1.在最近的某个项目或公司中，你是如何做的自动化测试工作?

"""情况一:测试团队已经有成型的自动化测试框架，且已经部分功能的测试用例在执行了。"""

1.明确框架：首先团队采用web自动化测试框架为:selenium+unittest+jenkins+git;
2.明确web自动化工作执行流程：
2.1 测试负责人划分功能模块web自动化测试任务到相关测试人员，'我'主要负责其中哪些模块，例如:商品管理、会员管理、订单模块部分功能。需要学员自行选择熟悉对应的功能模块业务，但是这里需要注意尽量不要挑选下单流程，因为既然团队有了的成型自动化框架并且有了部分测试用例在执行了，则其实现的部分肯定是下单流程的。
2.2 测试计划制定:根据测试负责人划分的功能模块制定相关测试计划，确认工作完成时间，并交给测试负责人进行审核;
2.3 环境搭建，初始化项目：那么这里只需要搭建好本地的selenium运行环境即可。不需要自己初始化项目，直接从对应git上面获取到最新的自动化代码，并确保获取的最新代码是能正常运行的。
2.4 编写测试用例：按拟定的测试计划设计文案形式的web自动化测试用例，确定测试场景。那么这里需要简单说明下主要要正向的测试用例，如商品管理模块:成功新增商品;
2.5 编写测试脚本：那么在这里说明团队统一采用的是的市场上比较流行的一套测试用例编写架构，PO模式(页面对象)。会把页面分层三层：对象库层、操作层、业务层，在编写测试脚本时，'我'会先封装好页面PO文件，例如新增商品:
会新增一个对应PO文件，在对象库层里面会把新增商品页面中需要操作的元素都定位回来(如品名输入框、价格输入框、类目选择框等等)，并将定位方式和值抽取到初始化方法中进行管理；在操作层会封装所有的元素对象的操作方法，从对象库层拿到对应的元素对象；最后再最后一层业务层:会组织多个元素对象的操作，连贯起来就是一个业务动作，那么对于新增来讲，会在业务层中封装一个新增商品的方法：里面通过调用操作层例如：品名输入方法、价格输入的方法等，连贯起来就形成了业务操作。业务层提供的业务方法最终是提供给测试用例去进行调用，业务所需要的数据时进行过参数处理的；至此，对于页面封装好了。'我'再来编写测试用例，测试用例会专门放在一个/case的目录下，使用unittest去组织用例，会通过调用业务层中已经写好的业务方法实现对应的业务操作，通过parameterized参数化不同数据来实现对同个业务的不同场景实现测试，另外还会将测试数据定义到json文件中。当脚本编写完成后会单个进行调试，确认脚本没有问题后提交到git上。
2.6 统一运行，生成测试报告：通过jenkins统一运行所有测试用例。一般情况下，每次测试版本发布时都会统一运行一次的全部自动化测试用例；运行完后会生成测试报告，已邮件的形式发送给测试组员，测试人员会根据自己负责的模块去查看对应的类容，确认是否正常；

"""情况二:进入团队时，还没有web自动化相关的实施工作。"""

1.先明确背景：刚进入团队时项目还处于初期紧张的阶段，团队也小，测试人员基于基本的功能测试工作已经很饱和了。那么web自动化的工作不是'我'一进入团队时就开始做了的。
2.引入自动化原因剖析：经过和团队的一起努力后，'我们'的平台的第一个成型版本总算成功上线，包含了核心业务:例如电商平台，已经有整体商品体系、会员体系、订单体系等核心业务，基本满足业务需求。但是随着业务发展，结合运营、业务要求，平台在后期还在不断小的更新、以及新的独立功能模块陆续增加(例如业务行为分析统计、抢购、团购等)。在这个期间团队每周~半个月都会有一个的版本来上线优化或者小功能。每次版本都需要测试组员对原核心业务：商品体系、会员体系、订单体系进行一个简单回归测试。这里强调下有两个问题：一个回归不全面、二个几乎每次回归测试占了整体测试过程了20%，投入大量的人力资源。
3.引入web自动化：此时，需要体现自己的价值。'我'发现我们回归的功能部分在版本迭代过程中页面功能交互基本没有什么变动，处于比较稳定的状态。通过同行了解到使用web自动化测试可以解决回归测试的问题。'我'开始尝试学习基于python语言来编写selenium自动化脚本，为了更好的组织测试用例还引入了unittest自动化测试框架。我利用自己的时间使用unittest编写测试用例实现了基于'我'自己负责的功能模块，后面每次回归测试都节省了'我'大量时间，只需要查看下对应的测试报告即可。
4.经过实践后觉得对于我们现在的测试团队还是有很大价值的，就组织小组一起共同学习，还有开发介入协助，后续引入了PO模式和持续集成。PO模式(页面对象)更好的管理我们的测试用例。

