

1. 简述 python GIL 的概念， 以及它对 python 多线程的影响？

Global Interpreter Lock

GIL 为全局解释器锁，Python 解释器为了防止同一时刻多个线程操作同一个数据，造成数据混乱，因此每个线程在执行的过程中都需要先获取 GIL，，使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。保证同一时刻只有一个线程可以执行代码。

2. match 和 search 都是 re 模块下的函数，

match: 从字符串开头开始匹配等于 ^

search: 会搜寻整个字符串，返回匹配到的第一个结果

3. Deepcopy 和 copy 的区别？

copy 仅拷贝对象本身，而不拷贝对象中引用的其它对象。

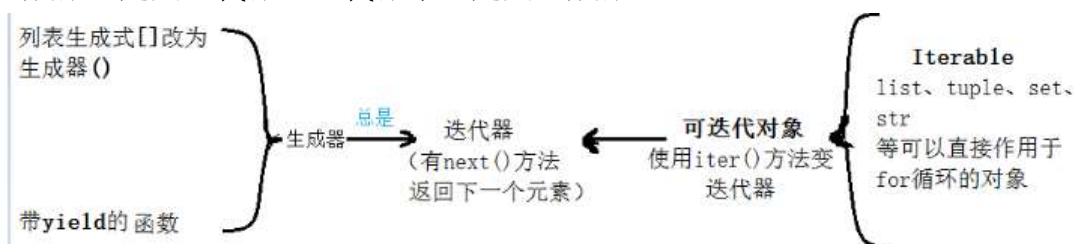
deepcopy 除拷贝对象本身，而且拷贝对象中引用的其它对象。

4. 简述迭代器生成器以及他们之间的区别

列表生成式[] 中括号变 () 括号，generator_ = (x for x in range(10)) 和带 yield 的函数称为生成器

列表、元组、字符串、字典等可以直接作用于 for 循环的可迭代对象，通过 iter() 方法都可以转变为迭代器。

生成器一定是迭代器，迭代器不一定是生成器



Python 中 pass 语句的作用是什么？

pass 语句不会执行任何操作，一般作为占位符使用

判断一个邮箱是否合法

```
email = 'linal994@outlook.com'
ret = re.match(r"[a-zA-Z_0-9]{4,20}@163\.com$", email)
if ret:
    print("%s 符合要求..." % email)
```

如何在一个函数内部修改全局变量

利用 global 修改全局变量

8. 数组和元组之间的区别是什么？

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，比如作为字典的键。

11. 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。

24. Xrange 和 range 的区别是什么？

Xrange 用于返回一个 xrange 对象，而 range 用于返回一个数组。不管那个范围多大，Xrange 都使用同样的内存。

Python 中的 yield 用法

yield 简单来说就是一个生成器，这样函数它记住上次返回时在函数中的位置。对于生成器第二次（或 n 次）调用跳转至该函数。

11、描述数组、链表、队列、堆栈的区别？

数组和链表是数据存储方式的概念，数组在连续的空间中存储数据，而链表可以在非连续的空间中存储数据；

队列和堆栈是描述数据存取方式的概念，队列是先进先出，而堆栈是后进先出；队列和堆栈可以用数组来实现，也可以用链表实现。

随机整数：random.randint(a, b):

返回随机整数 x, $a \leq x \leq b$

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

什么是 lambda 函数？

lambda 函数：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

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。

```
g = lambda x, y: x*y
```

```
>>> g(3,4)
```

```
12
```

```
>>> g = lambda x, y=0, z=0: x+y+z
```

```
>>> g(1)
```

如何用 Python 来进行查询和替换一个文本字符串

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：

```
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
```

介绍一下 except 的用法和作用？

Except 语句用来捕获异常

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。

```
try:
foo = opne(" file" ) # 这时候 except 只捕获 IOError
except IOError:
sys.exit(" could not open file" )
```

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(10) 会产生 10 个值， 也可以让 range() 从另外一个数字开始

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

Python 里面如何拷贝一个对象？（赋值，浅拷贝，深拷贝的区别）

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
{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}

