Python—编程八股

本文讨论了Python编程相关的教程、就业情况以及不同难度级别的常见问题解答，涵盖了Python特性、内存管理、函数使用等多方面内容。关键要点包括：

教程与就业：提供AI大模型全套及企业级应用开发、MCP Server开发零基础教程，还提及AI大模型工程师就业现状及方向选择问题。

初级特性：Python是解释型、动态类型、面向对象语言，有Lambda函数、描述符等特性，使用tuple、list、dictionary存储不同类型变量。

内存管理：采用引用计数、垃圾回收、内存池机制管理内存，解决对象引用和内存碎片问题。

工具与变量：Pychecker和Pylint可进行静态分析，可通过config.py文件在模块间共享全局变量。

函数与方法：装饰器可增加函数额外功能，args和kwargs处理可变参数，生成器可挂起并保持状态。

设计模式：单例模式确保类只有一个实例，工厂模式将对象创建过程抽离。

高级特性：super函数调用父类方法，协程有执行效率高、无需锁机制等优点。

初 级

Q1： 说出 Python 的一些特性?

Python 是一种解释型语言。这意味着，与 C 及其变体等语言不同，Python 在运行前不需要编译。其他解释性语言包括PHP和Ruby

Python 是动态类型的，类型之间可以动态转换，不需要在声明变量或类似内容时声明变量的类型

Python 是面向对象的，允许定义类以及组合和继承。在 Python 中，函数是一种对象，它们可以分配给变量，从其他函数返回并传递给函数。类也是一种对象

Q2： 什么是Python中的Lambda 函数?

Lambda 函数 是一个小型匿名函数；它允许快速定义单行函数，类似于C语言的宏，可以用在任何需要函数的地方

lambda的使用方式为：lambda [arg1 [, agr2,.....argn]] : expression

lambda表达式会返回一个函数对象，如果没有变量接受这个返回值的话，它很快就会被丢弃。也正是由于lambda只是一个表达式，所以它可以直接作为list和dict的成员

Q3：Python中局部变量和全局变量的规则是什么?

Python中，在子程序中定义的变量称为局部变量，在程序的一开始定义的变量称为全局变量

全局变量作用域是整个程序，局部变量作用域是定义该变量的子程序

当全局变量与局部变量同名时：在定义局部变量的子程序内，局部变量起作用；在其它地方全局变量起作用

Q4： 什么时候在 Python中使用 `tuple` vs `list` vs？ `dictionary`

使用 tuples 存储一系列不会更改的变量

使用 list 存储一系列可能更改的变量

当您想要关联成对的两个变量时，使用 dictionary

Q5： 什么是python的描述符?

python描述符是一个“绑定行为”的对象属性，在描述符协议中，它可以通过方法重写属性的访问。这些方法有 get (), set (), 和 delete ()。如果这些方法中的任何一个被定义在一个对象中，这个对象就是一个描述符。

描述符可以用来控制对属性的访问行为，实现计算属性、懒加载属性、属性访问控制等功能

Q6： 是否可以在 Python中使用静态方法?

Python 中的静态方法是指静态方法绑定到类而不是该类的对象，可以在没有该类对象的情况下调用静态方法

可以使用staticmethod()或者@staticmethod() 创建静态方法

Q7： 解释 `UnboundLocalError` 异常以及如何避免它?

当局部变量在赋值之前被引用时，会发生 UnboundLocalError

查看以下的代码块：

代码块

```
num = 10
def test_this ( ) :
    num + = 1
print ( num )
test_this ( )
```

运行它将会抛出** `UnboundLocalError` ，原因是Python中，如果内部函数有引用外部函数的同名变量或者全局变量，并且对这个变量有修改，那么python会认为它是一个局部变量，又因为函数中没有num的定义和赋值，所以报错

解决方案一：用global关键字来进行说明该变量是全局变量，修改代码块如下：

代码块

```
num = 10
def test_this ( ) :
    global num
                num + = 1
print ( num )
test_this ( )
```

解决方案二：将变量作为函数参数，传入函数

Q8： Python2和Python3的主要区别?

