# 真题模拟

## 基础题

1. 如何让print()函数不换行

print('hello world', end='')

1. **name**=='**main**', \_\_name\_\_的值是什么

当执行文件本身时候\_\_name\_\_变量等于main，当调用该模块的时候\_\_name\_\_等于模块名

1. crontab的含义

分时日月周

例子：每年的4月份每周的周一到周三的11点执行脚本。

00 11 \* 4 1-3 /bin/sh /home/omc/h.sh

1. celery启动命令

celery -A tasks worker --loglevel=info

1. 简单正则的含义

https://www.cnblogs.com/guyuyun/p/5839881.html

：匹配单个字符 ：匹配非数字字符 ：匹配空白字符：空格，，，，，：匹配非空白字符 ：匹配单词字符：[a-zA-Z0-9\_] ：匹配非单次字符

1. 字典排序

list1= sorted(dict1.items(),key=lambda x:x[1])

1. docker的常用命令
2. 什么是csrf

CSRF跨站点请求伪造(Cross—Site Request Forgery)

1 用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A； 2 在用户信息通过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A； 3 用户未退出网站A之前，在同一浏览器中，打开一个TAB页访问网站B； 4 网站B接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A；

1. 堆和栈的区别
2. 为什么了有了GIL还要关注线程安全

首先明确：GIL并不是Python的特性，Python完全可以不依赖于GIL

GIL：保证了同一进程中的多线程在同一时刻只能有一个线程使用CPU执行指令。

Lock互斥锁：保证了某一段代码，在没有执行完毕之前无法被另一个线程执行。

参考资料： https://www.cnblogs.com/ryxiong-blog/p/10730085.html https://blog.csdn.net/qq\_37189082/article/details/95391482

1. IO密集，CPU密集
* 所谓IO密集型任务，是指磁盘IO、网络IO占主要的任务，计算量很小。比如请求网页、读写文件等。当然我们在Python中可以利用sleep达到IO密集型任务的目的。
* 所谓计算密集型任务，是指CPU计算占主要的任务，CPU一直处于满负荷状态。比如在一个很大的列表中查找元素（当然这不合理），复杂的加减乘除等。

https://blog.csdn.net/youanyyou/article/details/78990156 https://blog.csdn.net/cn\_wk/article/details/80216310

## 编程题

1. 手写单例模式

第一种方法:使用装饰器

def singleton(cls):
 instances = {}
 def wrapper(\*args, \*\*kwargs):
 if cls not in instances:
 instances[cls] = cls(\*args, \*\*kwargs)
 return instances[cls]
 return wrapper

@singleton
class Foo(object):
 pass
foo1 = Foo()
foo2 = Foo()
print(foo1 is foo2) # True

第二种方法：使用基类

New 是真正创建实例对象的方法，所以重写基类的new 方法，以此保证创建对象的时候只生成一个实例

class Singleton(object):
 def \_\_new\_\_(cls, \*args, \*\*kwargs):
 if not hasattr(cls, '\_instance'):
 cls.\_instance = super(Singleton, cls).\_\_new\_\_(cls, \*args, \*\*kwargs)
 return cls.\_instance

class Foo(Singleton):
 pass

foo1 = Foo()
foo2 = Foo()

print(foo1 is foo2) # True

第三种方法：元类，元类是用于创建类对象的类，类对象创建实例对象时一定要调用call方法，因此在调用call时候保证始终只创建一个实例即可，type是python的元类

class Singleton(type):
 def \_\_call\_\_(cls, \*args, \*\*kwargs):
 if not hasattr(cls, '\_instance'):
 cls.\_instance = super(Singleton, cls).\_\_call\_\_(\*args, \*\*kwargs)
 return cls.\_instance

# Python2
class Foo(object):
 \_\_metaclass\_\_ = Singleton

# Python3
class Foo(metaclass=Singleton):
 pass

foo1 = Foo()
foo2 = Foo()
print(foo1 is foo2) # True

1. 读大文件

def readInChunks(fileObj, chunkSize=4096):
 while 1:
 data = fileObj.read(chunkSize):
 if not data:
 break
 yield data

with open('bigFile') as f:
 for chunk in readInChunks(f):
 pass

1. ajax
2. 手写排序