一、练习题

选择题

1.[多选]：对于封装说法正确的是：

- [x] A.封装是将一些有共性的或多次被使用的代码提取到一个方法中，供其他地方调用；
- [x] B.封装可以降低代码冗余；
- [x] C.封装可以增强安全性和简化编程，使用者不需要了解具体的实现细节，只需要使用封装的代码提供的接口，来使用类的成员方法或属性；
- [ ] D.对于一个项目而言封装的越多代码越优秀；

2.[多选]：下面对于获取浏览器工具类说法错误的是：

```
from selenium import webdriver
class DriverUtil:
    _driver = None
    @classmethod
    def get_driver(cls):
        if cls._driver is None:
            cls._driver = webdriver.Chrome()
            cls._driver.maximize_window()
            cls._driver.implicitly_wait(10)
            cls._driver.get("http://localhost")
        return cls._driver
```

- [ ] A.使用if判断_driver是否为空是为在后续多个测试用例或测试文件需要获取浏览器驱动对象时，保障所整个测试执行过程所使用的driver是唯一的；
- [ ] B.return返回已经初始化的浏览器驱动对象是因为在测试用例中需要使用到初始化好的浏览器驱动对象；
- [ ] C.封装获取浏览器驱动的原因是因为在web自动化脚本中会多次使用到浏览器驱动对象；
- [x] D.封装为类级别方法是为了方便调用且必须使用类级别的方法进行封装；

3.[多选]：下面对于面向对象说法错误的是：

- [ ] A.面向对象包含三大特性:继承、多态、封装，万事万物都是对象，它本身是一种思维模式；
- [ ] B.面向对象和面向过程都是解决问题思路，面向过程的注重是解决问题的步骤，面向对象注重的是对象的特征和行为。
- [x] C.面向对象的编码方式一定比面向过程的编码方式优越；
- [ ] D.面向对象的编码方法可扩展性高，编码复杂度高。将类转换为对象的过程叫实例化；

4.[多选]：对于下面的代码说法正确的是：

```
from selenium import webdriver
class DriverUtil:
    def __init__(self):
        self.driver = None
```

```
        self.name = "name"
        print(self._name)

    def get_driver(self):
        if self.driver is None:
            self.driver = webdriver.Chrome()
            self.driver.maximize_window()
            self.driver.implicitly_wait(10)
            self.driver.get("http://localhost")
        return self._driver
driver_util = DriverUtil()
driver_util.get_driver()
```

- [x] A.driver_util是DriverUtil类的一个实例；
- [x] B.删除上面代码的最后一行再执行代码，会打印字符串"name"
- [x] C.self.driver和self.name都是DriverUtil类的类属性，在成员方法中通过对象(self,cls)来调用类属性；
- [ ] D.最后两行修改为DriverUtil.get_driver()也能打开浏览器；

5.[多选]：对于PO模式理解正确的是：

- [x] A.PO模式是基于面向对象的思想，将操作每个页面当成对象，web自动化使用PO模式来编写脚本是为了减少自动化脚本对元素定位的依赖，提高测试用例的可读性、可维护性；
- [x] B.PO模式是对面向对象多态、继承、封装特性的应用，是自动化测试项目的最佳实践之一；
- [ ] C.只要做web自动化测试就必须得引入PO模式；
- [x] D.使用PO模式的方式来进行web自动化测试设计所有元素定位信息在整个测试项目中都只会存在一份；

6.[多选]：对于PO模式分层思想说法正确的是：

- [x] A.PO模式将每个页面当成对象分成三层进行管理:对象库层、操作层、业务层，各层分别负责不同的、唯一的工作，通过这种形式来降低代码之间的耦合性；
- [x] B.PO对象库层专门负责管理元素定位方式和信息以及获取到测试过程中所要操作的元素对象；
- [x] C.PO操作层专门负责管理所有元素对象操作方法，而元素对象可直接从对象库中获取；
- [x] D.PO业务层专门组织多个元素对象的操作，独立的操作连贯起来后就是一条业务操作流程，形成业务动作，对应测试用例需要执行某测试用例的时候只需调用对应的业务层组织好的业务动作方法即可；