在 Python 2 中，print 是一条语句，而 Python3 中作为函数存在

Python2 的默认编码是 asscii，而Python3的默认编码是utf-8

Python 2 中的 xrange() 函数在 Python 3 中不再存在。它已被 range() 函数取代，该函数在迭代序列时提高了性能

Q9： Python中如何创建一个对象的拷贝?

Python中的赋值语句‘=’不创建对象的副本，它们只将名称绑定到对象；如果需要拷贝对象，需要使用标准库中的copy模块（import copy）

使用type（object）进行特殊拷贝，属于浅拷贝的一种；或使用copy.copy，进行对象的浅拷贝(shallow copy)，它复制了对象，但对于对象中的元素，依然使用引用

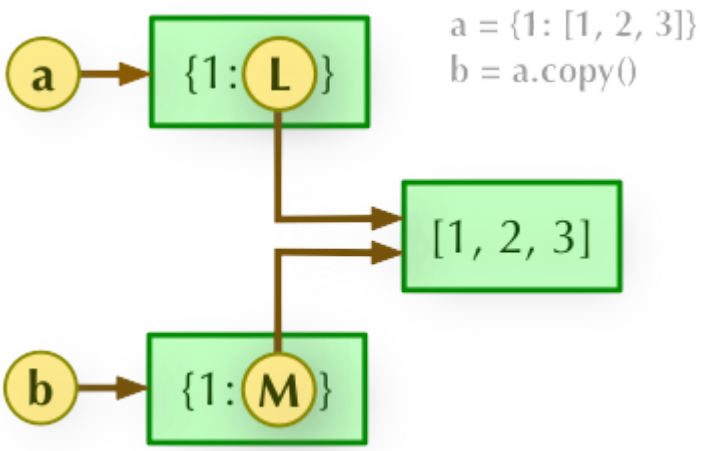
代码块

```
import copy
xs = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
# Make a shallow copy, 进行浅复制
ys = list(xs)
zs = copy.copy(xs)
```

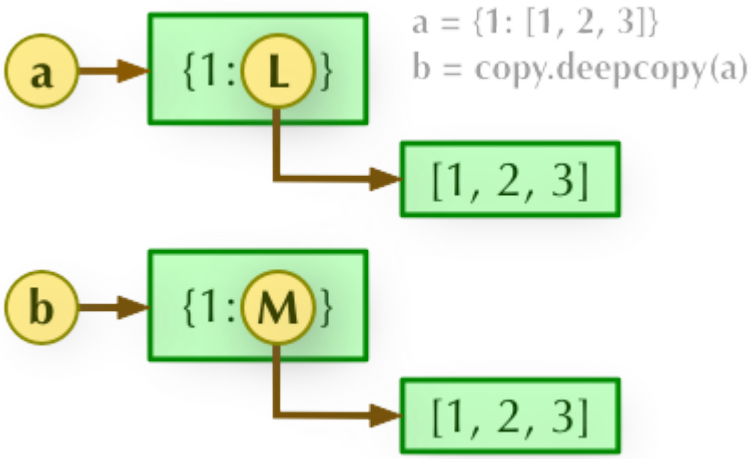
深拷贝：使用copy.deepcopy，它可以进行深拷贝，不仅拷贝了对象，同时也拷贝了对象中的元素，获得了全新的对象，与被拷贝对象完全独立，但这需要牺牲一定的时间和空间