然后介绍一下PO模式：把页面分层三层：对象库层、操作层、业务层，在编写测试脚本时，'我'会先封装好页面PO文件，例如新增商品：会新增一个对应PO文件，在对象库层里面会把新增商品页面中需要操作的元素都定位回来(如品名输入框、价格输入框、类目选择框等等)，并将定位方式和值抽取到初始化方法中进行管理；在操作层会封装所有的元素对象的操作方法，从对象库层拿到对应的元素对象；最后再最后一层业务层：会组织多个元素对象的操作，连贯起来就是一个业务动作，那么对于新增来讲，会在业务层中封装一个新增商品的方法：里面通过调用操作层例如：品名输入方法、价格输入的方法等，连贯起来就形成了业务操作。业务层提供的业务方法最终是提供给测试用例去进行调用，业务所需要的数据时进行过参数处理的；至此，对于页面封装好了。'我'再来编写测试用例，测试用例会专门放在一个/case的目录下，使用unittest去组织用例，会通过调用业务层中已经写好的业务方法实现对应的业务操作，通过parameterized参数化不同数据来实现对同个业务的不同场景实现测试，另外还会将测试数据定义到json文件中。当脚本编写完成后会单个进行调试，确认脚本没有问题后提交到git上。通过jenkins统一运行所有测试用例。
5.最后总结:后续一段时间内我们的回归测试工作缩短到了测试周期的5%以内。

2.请简单描述一下webdriver的原理

可以更通俗的理解：由于客户端脚本(java, python, ruby)不能直接与浏览器通信，这时候可以把WebService当做一个翻译器，它可以把客户端代码翻译成浏览器可以识别的代码(比如js)

1.客户端(也就是测试脚本)创建1个session，在该session中通过http请求向WebService发送restful的请求

2.WebService翻译成浏览器懂得脚本传给浏览器

3.浏览器把执行的结果返回给WebService,WebService把返回的结果做了一些封装(一般都是json格式)然后返回给client

4.根据返回值就能判断对浏览器的操作是不是执行成功

3.webdriver在服务器上面没有界面你如何进行处理?

```
"""设置无头浏览器"""
from selenium import webdriver

# 创建出启动浏览器所需要配置 -- 实例化ChromeOptions浏览器选项对象
co = webdriver.ChromeOptions()
# 构建配置信息 -- 通过浏览器选项对象调用配置方法
co.headless = True  # 设置浏览器为无头模式

# 将配置信息加入到浏览器启动 -- 实例化浏览器驱动对象添加属性option值
driver = webdriver.Chrome(options=co)
# 设置浏览器页面大小为常用窗口大小1366*768
driver.set_window_size(1366, 768)

# 打开页面, 进行页面操作
driver.get("http://www.baidu.com")
```

```
# 业务操作
driver.find_element_by_id("kw").send_keys("chrome")
driver.get_screenshot_as_file("./a.png")

# 退出浏览器驱动对象
driver.quit()
```

4.web自动化中影响页面定位的场景有哪些?

