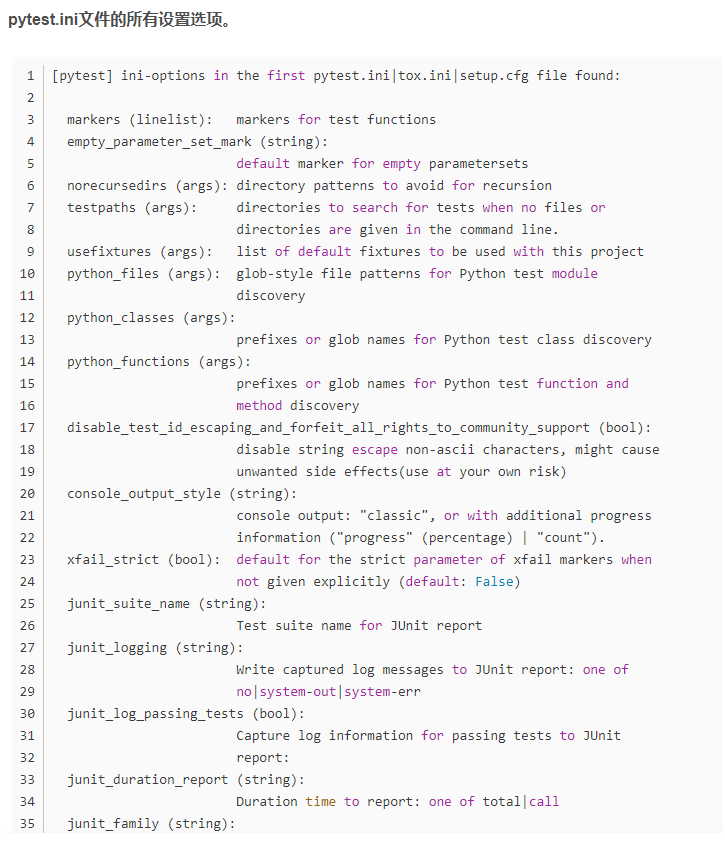
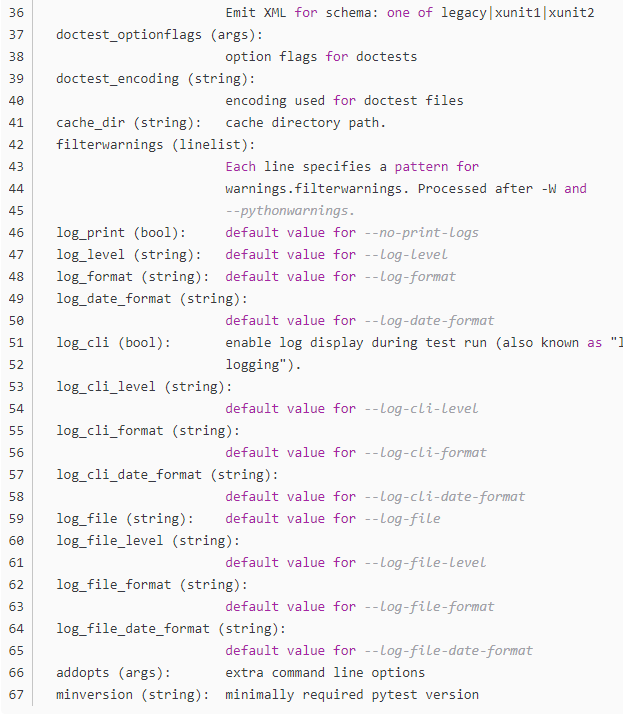
9/6/21\*

3/9/22, 5/8     <https://blog.csdn.net/python_jeff/article/details/121848584?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0.pc_relevant_default&spm=1001.2101.3001.4242.1&utm_relevant_index=3>