Q10： 解释Python中的浅拷贝与深拷贝

浅拷贝(copy)：拷贝父对象，不会拷贝对象的内部的子对象



深拷贝(deepcopy)： copy 模块的 deepcopy 方法，完全拷贝了父对象及其子对象

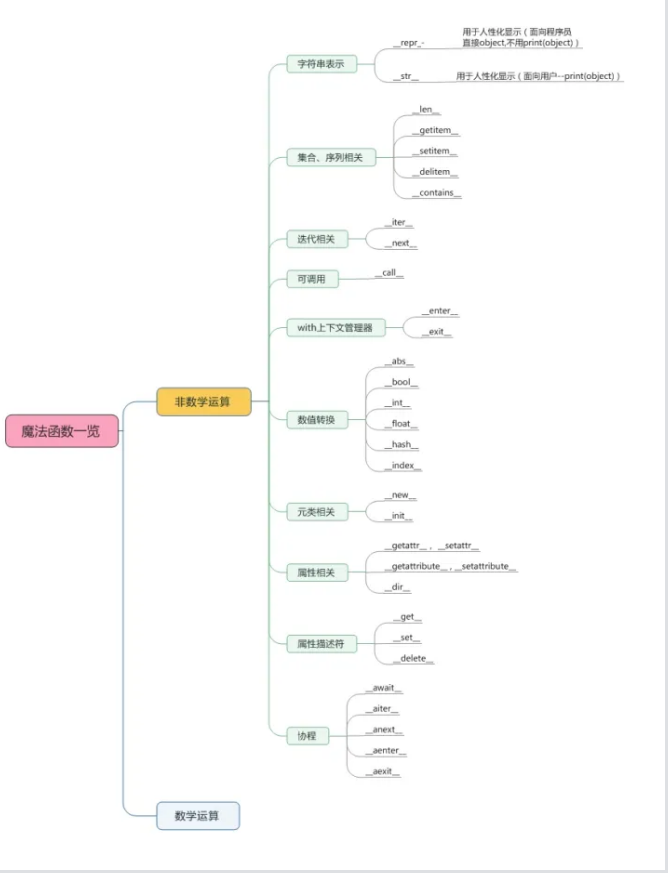


Q11： python中self函数的作用是什么?

在Python类中规定，函数的第一个参数是实例对象本身，并且约定俗成，把其名字写为self。其作用相当于java中的this，表示当前类的对象，可以调用当前类中的属性和方法

Q12： Python中的魔术方法是什么？举几个例子

在python中，任何以‘__’开头和结尾的函数，都是魔法函数，例如：__init__就是一种魔法函数。它可以重载默认函数的行为，允许每个类自行定义基于操作符的特定行为 魔术方法的几个例子是： init, add, _len , __repr__等	



Q13： Python中，什么时候使用pass语句?

pass 语句用作未来代码的占位符。注释和 pass 语句之间的区别在于，虽然解释器完全忽略了注释，但 pass 并没有被忽略

执行 pass 语句时，什么也不会发生，但可以避免在不允许空代码时出现错误。Python语言在循环、函数定义、类定义或 if 语句中不允许使用空代码

Q14： Python 中的不可变对象是什么?

Python 在heap 中分配的对象分成两类：可变对象和不可变对象。 不可变对象属于值类型，可变对象属于引用类型

不可变对象包括int、bool、string、float、tuple；在Python中，这些数据类型属于值类型，本身不允许被修改（不可变类型），数值的修改实际上是让变量指向了一个新的对象（新创建的对象），所以不会发生共享内存问题。 这种方式同Java的不可变对象实现方式相同。原始对象被Python的垃圾回收机制回收

Q15： 每当退出Python 时，是否所有内存都已取消分配?

不是；那些具有对象循环引用或者全局命名空间引用的变量，在 Python 退出是往往不会被释放，另外不会释放 C 库保留的部分内容

中 级

Q16： Python 中是如何管理内存的?