不改变排序 Python 去除列表重复元素的方法

```
m = ['b', 'c', 'd', 'b', 'c', 'a', 'a']  
n = list(set(m))  
  
n.sort(key=m.index)      #保持顺序  
print n
```

`fun(*args, **kwargs)` 中的 `*args`, `**kwargs` 什么意思？

当函数的参数不确定时，可以使用 `*args` 和 `**kwargs`,

`*args` 没有 key 值，传入的是一个列表； `**kwargs` 有 key 值，传入的是一个字典

python2 和 python3 的 `range(100)` 的区别

python2 返回列表，python3 返回迭代器，节约内存

简述 `with` 方法打开处理文件帮我们做了什么？

打开文件在进行读写的时候可能会出现一些异常状况，如果按照常规的 `f.open` 写法，我们需要 `try, except, finally`，做异常判断，并且文件最终不管遇到什么情况，都要执行 `finally` `f.close()` 关闭文件，`with` 方法帮我们实现了 `finally` 中 `f.close`

```

3 f=open("./1.txt","wb")
9 try:
0     f.write("hello world")
1 except:
2     pass
3 finally:
4     f.close()

```

10 个 Linux 常用命令

LS 命令

一作用：显示目录内容，类似 DOS 下的 DIR

cat 命令

一作用：显示文件内容，concatenate 的缩写，类似 dos 的 type 命令。

mv 命令

一作用：更改文件或者目录的名字。

rm 命令

一作用：删除文件命令，类似 dos 的 del 命令

mkdir 命令

一作用：创建目录，类似 dos 的 md 命令

more 命令

一作用：分屏[显示输出](#)结果，同 DOS 下的 more 命令。

grep 命令

一作用：在文件中搜索特定的字符串。

find 命令

一作用：搜索指定目录下的文件

file 命令

一作用：判断文件的类型

chmod 命令

一作用：改变文件存取权限。

29、正则 re.compile 作用

re.compile 是将正则表达式编译成一个对象，加快速度，并重复使用

两个列表[1, 5, 7, 9]和[2, 2, 6, 8]合并为[1, 2, 2, 3, 6, 7, 8, 9]

```
List1=[1, 5, 7, 9]
```

```
List2=[2, 2, 6, 8]
```

```
List1=List1.extend(list2)
```

```
List1=List1.sort(reverse=False)
```

32、用 python 删除文件和用 linux 命令删除文件方法

python: `os.remove(文件名)`

linux: `rm 文件名`

37、正则表达式匹配中，`(.*)` 和 `(.*?)` 匹配区别？

`(.*)` 是贪婪匹配，会把满足正则的尽可能多的往后匹配

`(.*?)` 是非贪婪匹配，会把满足正则的尽可能少匹配

```
97 s("<a>哈哈</a><a>呵呵</a>")
98 import re
99 res1=re.findall("<a>(.)</a>",s)
100 print("贪婪匹配",res1)
101 res2=re.findall("<a>(.*?)</a>",s)
102 print("非贪婪匹配",res2)

贪婪匹配 ['哈哈</a><a>呵呵']
非贪婪匹配 ['哈哈', '呵呵']
[Finished in 0.1s]
```

48、提高 python 运行效率的方法

1、使用生成器，因为可以节约大量内存

2、循环代码优化，避免过多重复代码的执行

3、核心模块用 Cython PyPy 等，提高效率

4、多进程、多线程、协程

5、多个 if elif 条件判断，可以把最有可能先发生的条件放到前面写，这样可以减少程序判断的次数，提高效率

64、简述 `any()` 和 `all()` 方法

`any()`:只要迭代器中有一个元素为真就为真

`all()`:迭代器中所有的判断项返回都是真，结果才为真

python 中什么元素为假？

答案：(0, 空字符串, 空列表、空字典、空元组、None, False)