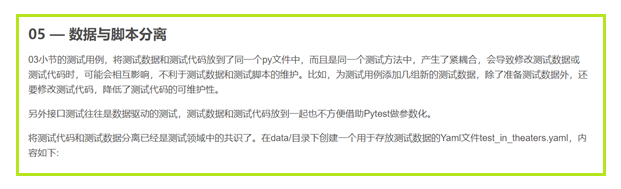
  



**Test case (data file driven,),**

**test results (compare with expected)**

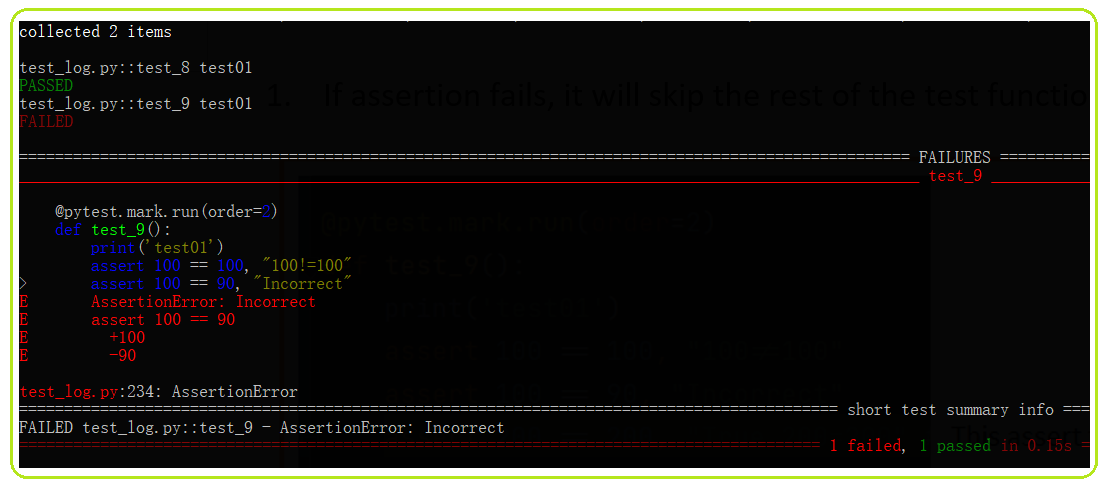
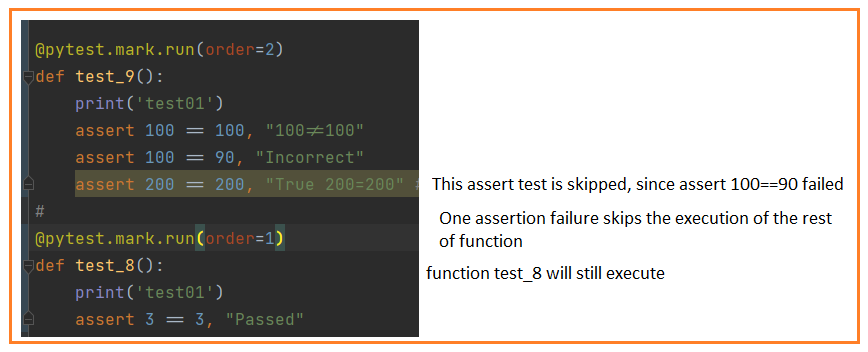
**and test environment**



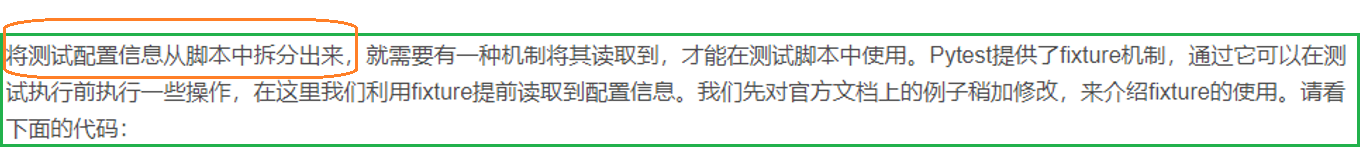
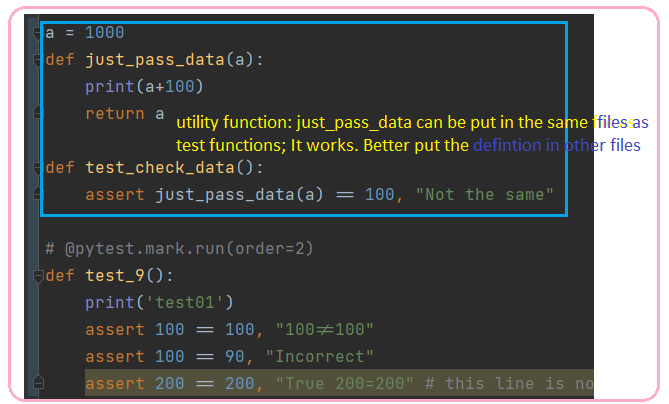
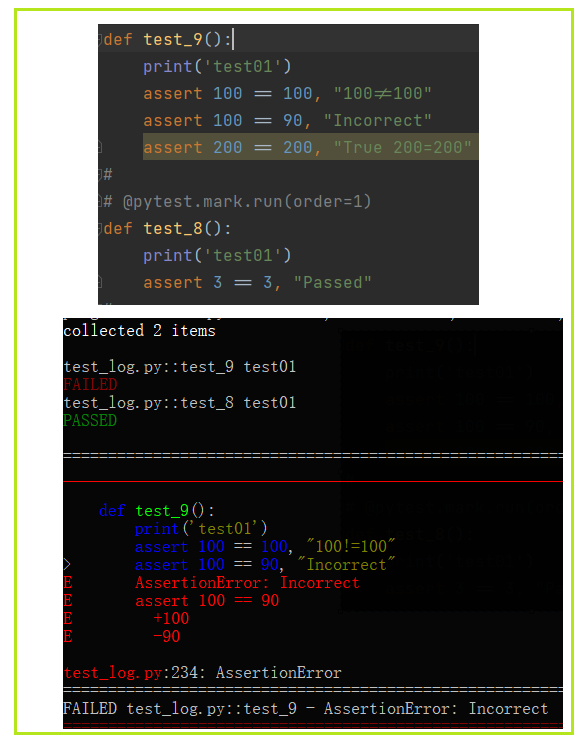
Not quite sure how yield works in setup and teardown