Python的内存管理机制：引用计数、垃圾回收、内存池机制

引用计数：Python通过引用计数来保存内存中的变量追踪，即记录该对象被其他使用的对象引用的次数。Python中有个内部跟踪变量叫做引用计数器，每个变量有多少个引用，简称引用计数。当某个对象的引用计数为0时，就列入了垃圾回收队列

代码块

```
>>> a=[1,2]
>>> import sys
>>> sys.getrefcount(a)  ## 获取对象a的引用次数
2
>>> b=a
>>> sys.getrefcount(a)
3
>>> del b  ## 删除b的引用
>>> sys.getrefcount(a)
2
>>> c=list()
>>> c.append(a) ## 加入到容器中
>>> sys.getrefcount(a)
3
>>> del c  ## 删除容器，引用-1
>>> sys.getrefcount(a)
2
>>> b=a
>>> sys.getrefcount(a)
3
>>> a=[3,4]  ## 重新赋值
>>> sys.getrefcount(a)
2
```

垃圾回收：

内存池：当创建大量消耗小内存的对象时，频繁调用new/malloc会导致大量的内存碎片，致使效率降低。内存池的作用就是预先在内存中申请一定数量的，大小相等的内存块留作备用，当有新的内存需求时，就先从内存池中分配内存给这个需求，不够之后再申请新的内存。这样做最显著的优势就是能够减少内存碎片，提升效率

Q17： Python中range和xrange有什么不同?

range和xrange都是在循环中使用，输出结果一样

range返回的是一个list对象，而xrange返回的是一个生成器对象(xrange object)；xrange不会直接生成一个list，而是每次调用返回其中的一个值，内存空间使用极少，因而性能非常好

python3中取消了xrange，只有range，其内核与python2的xrange一致

Q18： 有什么工具可以帮助Python查找错误或执行静态分析?

Pychecker 和 Pylint 是帮助查找 python 中的错误的静态分析工具

Pychecker 是一个用于静态分析的开源工具，可以从源代码中检测错误并警告错误的样式和复杂性。

Pylint 是高度可配置的，它像特殊程序一样控制警告和错误，是一个广泛的配置文件。 Pylint 也是一个用于静态代码分析的开源工具，它查找编程错误并用于编码标准。它检查每个编程行的长度，根据项目样式检查变量名称。它也可以作为一个独立的程序使用， Pylint 还与 Pycharm、Spyder、Eclipse 和 Jupyter 等 Python IDE 集成

Q19： 如何在python的模块之间共享全局变量?

可以创建一个配置文件config.py并存储整个全局变量以在其中跨模块或脚本共享。通过简单地导入配置文件，定义它的整个全局变量将可用于其他模块，调用方式如下：

代码块

```
import config
config.a = 1
config.b =2
config.c=3
```

Q20： Python `nonlocal` 语句的作用是什么

关键字nonlocal用来在函数或者其他作用域中使用外层（非全局变量）；换句话说，nonlocal用来声明变量不处于当前的函数当中，需要解释器在包含这个函数的函数中寻找nonlocal声明的同名变量，找到后就可以使用这个对象对应的值在当前函数中进行操作

Q21： 解释 Python内存管理是如何工作的

python中的内存管理机制为Pymalloc，python的对象管理主要位于Level+1~Level+3层

Level+3层：对于python内置的对象（比如int,dict等）都有独立的私有内存池，对象之间的内存池不共享，即int释放的内存，不会被分配给float使用

Level+2层：当申请的内存大小小于256KB时，内存分配主要由 Python 对象分配器（ Python’ s object allocator ）实施

Level+1层：当申请的内存大小大于256KB时，由Python原生的内存分配器进行分配，本质上是调用C标准库中的 malloc/realloc 等函数

Q22： `append()` 方法和 `extend()` 方法之间有什么区别?

extend与append方法的相似之处是都将新接收到参数放置到已有列表的后面。 但extend方法只能接收list，且把这个list中的每个元素添加到原list中。 而append方法可以接收任意数据类型的参数，并且简单地追加到list尾部

Q23：解释Python中的装饰器?