63、简述多线程、多进程

进程：

- 1、操作系统进行资源分配和调度的基本单位，多个进程之间相互独立
- 2、稳定性好，如果一个进程崩溃，不影响其他进程，但是进程消耗资源大，开启的进程数量有限制

线程：

- 1、CPU 进行资源分配和调度的基本单位，线程是进程的一部分，是比进程更小的能独立运行的基本单位，一个进程下的多个线程可以共享该进程的所有资源
- 2、如果 IO 操作密集，则可以多线程运行效率高，缺点是如果一个线程崩溃，都会造成进程的崩溃

应用：

IO 密集的用多线程，在用户输入，sleep 时候，可以切换到其他线程执行，减少等待的时间

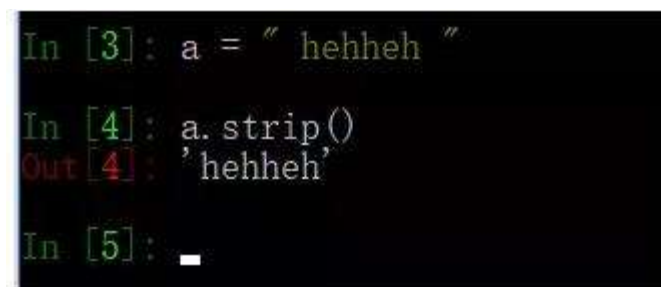
CPU 密集的用多进程，因为假如 IO 操作少，用多线程的话，因为线程共享一个全局解释器锁，当前运行的线程会霸占 GIL，其他线程没有 GIL，就不能充分利用多核 CPU 的优势

、请将[i for i in range(3)]改成生成器

生成器是特殊的迭代器，

- 1、列表表达式的【】改为（）即可变成生成器

a = " hehheh ",去除收尾空格



```
In [3]: a = " hehheh "
In [4]: a.strip()
Out [4]: 'hehheh'
In [5]: _
```

71、举例 sort 和 sorted 对列表排序，list=[0,-1,3,-10,5,9]

```

187
188 # 列表排序
189 list = [0,-1,3,-10,5,9]
190 list.sort(reverse=False)
191 print("list.sort在list基础上修改, 无返回值",list)
192
193 list = [0,-1,3,-10,5,9]
194 res = sorted(list,reverse=False)
195 print("sorted有返回值是新的list",list)
196 print("返回值",res)

```

数字大于2了

list.sort在list基础上修改, 无返回值 [-10, -1, 0, 3, 5, 9]

sorted有返回值是新的list [0, -1, 3, -10, 5, 9]

返回值 [-10, -1, 0, 3, 5, 9]

72、对 list 排序 foo = [-5, 8, 0, 4, 9, -4, -20, -2, 8, 2, -4], 使用 lambda 函数从小到大排序

```

199
200 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4]
201 a=sorted(foo,key=lambda x:x)
202 # foo.sort()
203 print(a)
204

```

数字大于2了

[-20, -5, -4, -4, -2, 0, 2, 4, 8, 8, 9]

73、使用 lambda 函数对 list 排序 foo = [-5, 8, 0, 4, 9, -4, -20, -2, 8, 2, -4], 输出结果为 [0, 2, 4, 8, 8, 9, -2, -4, -4, -5, -20], 正数从小到大, 负数从大到小
(传两个条件, $x < 0$ 和 $\text{abs}(x)$)


```
199
200 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4]
201 a=sorted(foo,key=lambda x:(x<0,abs(x)))
202 # foo.sort()
203 print(a)
204
```

数字大于2了

```
[0, 2, 4, 8, 8, 9, -2, -4, -4, -5, -20]
```

74、列表嵌套字典的排序，分别根据年龄和姓名排序

```
foo = [{"name": "zs", "age": 19}, {"name": "ll", "age": 54},
{"name": "wa", "age": 17}, {"name": "df", "age": 23}]
```

```
204
205 foo = [{"name": "zs", "age": 19}, {"name": "ll", "age": 54},
206         {"name": "wa", "age": 17}, {"name": "df", "age": 23}]
207 a=sorted(foo,key=lambda x:x["age"],reverse=True) ← 年龄从大到小
208 print(a)
209 a=sorted(foo,key=lambda x:x["name"]) ← 姓名从小到大
210 print(a)
211
```