7.[多选]：找出下面某对象库封装代码错误说明正确的选项：

```
class HomePage:
    def __int__(self):
        self.driver = DriverUtil.get_driver()#该出为获取浏览器驱动对象
        self.input_box_1 = (By.ID,"box_1")
        self.input_box_1 = (By.ID,"box_2")
        self.click_btn = (By.ID,"btn")
    def find_input_box_1(self):
        self.driver.find_element(*self.input_box_1)
    def find_input_box_2(self):
        self.driver.find_element(*self.input_box_1)
        def find_click_btn(self):
            self.driver.find_elements(*self.click_btn)
```

- [x] A.对象库需要找回元素对象供操作层使用，上面查找元素对象的方法都没有将元素对象返回；
- [x] B.初始化类成员变量同名；
- [x] C.初始化方法写错成了int；
- [x] D.find_click_btn元素方法对齐方式错误；

8.[多选]：找出下面某操作层封装代码错误说明正确的选项：

```
class HomeHandler:
    def __init__(self):
        self.home_page = HomePage

    def input_box_1(self,text):
        self.home_page.find_input_box_1().send_keys(text)

    def input_box_2(self,text):
        self.home_page.input_box_2.send_keys(text)

    def click_btn(self):
        self.driver.find_click_btn().click()
```

- [x] A.实例化对象库层时HomePage遗漏括号；
- [x] B.input_box_2方法中实现输入操作，调用对象库层次获取元素对象方法错误调用成了对象库层属性；
- [ ] C.input_box_1和input_box_2参数一样；
- [ ] D.3个操作方法都没有return返回；

9.[多选]：找出下面某业务层封装代码错误说明正确的选项：

```
class HomeProxy:
     def __init__(self):
        self.home_handler = HomePage()
     def test_demo()
        self.home_handler.input_box_1()
        self.home_handler.input_box_1()
        self.home_handler.click_btn
```

- [x] A.实例化操作层错误实例化成了对象库层；
- [x] B.调用操作层方法input_box_1、input_box_1参数遗漏，test_demo也需要从外部接受参数传递；
- [x] C.click_btn方法丢失括号；
- [x] D.test_demo后遗漏":"；