装饰器本质上是一个Python函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能；它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景，装饰器是解决这类问题的绝佳设计。有了装饰器，我们就可以抽离出大量与函数功能本身无关的雷同代码到装饰器中并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能

简单的装饰器示例：use_logging 就是一个装饰器，它一个普通的函数，它把执行真正业务逻辑的函数 func 包裹在其中

代码块

```
def use_logging(func):

    def wrapper():
        logging.warn("%s is running" % func.__name__)
        return func()   # 把 foo 当做参数传递进来时，执行func()就相当于执行foo()
    return wrapper

def foo():
    print('i am foo')

foo = use_logging(foo)  # 因为装饰器 use_logging(foo) 返回的时函数对象 wrapper，这条语句相当于  foo = wrapper
foo()                   # 执行foo()就相当于执行 wrapper()
```

Q24： 这些东西是什么意思： `args` ，`kwargs` ？我们为什么要使用它?

args 是 arguments 的缩写，表示位置参数；kwargs 是 keyword arguments 的缩写，表示关键字参数；args和kwargs被用来处理可变参数，允许在调用函数的时候传入多个实参；接收参数后，args会变成一个tuple，kwargs会变成一个dict

Q25： 解释python的生成器

生成器是一种可迭代对象，可以挂起并保持当前的状态

生成器遇到yield处会停止执行，调用next()或send()才会继续执行

定义一个生成器有两种方式，一种是生成器推导式，一种是在普通函数中添加yield语句并实例化

Q26： Python中浅拷贝和深拷贝的区别

浅拷贝出来的是一个独立的对象，但它的子对象还是原对象中的子对象

深拷贝会递归地Q

Q27： 解释Python中的闭包

在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包。简单的说，如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包(closure)

闭包可以保存外部函数内的变量，不会随着外部函数调用完而销毁，例子：

代码块

```
def addx(x):
    def adder(y):
        return x + y
    return adder

c = addx(8)
print(type(c))
print(c.__name__)
print(c(10))
```

Q28： python dict数据结构的底层实现是什么？dict查找的时间复杂度？

dict的底层实现是哈希表；dict查找的时间复杂度为O(1)，可以实现按索引随机访问

Q29： `@staticmethod` 和 `@classmethod` 有什么不一样？

被staticmethod静态方法装饰器和classmethod类方法装饰器装饰的函数都可以用类名.方法名()来调用，区别是：

Q30： 可以用多线程加速Python程序吗？

不可以；Python 程序本身可以是多线程的，即使运行单线程程序，Python 也是多线程的——一个线程运行程序，另一个线程是垃圾收集器

python不支持多线程的原因是底层cpython存在全局锁

Q31： 解释什么是Python中的元类？为什么需要元类？

Python 中的元类是定义类行为方式的类的类。一个类本身就是元类的一个实例。Python 中的类定义了类实例的行为方式，即类是元类的实例

元类的作用过程：拦截类的创建；拦截下后，进行修改；修改完后，返回修改后的类。使用元类，可以对类进行定制修改；元类的作用过程是拦截类的创建

Q32： Python面向对象的继承有什么特点？

同时支持单继承与多继承，当只有一个父类时为单继承，当存在多个父类时为多继承

子类会继承父类所有的属性和方法，子类也可以覆盖父类同名的变量和方法

在继承中基类的构造（ init ()）方法不会被自动调用，它需要在其派生类的构造中专门调用

在调用基类的方法时，需要加上基类的类名前缀，且需要带上 self 参数变量。区别于在类中调用普通函数时并不需要带上 self 参数

高 级

Q33：Python中super函数的作用

super() 函数是用于调用父类(超类)的一个方法

代码块

```
class A():
    def funcA(self):
        print("this is func A")

class B(A):
    def funcA_in_B(self):
        super(B, self).funcA()

    def funcC(self):
        print("this is func C")

>>> ins = B()
>>> ins.funcA_in_B()
this is func A
>>> ins.funcC()
this is func C
```