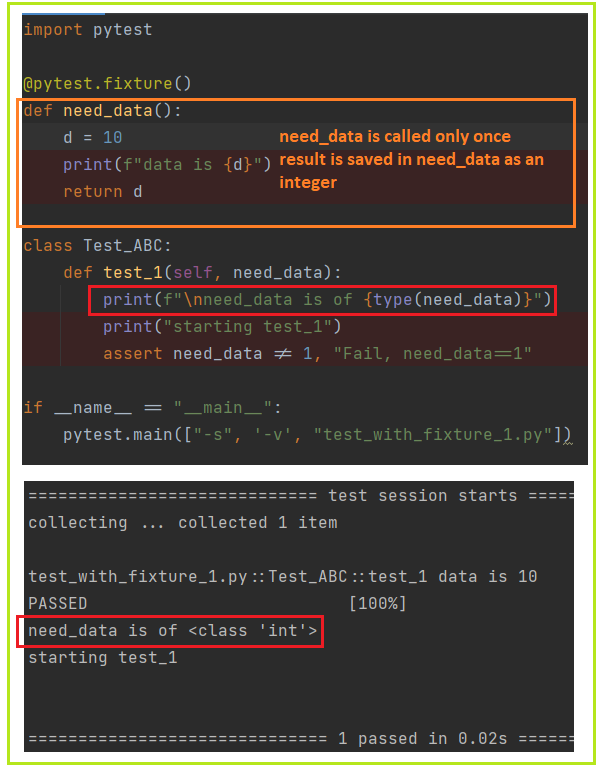
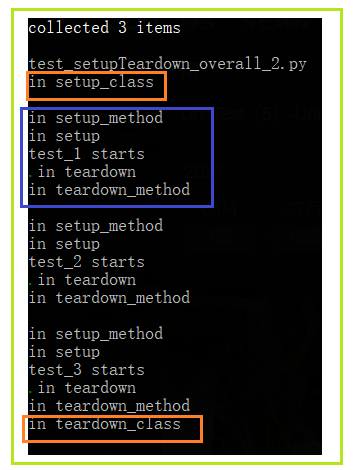
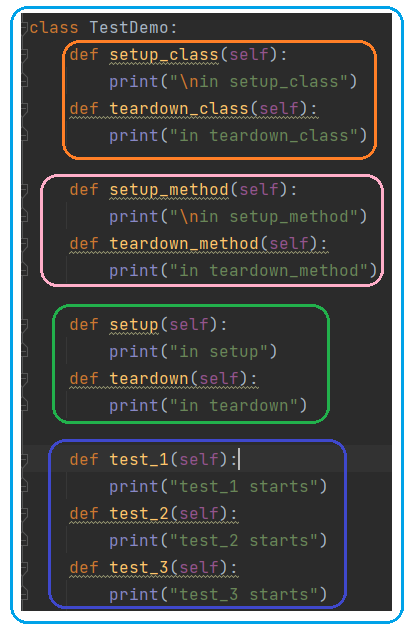
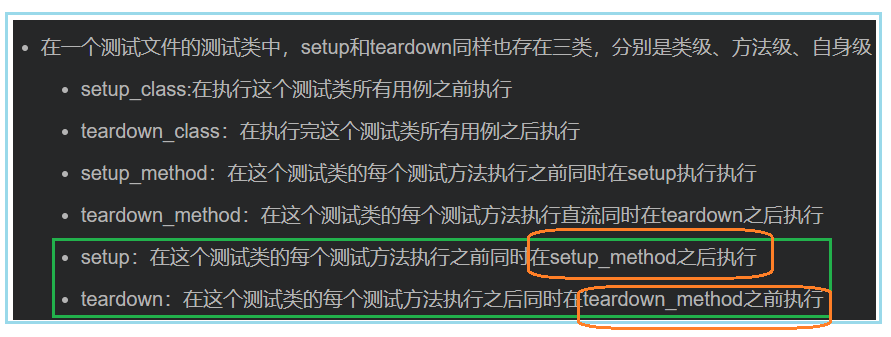
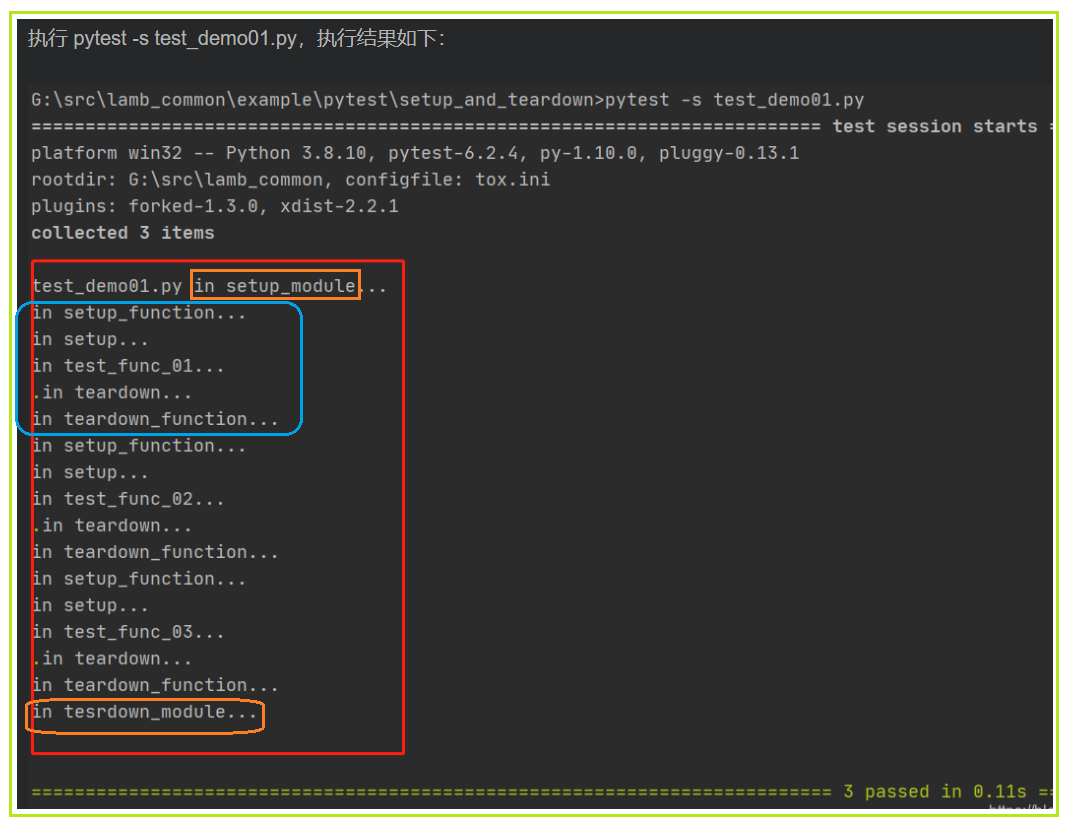