二、提高题

代码题

1.使用PO模式封装TPSHOP后台管理系统登录测试用例；

```
# ~/page/login_page.py
# 该py文件表示的是:登陆页面
from selenium.webdriver.common.by import By
```

```python
from utils import DriverUtils

# 对象库层:专门找到元素对象，并且管理元素对象的定位
class LoginPage:
    # 初始化方法用来管理属性,将元素定位方式以及值提取到初始化方法中的属性进行管理
    def __init__(self):
        # 浏览器驱动对象
        self.driver = DriverUtils.get_driver()
        # 用户名输入框
        self.usename = (By.NAME, 'username')
        # 密码输入框
        self.password = (By.NAME, 'password')
        # 验证码输入框
        self.verify_code = (By.NAME, 'vertify')
        # 登陆按钮
        self.submit_btn = (By.NAME, 'submit')

    # 找到用户名输入框元素对象
    def find_username(self):
        return self.driver.find_element(*self.usename)

    # 找密码输入框元素对象
    def find_password(self):
        return self.driver.find_element(*self.password)

    # 找验证码输入框元素对象
    def find_verify_code(self):
        return self.driver.find_element(*self.verify_code)

    # 找登陆按钮元素对象
    def find_submit_btn(self):
        return self.driver.find_element(*self.submit_btn)


# 操作层:专门管理元素对象的操作事件
class LoginHandler:

    def __init__(self):
        # 实例化对象库层
        self.login_page = LoginPage()

    # 用户名输入
    def input_username(self, username):
        # 清除文本信息
        self.login_page.find_username().clear()
        self.login_page.find_username().send_keys(username)

    # 密码输入
    def input_password(self, pwd):
        self.login_page.find_password().clear()
        self.login_page.find_password().send_keys(pwd)

    # 验证码输入
    def input_code(self, code):
        self.login_page.find_verify_code().clear()
        self.login_page.find_verify_code().send_keys(code)

    # 登陆按钮点
```

```
    def click_submit(self):
        self.login_page.find_submit_btn().click()


# 业务层:组装多个元素操作形成业务流，业务流就是测试用例一个连续功能操作流程
class LoginProxy:

    def __init__(self):
        # 实例化操作层对象
        self.login_handler = LoginHandler()

    # 提供一个登陆方法功登陆测试用例使用
    def test_login(self, username, pwd, code):
        # 输入用户名
        self.login_handler.input_username(username)
        # 输入密码
        self.login_handler.input_password(pwd)
        # 输入验证码
        self.login_handler.input_code(code)
        # 点击登陆按钮
        self.login_handler.click_submit()
```

```
# utils.py工具类文件
# 获取浏览器驱动对象的工具类
import time
from selenium import webdriver


class DriverUtils:
    # 定义私有变量，用来存储浏览器驱动对象
    __driver = None
    # 获取浏览器驱动对象并且初始化
    # 1.为了方便调用，设置类级别的方法
    # 2.为了保障get_driver在多次调用的时候浏览器驱动对象的唯一性，需要添加判断
    @classmethod
    def get_driver(cls):
        # 如果浏览驱动对象的私有变量__driver=None
        if cls.__driver is None:
            # 实例化浏览器驱动
            cls.__driver = webdriver.Chrome()
            # 窗口最大化
            cls.__driver.maximize_window()
            # 隐式等待
            cls.__driver.implicitly_wait(10)
        # 返回浏览器驱动对象
        return cls.__driver  # driver-001   # 缓存信息

    # 关闭浏览器驱动对象
    @classmethod
    def quit_driver(cls):
        # 通过获取浏览器驱动对象的方法拿到浏览器驱动对象
        # 为了保障代码的健壮性，添加__driver不为空的判断
        if cls.__driver is not None:
            cls.get_driver().quit()
            # 浏览器驱动对象cls.__driver 在调用quit()之后只是关闭只管所看到的效果，实际里
面还有一些缓存信息
            cls.__driver = None
```

```python
# ~/case/test_login.py
import unittest
from parameterized import parameterized
from page.login_page import LoginProxy
from utils import DriverUtils

# 定义测试类
class TestLogin(unittest.TestCase):

    # 类级别的初始化fixture:用来在第一个测试方法前打开浏览器，最大化隐式等待
    @classmethod
    def setUpClass(cls):
        # 获取浏览器驱动对象后为什么要用cls.driver类属性来承接
        # 原因：因为在测试方法中还需要用到大量浏览器驱动对象所提供的方法：元素定位等等
        cls.driver = DriverUtils.get_driver()
        # 实例login_page.py业务层对象
        cls.login_proxy = LoginProxy()

    # 类级别的销毁fixture:执行完所有的测试方法之后关闭浏览器
    @classmethod
    def tearDownClass(cls):
        # 调用工具类中关闭浏览器驱动的方法
        DriverUtils.quit_driver()

    # 方法级别fixture:每个测试方法都需要从首页开始
    def setUp(self):
        self.driver.get('http://localhost/Admin/Admin/login')

    # 定义测试方法
    @parameterized.expand([("admin", "admin123", "8888", "TPshop")])
    def test_login(self, username, password, code, expect):
        print("username={} password={} code={} expect={} ".format(username,
password, code, expect))
        # 调用登陆PO文件业务层登陆的方法
        self.login_proxy.test_login(username, password, code)
        # 断言
        self.assertIn(expect,self.driver.title)
```