Q34： 解释单例模式和工厂模式

单例模式：确保某一个类只有一个实例存在；这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。单例模式保证一个类仅有一个实例，并提供一个访问它的全局访问点，避免一个全局使用的类被频繁地创建与销毁

工厂模式：包含一个超类，这个超类提供一个抽象化的接口来创建一个特定类型的对象，而不是决定哪个对象可以被创建。工厂模式的主要解决的问题：将原来分布在各个地方的对象创建过程单独抽离出来，交给工厂类负责创建。其他地方想要使用对象直接找工厂（即调用工厂的方法）获取对象。工厂模式的优点是扩展性高、安全性强

Q35： 说说协程的优点

协程并没有增加线程数量，只是在线程的基础之上通过 分时复用 的方式运行多个协程，而且协程的切换在用户态完成，切换的代价比线程从用户态到内核态的代价小很多

协程最大优势就是协程极高的 执行效率 。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显

协程不需要多线程的锁机制 ，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多

Q36： 现要处理一个大小为10G的文件，但是内存只有4G，如果在只修改 get_lines 函数而其他代码保持不变的情况下，应该如何实现？需要考虑的问题都有那些?

readlines() 函数在文件过大时并不适用，应添加参数，限制读取的字节数，并使用生成器

代码块

```
def get_lines():
    l = []
    with open('file.txt','rb') as f:
        data = f.readlines(60000)
    l.append(data)
    yield l
```

或者使用mmap

代码块

```
from mmap import mmap

def get_lines(fp):
    with open(fp, "r+") as f:
        m = mmap(f.fileno(), 0)
```

Q37： 什么是Python中的猴子补丁?

猴子补丁是指在运行时动态修改类和模块。猴子补丁主要有以下几个用处：

Q38： python如何传递参数的?

Python 的参数传递是赋值传递 （pass by assignment），或者叫作对象的引用传递（pass by object reference）。Python 里所有的数据类型都是对象，所以参数传递时，让新变量与原变量指向相同的对象

根据对象的引用来传递，根据对象是可变对象还是不可变对象，得到两种不同的结果。如果是可变对象，则直接修改。如果是不可变对象，则生产新对象，让形参指向新对象

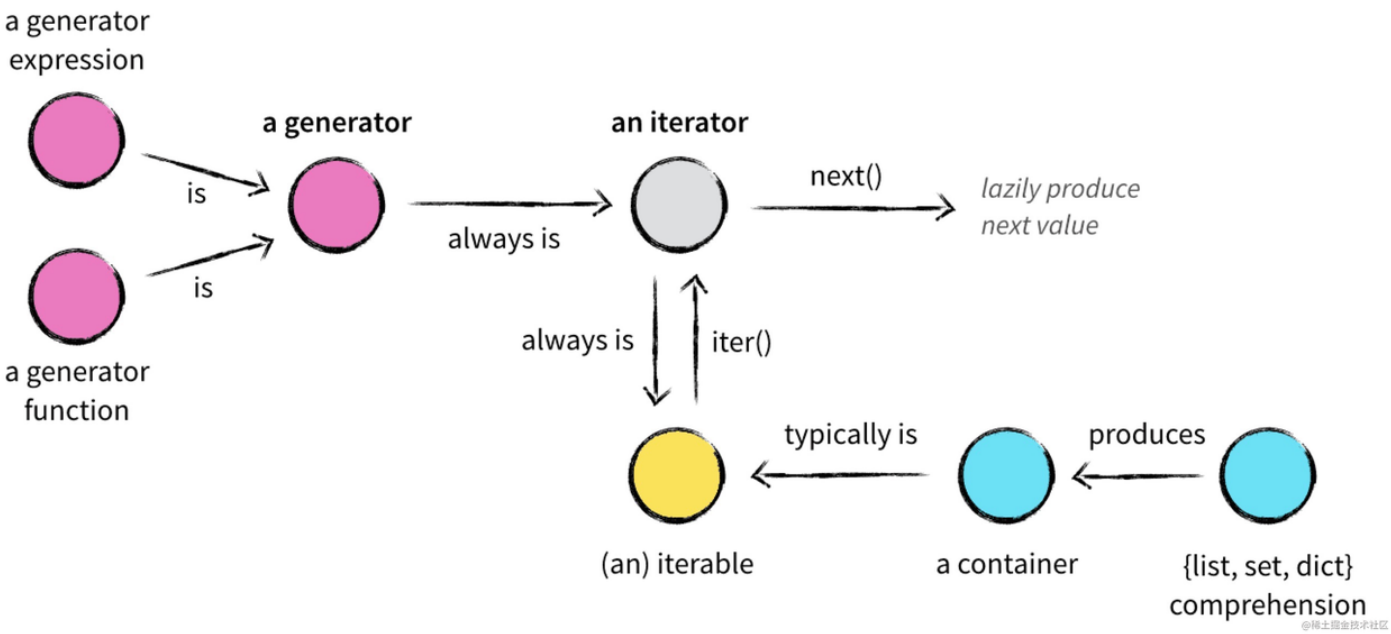
Q39： 了解Python中的GIL吗?