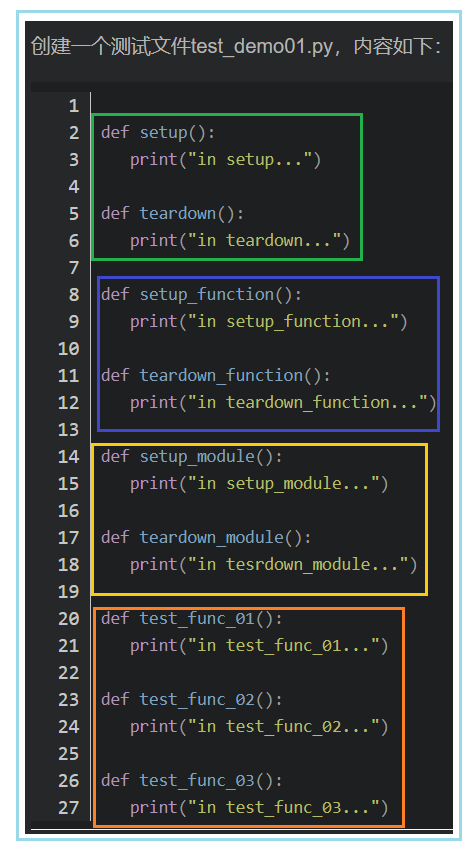
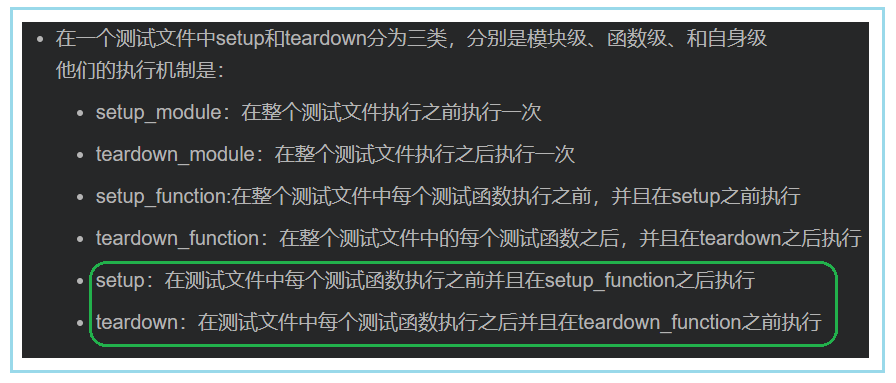
* How to use pytest.ini
* **Test data case** separate from test script
* **Test setup** separate from test script

