容易阅读。函数在调用前，必须先定义。也可以在一个函数内部定义函数，内部函数只有在外部函数调用时才能够被执行。程序调用函数时，转到函数内部执行函数内部的语句，函数执行完毕后，返回到它离开程序的地方，执行程序的下一条语句。

**5：Python是如何进行内存管理的？**

Python的内存管理是由Python得解释器负责的，开发人员可以从内存管理事务中解放出来，致力于应用程序的开发，这样就使得开发的程序错误更少，程序更健壮，开发周期更短

**6：如何反序的迭代一个序列？**

how do I iterate over a sequence in reverse order

如果是一个list, 最快的解决方案是：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | list.reverse()  try:  for x in list:  “do something with x”  finally:  list.reverse() |

如果不是list, 最通用但是稍慢的解决方案是：

|  |  |
| --- | --- |
| 1  2  3 | for i in range(len(sequence)-1, -1, -1):  x = sequence[i]  <do something with x> |

**7：Python里面如何实现tuple和list的转换？**

函数tuple(seq)可以把所有可迭代的(iterable)序列转换成一个tuple, 元素不变，排序也不变。  
例如，tuple([1,2,3])返回(1,2,3), tuple('abc')返回('a'.'b','c').如果参数已经是一个tuple的话，函数不做任何拷贝而直接返回原来的对象，所以在不确定对象是不是tuple的时候来调用tuple()函数也不是很耗费的。  
函数list(seq)可以把所有的序列和可迭代的对象转换成一个list,元素不变，排序也不变。  
例如 list([1,2,3])返回(1,2,3), list('abc')返回['a', 'b', 'c']。如果参数是一个list, 她会像set[:]一样做一个拷贝

**8：Python面试题：请写出一段Python代码实现删除一个list里面的重复元素**

可以先把list重新排序，然后从list的最后开始扫描，代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | if List:  List.sort()  last = List[-1]  for i in range(len(List)-2, -1, -1):  if last==List[i]: del List[i]  else: last=List[i] |

**9：Python文件操作的面试题**

如何用Python删除一个文件？  
使用os.remove(filename)或者os.unlink(filename);  
Python如何copy一个文件？  
shutil模块有一个copyfile函数可以实现文件拷贝

**10：Python里面如何生成随机数？**

标准库random实现了一个随机数生成器，实例代码如下：

|  |  |
| --- | --- |
| 1  2 | import random  random.random() |

它会返回一个随机的0和1之间的浮点数

**11：如何用Python来发送邮件？**

可以使用smtplib标准库。

以下代码可以在支持SMTP监听器的服务器上执行。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import sys, smtplib  fromaddr = raw\_input(”From: “)  toaddrs = raw\_input(”To: “).split(',')  print “Enter message, end with ^D:”  msg = ”  while 1:  line = sys.stdin.readline()  if not line:  break  msg = msg + line  # 发送邮件部分  server = smtplib.SMTP('localhost')  server.sendmail(fromaddr, toaddrs, msg)  server.quit() |

**12：Python里面如何拷贝一个对象？**

一般来说可以使用copy.copy()方法或者copy.deepcopy()方法，几乎所有的对象都可以被拷贝  
一些对象可以更容易的拷贝，Dictionaries有一个copy方法：

newdict = olddict.copy()

**13：有没有一个工具可以帮助查找python的bug和进行静态的代码分析？**

有，PyChecker是一个python代码的静态分析工具，它可以帮助查找python代码的bug, 会对代码的复杂度和格式提出警告  
Pylint是另外一个工具可以进行coding standard检查。

**14：如何在一个function里面设置一个全局的变量？**

解决方法是在function的开始插入一个global声明：

|  |  |
| --- | --- |
| 1  2 | def f()  global x |

15：用Python匹配HTML tag的时候，<.>和<.?>有什么区别？

当重复匹配一个正则表达式时候， 例如<.\*>, 当程序执行匹配的时候，会返回最大的匹配值  
例如：

|  |  |
| --- | --- |
| 1  2  3 | import re  s = ‘<html><head><title>Title</title>'  print(re.match('<.\*>', s).group()) |

会返回一个匹配而不是

|  |  |
| --- | --- |
| 1  2  3 | import re  s = ‘<html><head><title>Title</title>'  print(re.match('<.\*?>', s).group()) |

则会返回  
<.>这种匹配称作贪心匹配 <.?>称作非贪心匹配

**16：Python里面search()和match()的区别？**

match（）函数只检测RE是不是在string的开始位置匹配， search()会扫描整个string查找匹配, 也就是说match（）只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回none  
例如：

|  |  |
| --- | --- |
| 1 | print(re.match('super', 'superstition').span()) |

会返回(0, 5)  
而

|  |  |
| --- | --- |
| 1 | print(re.match('super', ‘insuperable')) |

则返回None  
search()会扫描整个字符串并返回第一个成功的匹配

例如：

|  |  |
| --- | --- |
| 1 | print(re.search('super', 'superstition').span()) |

返回(0, 5)

|  |  |
| --- | --- |
| 1 | print(re.search('super', ‘insuperable').span()) |

返回(2, 7)