全局解释器锁 GIL，英文名称为 Global Interpreter Lock，它是解释器中一种线程同步的方式

对于每一个解释器进程都具有一个 GIL ，它的直接作用是限制单个解释器进程中多线程的并行执行，使得即使在多核处理器上对于单个解释器进程来说，在同一时刻运行的线程仅限一个

Python 代码被编译后的字节码会在解释器中执行，在执行过程中，存在于 CPython 解释器中的 GIL 会致使在同一时刻只有一个线程可以执行字节码。GIL 的存在引起的最直接的问题便是：在一个解释器进程中通过多线程的方式无法利用多核处理器来实现真正的并行。因此，Python的多线程是伪多线程，无法利用多核资源，同一个时刻只有一个线程在真正的运行

Q40： 什么是迭代器和生成器?



迭代器：python中的容器有许多，比如列表、元组、字典、集合等，对于容器，可以很直观地想象成多个元素在一起的单元，所有的容器都是可迭代的（iterable）。通常使用for in 语句对可迭代的对象进行枚举，其底层机制在于：可迭代对象，通过 iter() 函数返回一个迭代器（iterator），迭代器提供了一个 next 的方法。调用用这个方法后，你要么得到这个容器的下一个对象，要么得到一个 StopIteration 的错误。迭代器就像一个懒加载的工厂，等到有人需要的时候才给它生成值返回，没调用的时候就处于休眠状态等待下一次调用

生成器：生成器(generator)可以简单理解为懒人版本的迭代器。它相比于迭代器的优势是，生成器并不会像迭代器一样占用大量内存。比如声明一个迭代器：

代码块

```
1   L = [i for i in range(100000000)]
```

就可以声明一个包含一亿个元素的列表，每个元素在生成后都会保存到内存中。但实际上我们也许并不需要保存那么多东西，只希望在你用 next() 函数的时候，才会生成下一个变量，因此生成器应运而生，在python中的写法为

```
代码块
(i for i in range(100000000)
```

此外，生成器还可以有别的形式，比如生成器函数，通过yield关键字，把结果返回到next()方法中，举个例子：

```
代码块
def frange(start, stop, increment):
    x = start
    while x < stop:
        yield x
        x += increment

for n in frange(0, 2, 0.5):
    print(n)

0
0.5
```

Q41：解释Python中的垃圾回收机制

GC要做的有 2 件事，一是找到内存中无用的垃圾对象资源，二是清除找到的这些垃圾对象，释放内存给其他对象使用

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收效率

Q42： Python中__new__和__init__的区别?

new 是一个静态方法,而 init 是一个实例方法

new 方法会返回一个创建的实例,而 init 什么都不返回

只有在 new 返回一个cls的实例时后面的 init 才能被调用

当创建一个新实例时调用 new ,初始化一个实例时用 init