数字大于2了

```
[8, 0, 4, 9, 8, 2, -5, -4, -20, -2, -4]
[{'name': 'll', 'age': 54}, {'name': 'df', 'age': 23}, {'name': 'zs', 'age': 19}, {'name': 'wa', 'age': 17}]
[{'name': 'df', 'age': 23}, {'name': 'll', 'age': 54}, {'name': 'wa', 'age': 17}, {'name': 'zs', 'age': 19}]
[Finished in 0.1s]
```

103、lambda 匿名函数好处

精简代码，lambda 省去了定义函数，

单引号、双引号、三引号用法

1、单引号和双引号没有什么区别，不过单引号不用按 shift，打字稍微快一点。表示字符串的时候，单引号里面可以用双引号，而不用转义字符，反之亦然。

三引号可以直接书写多行，通常用于大段，大篇幅的字符串

```
"""
```

```
hello
```

```
world
```

```
"""
```

29. python 代码得到列表 list 的交集与差集
交集

```
1 b1=[1, 2, 3]
2 b2=[2, 3, 4]
3 b3 = [val for val in b1 if val in b2]
4 print b3
```

差集

```
1 b1=[1, 2, 3]
2 b2=[2, 3, 4]
3 b3 = [val for val in b1 if val not in b2]
4 print b3
```

Python 是如何进行内存管理的

1). 对象的引用计数机制

Python 内部使用引用计数，来保持追踪内存中的对象，所有对象都有引用计数。

引用计数增加的情况：一个对象分配一个新名称

引用计数减少的情况：引用超出作用域或被重新赋值

2). 垃圾回收

当一个对象的引用计数归零时，它将被垃圾收集机制处理掉。

3). 内存池机制

Python 提供了对内存的垃圾收集机制

Pymalloc 机制：为了加速 Python 的执行效率，Python 引入了一个内存池机制，用于管理对小块内存的申请和释放。

4. 如何用 Python 输出一个 Fibonacci 数列？

```
1 a,b = 0, 1
2 while b<100:
3     print (b),
4     a, b = b, a+b
```

如何反序的迭代一个序列？how do I iterate over a sequence in reverse order
`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：

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

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')

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']。

21. 如何用 Python 来发送邮件？

python 实现发送和接收邮件功能主要用到 poplib 和 smtplib 模块。

poplib 用于接收邮件, 而 smtplib 负责发送邮件。

代码如下：

```
1  #! /usr/bin/env python
2  #coding=utf-8
3  import sys
4  import time
5  import poplib
6  import smtplib
7  #邮件发送函数
8  def send_mail():
9      try:
10         handle = smtplib.SMTP('smtp.126.com', 25)
11         handle.login('XXXX@126.com', '*****')
12         msg = 'To: XXXX@qq.com\r\nFrom: XXXX@126.com\r\nSubject: hello\r\n'
13         handle.sendmail('XXXX@126.com', 'XXXX@qq.com', msg)
14         handle.close()
15         return 1
16     except:
17         return 0
18 #邮件接收函数
19 def accpet_mail():
20     try:
21         p=poplib.POP3('pop.126.com')
22         p.user('pythontab@126.com')
23         p.pass_('*****')
```

```

24         ret = p.stat() #返回一个元组:(邮件数, 邮件尺寸)
25         #p.retr(' 邮件号码')方法返回一个元组:(状态信息, 邮件, 邮件尺寸)
26     except poplib.error_proto, e:
27         print "Login failed:", e
28         sys.exit(1)
29
30 #运行当前文件时, 执行 sendmail 和 accpet_mail 函数
31 if __name__ == "__main__":
32     send_mail()
33     accpet_mail()

```

23. 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?

pycheck pylint

python 装饰器

python 装饰器就是用于拓展原来函数功能的一种函数, 这个函数的特殊之处在于它的返回值也是一个函数, 使用 python 装饰器的好处就是在不用更改原函数的代码前提下给函数增加新的功能。