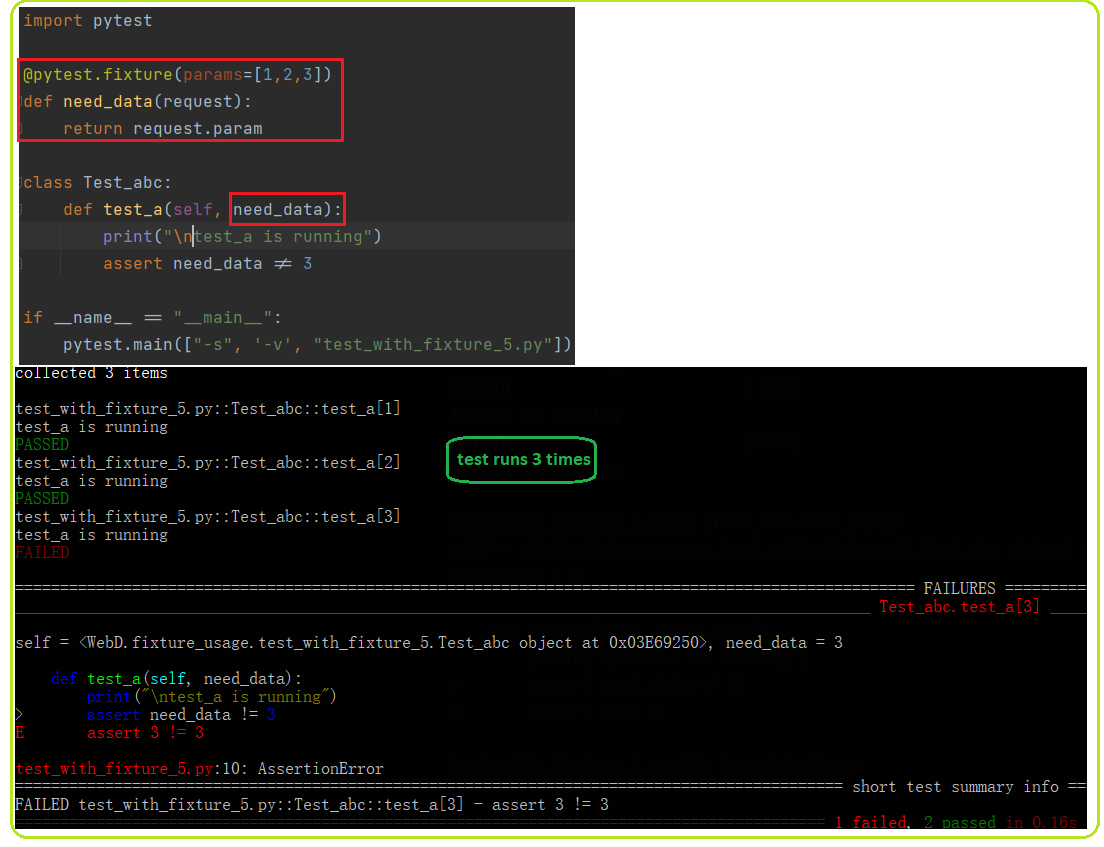
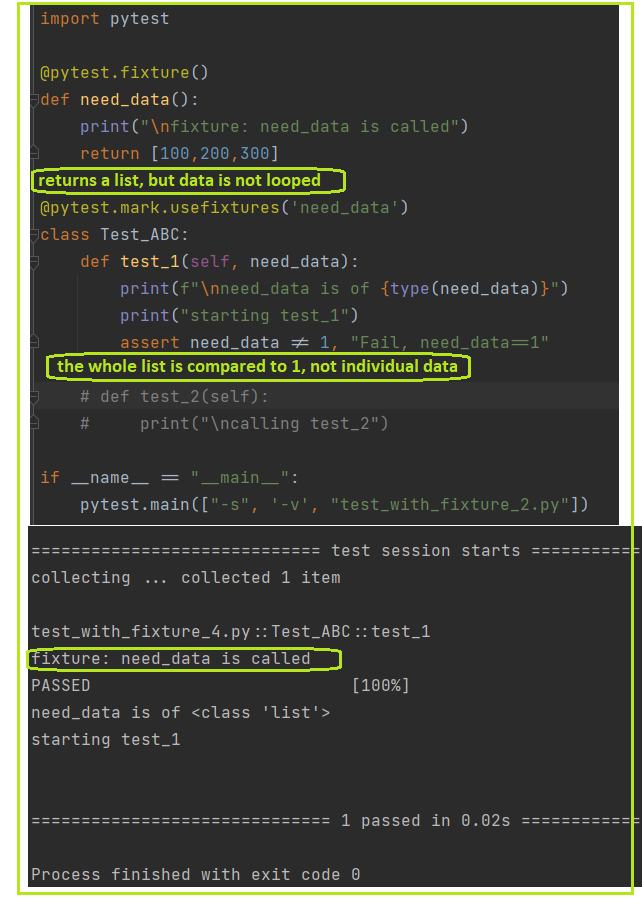
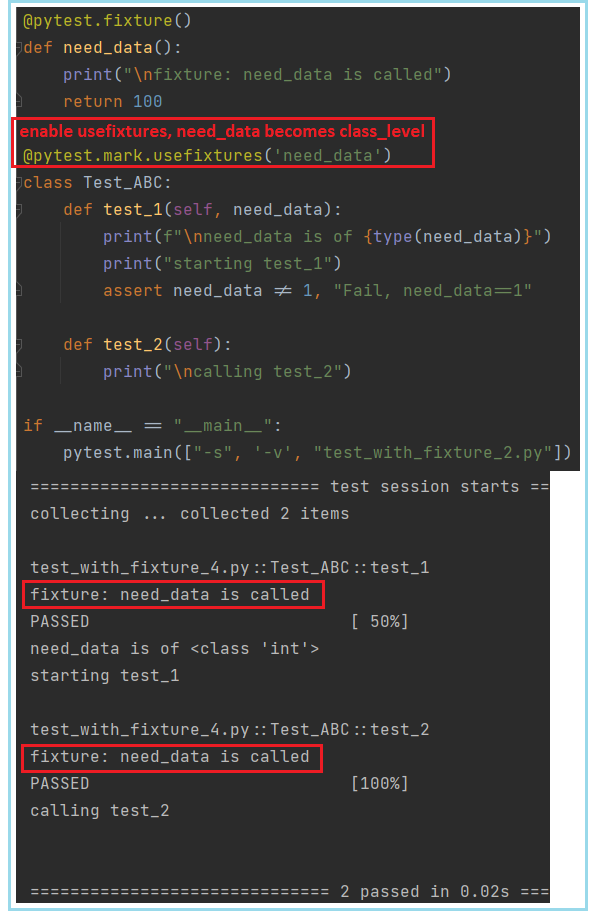