**17：如何用Python来进行查询和替换一个文本字符串？**

可以使用sub()方法来进行查询和替换，sub方法的格式为：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

例子：

|  |  |
| --- | --- |
| 1  2  3  4 | import re  p = re.compile('(blue|white|red)')  print(p.sub('colour','blue socks and red shoes'))  print(p.sub('colour','blue socks and red shoes', count=1)) |

输出：

colour socks and colour shoes  
colour socks and red shoes

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

例如：

|  |  |
| --- | --- |
| 1  2  3  4 | import re  p = re.compile('(blue|white|red)')  print(p.subn('colour','blue socks and red shoes'))  print(p.subn('colour','blue socks and red shoes', count=1)) |

输出

|  |  |
| --- | --- |
| 1  2 | ('colour socks and colour shoes', 2)  ('colour socks and red shoes', 1) |

**18：介绍一下except的用法和作用？**

Python的except用来捕获所有异常， 因为Python里面的每次错误都会抛出 一个异常，所以每个程序的错误都被当作一个运行时错误。

一下是使用except的一个例子：

|  |  |
| --- | --- |
| 1  2  3  4 | try:  foo = opne(”file”) #open被错写为opne  except:  sys.exit(”could not open file!”) |

因为这个错误是由于open被拼写成opne而造成的，然后被except捕获，所以debug程序的时候很容易不知道出了什么问题  
下面这个例子更好点：

|  |  |
| --- | --- |
| 1  2  3  4 | try:  foo = opne(”file”) # 这时候except只捕获IOError  except IOError:  sys.exit(”could not open file”) |

**19：Python中pass语句的作用是什么？**

pass语句什么也不做，一般作为占位符或者创建占位程序，pass语句不会执行任何操作，比如：

while False:  
pass

pass通常用来创建一个最简单的类：

class MyEmptyClass:  
pass  
pass在软件设计阶段也经常用来作为TODO，提醒实现相应的实现，比如：

|  |  |
| --- | --- |
| 1  2 | def initlog(\*args):  pass #please implement this |

**20：介绍一下Python下range()函数的用法？**

如果需要迭代一个数字序列的话，可以使用range()函数，range()函数可以生成等差级数。  
如例：

|  |  |
| --- | --- |
| 1  2 | for i in range(5)  print(i) |

这段代码将输出0, 1, 2, 3, 4五个数字

range(10)会产生10个值， 也可以让range()从另外一个数字开始，或者定义一个不同的增量，甚至是负数增量  
range(5, 10)从5到9的五个数字  
range(0, 10, 3) 增量为三， 包括0,3,6,9四个数字  
range(-10, -100, -30) 增量为-30， 包括-10, -40, -70

可以一起使用range()和len()来迭代一个索引序列

例如：

|  |  |
| --- | --- |
| 1  2  3 | a = ['Nina', 'Jim', 'Rainman', 'Hello']  for i in range(len(a)):   print(i, a[i]) |

**21：有两个序列a,b，大小都为n,序列元素的值任意整形数，**

无序；要求：通过交换a,b中的元素，使[序列a元素的和]与[序列b元素的和]之间的差最小。  
将两序列合并为一个序列，并排序，为序列Source

拿出最大元素Big，次大的元素Small

在余下的序列S[:-2]进行平分，得到序列max，min

将Small加到max序列，将Big加大min序列，重新计算新序列和，和大的为max，小的为min。

Python代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | def mean( sorted\_list ):  if not sorted\_list:  return (([],[]))  big = sorted\_list[-1]  small = sorted\_list[-2]  big\_list, small\_list = mean(sorted\_list[:-2])  big\_list.append(small)  small\_list.append(big)  big\_list\_sum = sum(big\_list)  small\_list\_sum = sum(small\_list)  if big\_list\_sum > small\_list\_sum:  return ( (big\_list, small\_list))  else:  return (( small\_list, big\_list))  tests = [ [1,2,3,4,5,6,700,800],  [10001,10000,100,90,50,1],  range(1, 11),  [12312, 12311, 232, 210, 30, 29, 3, 2, 1, 1]  ]  for l in tests:  l.sort()  print  print “Source List: ”, l  l1,l2 = mean(l)  print “Result List: ”, l1, l2  print “Distance: ”, abs(sum(l1)-sum(l2))  print ‘-\*'\*40 |

输出结果

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Source List: [1, 2, 3, 4, 5, 6, 700, 800]  Result List: [1, 4, 5, 800] [2, 3, 6, 700]  Distance:  99  -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*  Source List: [1, 50, 90, 100, 10000, 10001]  Result List: [50, 90, 10000] [1, 100, 10001]  Distance:  38  -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*  Source List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  Result List: [2, 3, 6, 7, 10] [1, 4, 5, 8, 9]  Distance:  1  -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*  Source List: [1, 1, 2, 3, 29, 30, 210, 232, 12311, 12312]  Result List: [1, 3, 29, 232, 12311] [1, 2, 30, 210, 12312]  Distance:  21  -\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\* |