一般来说,是使用Selenium进行页面操作时,定位失败的场景可以通过以下步骤进行判断:
1.首先判断定位所以使用的元素属性信息是否正确,可以通过浏览器工具验证,确保定位信息正确性
2.其次经常受到网络/服务器/浏览器影响,造成的页面加载过慢造成的定位失败,可以通过元素等待结局(硬等待/显式等待/隐式等待)
3.判断页面元素是否直接显示,部分元素需要执行不同的鼠标动作才会显示出来
4.注意页面元素的加载情况,有时候可能由于默认打开浏览器窗口过小,造成元素遮挡,不加载,可以默认在启动浏览器后设置窗口最大化
5.收到前端技术影响,页面内容分页内容动态加载的,这个时候需要手工操作滚动条扩大加载范围的时候,可以通过执行js语句实现
6.影响页面定位的东西还有窗口弹窗(alert),这个东西会影响页面元素的定位操作,所以要先针对处理,在进行后续处理
7.页面中的超链接点击打开是可能实在新窗口中进行的,需要定位的元素如果在新窗口中需要进行窗口切换操作
8.收受前端页面框架影响,页面内容可能在frame子页中(表单提交/后台管理/页面广告),如果操作的内容在frame子页中,需要进行切换

5、如何提高自动化测试的执行速度?

Selenium脚本的执行速度受多方面因素的影响，如网速，操作步骤的繁琐程度，页面加载的速度，以及我们在脚本中设置的等待时间，运行脚本的线程数等。所以不能单方面追求运行速度的，要确保稳定性，能稳定地实现回归测试才是关键。

我们可以从以下几个方面来提高速度：
一，减少操作步骤，如经过三四步才能打开我们要测试的页面的话，我们就可以直接通过网址来打开，减少不必要的操作。
二，中断页面加载，如果页面加载的内容过多，我们可以查看一下加载慢的原因，如果加载的内容不影响我们测试，就设置超时时间，中断页面加载。
三，在设置等待时间的时候，可以sleep固定的时间，也可以检测某个元素出现后中断等待(只能等待)也可以提高速度。
四，使用Grid分布式执行测试用例。在编写测试用例的时候，一定要实现松耦合，然后在服务器允许的情况下，尽量设置多线程运行，提高执行速度。

6、你写完一个功能的脚本所花费的时间我都可以做3个功能的手工测试了？你觉得web自动化的意义在哪?

首先在做自动化测试之前我会先去确认项目是否适合做自动化。如果是不稳定的项目或者短期的项目，那肯定是不太适合做web自动化的，使用手工测试更为适合。
那么反之适合做自动化的项目，web自动化的收益与迭代次数成正比，也就是说在长期的迭代过程中，使用自动化自动化来代替我们回归测试，相对收益会越来越大。而且在这期间通过自动化，可以将手工重复度较高的、人为容易犯错的、较为机械化的工作事项，都给自动化了，将执行这类工作的人力释放出来，去做有创造性的、复杂度高的、有成长性的工作。这就是我认为的自动化的意义。

7、selenium自动化页面元素找不到存在异常的原因?

1、元素定位错误
2、页面加载时间过慢，需要查找的元素程序已经完成，单页面还未加载，此时可以加载页面等待时间
3、有可能元素包含在iframe或者frame里面，需要切换。

8、selenium为什么不推荐使用xpath定位?

selenium使用xpath定位时采用遍历页面的方式，性能指标较差。另外xpath定位有通过绝对路径定位的，有时会不准确；而用css选择器定位比较简洁，运行速度更快，通常用于性能要求严格的场景。

9、如何提高脚本的稳定性?

首先只要页面一直没变过，说明定位方法是没问题的。
优化方向：① 自己写相对路径，多用id为节点查找，少用右键复制xpath，那种不稳定。
② 第二个影响因素就是等待了，sleep等待尽量少用(影响执行时间)
③ 定位元素方法重新封装，结合WebDriverWait和excepted_conditions判断元素方法，自己封装一套定位元素方法

10、一个元素明明定位了，点击无效(也没报错)，如何解决?

使用js点击，selenium有时候点击元素时会失效

```
js = 'document.getElementById('baidu').click()'
driver.execute_script(js)
```