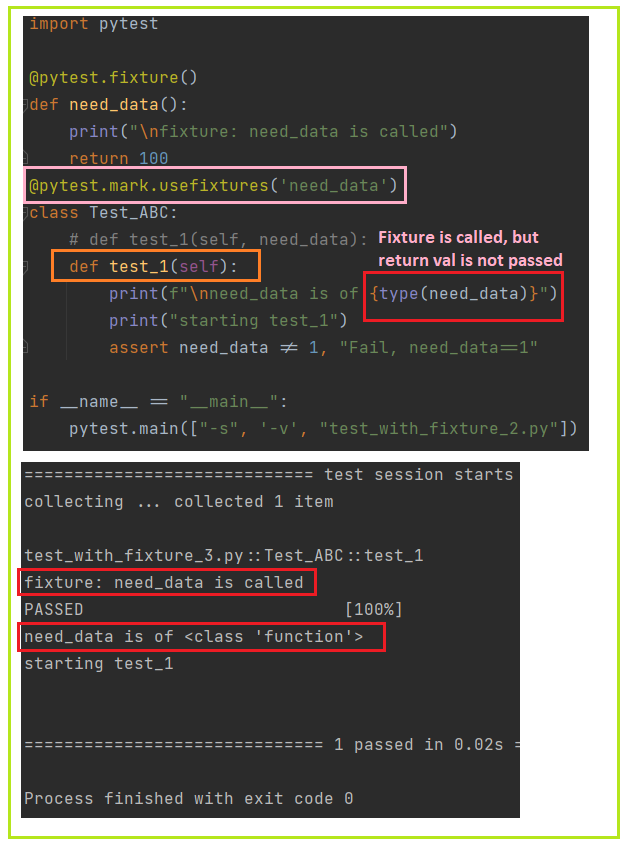
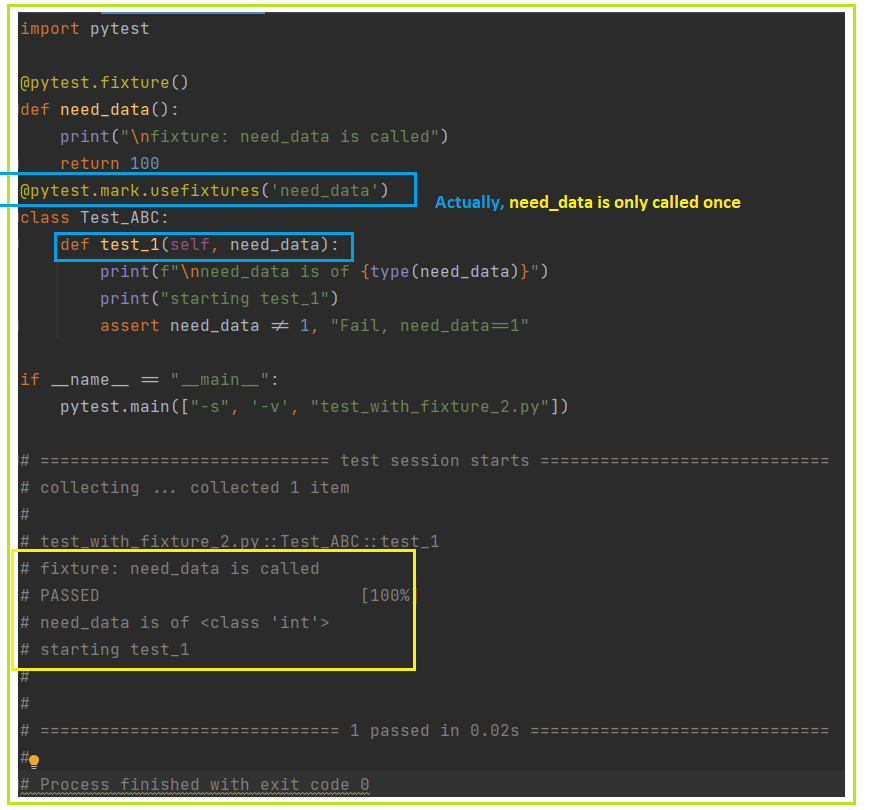
1. If assertion fails, it will skip the rest of the test function

