1、请写出代码,用blocks来取代上例中的protocol,并比较两种方法的优势。实际应用部分？请写出代码，用blocks取代协议或回调方法  
声明：  
#import <Foundation/Foundation.h>

typedef void(^TestBlock)(NSString \*string);

@interface LO\_Person : NSObject

+ (void)showStringFromBlcok:(TestBlock)justBlock;  
@end  
实现：  
#import "LO\_Person.h"

@implementation LO\_Person

+ (void)showStringFromBlcok:(TestBlock)justBlock

{

NSString \*str = @"测试blcok";

justBlock(str);

}  
@end  
调用：  
[LO\_Person showStringFromBlcok:^(NSString \*string) {

NSLog(@"-- %@",string);

}];

1. 你做iphone开发时候，有哪些传值方式，view和Controller之间是如何传值的？  
   属性、delegate、blocks
2. 给定的一个字符串，判断字符串中是否还有png，有就删除它？  
   [string stringByReplacingOccurrencesOfString:@"png" withString: @""]
3. 对于语句NSString\* testObject = [[NSData alloc] init];testObject 在编译时和运行时分别是什么类型的对象？  
   编译的时候是NSString类型，运行的时候是NSData类型
4. OC中是所有对象间的交互是如何实现的？  
   函数指针实现
5. 目标-动作机制  
   目标是动作消息的接收者。动作是控件发送给目标的消息，或者从目标的角度看，它是目标为了响应动作而实现的方法。常用的UIButton添加方法
6. for(int index = 0; index < largenumber; index ++){  
   NSString \*tempStr = @”tempStr”;  
   NSLog(tempStr);  
   NSNumber \*tempNumber = [NSNumber numberWithInt:2];  
   NSLog(tempNumber);  
   }  
   这段代码有什么问题.？会不会造成内存泄露（多线程）？在内存紧张的设备上做大循环时自动释放池是写在循环内好还是循环外好？为什么？  
   参照第三个文档
7. 描述上拉加载、下拉刷新的实现机制？  
   根据下拉或者上拉的距离来判断是否进行网络请求
8. 什么是沙盒（sandbox）？沙盒包含哪些文件，描述每个文件的使用场景。如何获取这些文件的路径？如何获取应用程序包中文件的路径？  
   iOS应用程序只能在为该改程序创建的文件系统中读取文件，不可以去其它地方访问，此区域被成为沙盒，所以所有的非代码文件都要保存在此，例如图像，图标，声音，映像，属性列表，文本文件等。  
   默认情况下，每个沙盒含有3个文件夹：Documents, Library 和 tmp。  
   Documents：苹果建议将程序中建立的或在程序中浏览到的文件数据保存在该目录下，iTunes备份和恢复的时候会包括此目录 Library：存储程序的默认设置或其它状态信息；

Library/Caches：存放缓存文件，iTunes不会备份此目录，此目录下文件不会在应用退出删除

tmp：提供一个即时创建临时文件的地方。

iTunes在与iPhone同步时，备份所有的Documents和Library文件。iPhone在重启时，会丢弃所有的tmp文件。

1. 介绍一下XMPP？有什么优缺点吗？  
   XMPP：基于XML的点对点的即时通讯协议。  
   XMPP 协议是公开的，XMPP 协议具有良好的扩展性，安全性  
   缺点是只能传输文本
2. 谈谈对性能优化的看法，如何做？

从用户体验出发：1、程序logging不要太长、2、相同数据不做重复获取、3、昂贵资源要重用（cell、sqlite、date），4、良好的编程习惯和程序设计：选择正确的集合对象和算法来进行编程、选择适合的数据存储格式（plist、SQLite）、优化SQLite查询语句5、数据资源方面的优化（缓存和异步加载）

解决方案：

* 能够发现问题
* 利用log或工具分析问题原因
* 假设问题原因
* 改进代码和设计

http://blog.csdn.net/yangxt/article/details/8173412