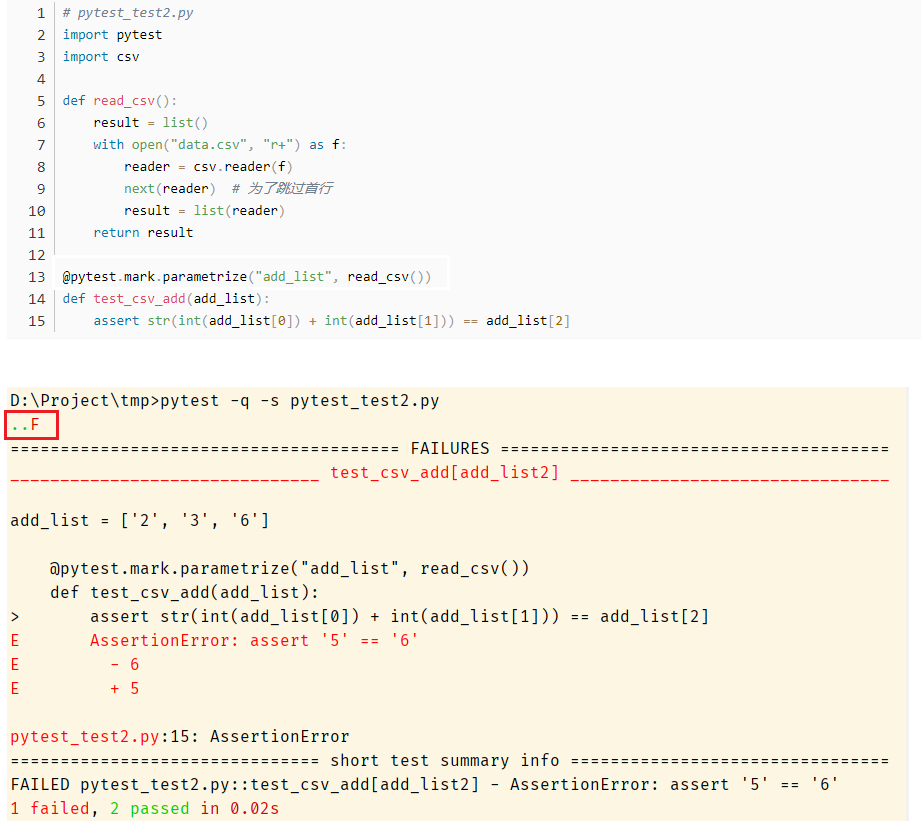
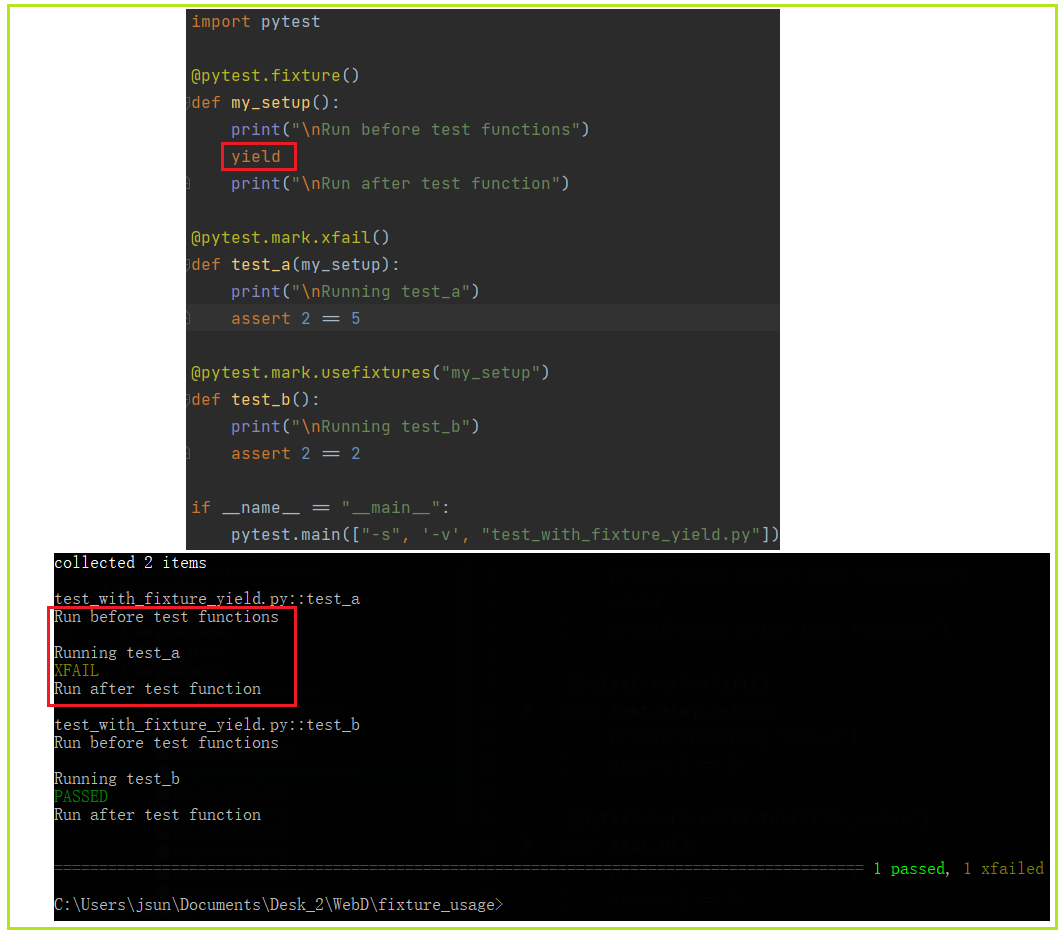


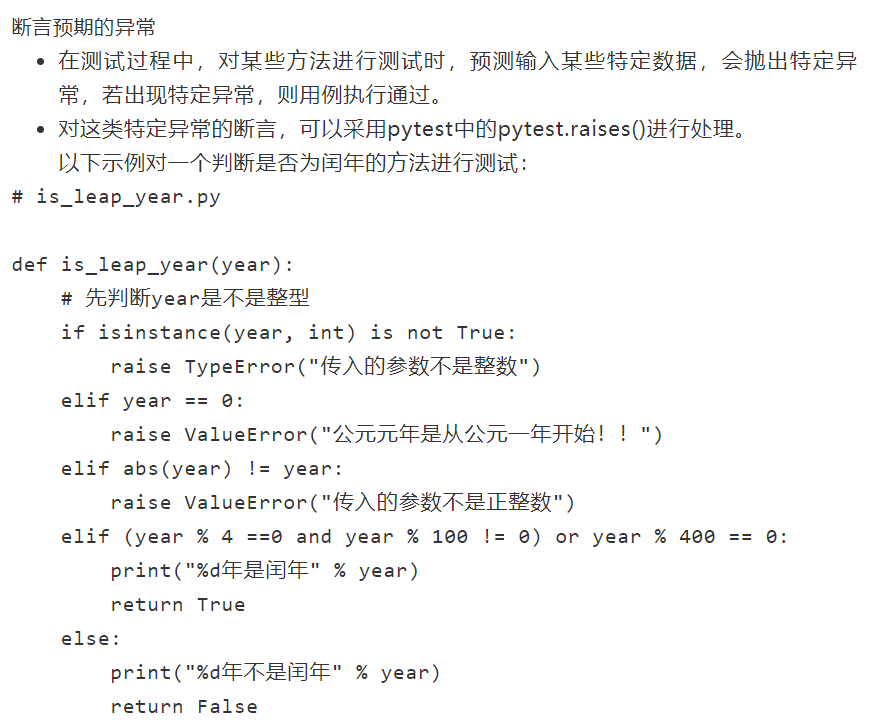
In the same test file, pytest executes **in order of functions located in the file**; .









**从形态上说，接口测试可以分为单接口的测试和基于业务的接口测试**，

**单接口**的很好理解，就是每个接口测试之间没有任何的依赖关系，

* 客户端请求服务端，
* 服务端返回响应数据
* 然后断言，

我个人觉得它的测试是有必要的，但是意义不大，毕竟从产品形态上来说，怎么可能没关联了，事物之间充满了太多的依赖性。

**基于业务场景的API的测试**，那么使用最广泛的。也是通过接口测试的技术实现业务场景的测试，

* 这样它的难点在于**业务之间的关联性，以及接口和接口之间的依赖关系**，处理最复杂也是最普遍的是参数的上下关联性，如A接口返回的参数，给下个接口B调用，使用的方式当然也是很多的，如postman里面使用定义变量方式解决，JMeter里面可以使用后置处理器来解决，**代码层面可以使用函数的返回值来解决**，其实思路都是一样的，只不过一个是工具，另外一个是代码。当然针对单纯的一个接口来测试，就存在几个维度，如一个接口中，它的请求参数是否为空验证，类型验证，边界值验证和安全性的验证。

