一、练习题

选择题

1.以下对数据驱动理解有误的是：

- [ ] A.数据驱动是一种理念或者理解为思想，其实现方式也可以有多种；
- [ ] B.黑盒测试对某个功能不同数据输入验证不同的数据结果也可以理解为数据驱动；
- [ ] C.web自动化测试要实现数据驱动需要依赖参数化技术；
- [x] D.web自动化使用数据驱动分离测试数据和测试脚本后，就不在需要维护测试脚本了；

2.selenium数据驱动实现后数据存储形式说法错误的是：

- [ ] A.测试数据可以以多种形态存储，可以放置到文件、也可以动态生成、也可以存储在数据库；
- [x] B.使用数据驱动技术后，数据的结构可以随意定义；
- [ ] C.使用数据驱动技术后，只是将数据从测试脚本中抽离出来，在测试脚本读取数据时还是需要严格遵循参数化所要求的数据格式；
- [ ] D.测试数据的设计也需要严格按照功能测试用例数据设计要求进行，避免无意义的测试数据。

3.[多选]下面的json文件错误说明选项正确的是：

```
{"status":1,
 "msg":"\u64cd\u4f5c\u6210\u529f",
 "data":{"url":"\/index.php\/Admin\/Goods\/categoryList",},
 "name":None,
 "test":["a","b","c"],
 "data":{"url":"\/index.php\/Admin"},
 "demo":Ture,
 "address":{
     "test1":"1",
     "test2":"2"
 }
}
```

- [x] A.第三行对象中的最后一个值多了逗号；
- [x] B.同个对象中存在两个"data"键值；
- [x] C.demo对应的boolean值为小写的ture；
- [ ] D.最后两行多了一个大括号；

4.[多选]下面对json文件说明错误是：

- [ ] A.json文本对比html、xml等，描述同样的对象时其字节数更少，再数据传输的时候可以更为高效；
- [x] B.json在表示整数和浮点数时需要单引号；
- [ ] C.json的对象和列表可以相互嵌套，json中的父对象和子对象能存在同样的键名；
- [x] D.在转换json字符串为python字典时，对json字符串的格式没有要求；

简答题

1.实现读取任何文件键值并组装成parameterized所要求的格式方法：

```
import json
# 公用文件读取并执行数据组装的方法
def get_json_data(json_file_path):
    # 打开json文件
    with open(json_file_path, encoding='utf-8') as f:
        # 定义一个空列表用来存储最终需要传递给参数parameterized
        login_data = []
        # 读取json并文件类容并转换为python字典
        dict_str = json.load(f)
        # 遍历字典，取键值
        for case_data in dict_str.values():
            login_data.append(list(case_data.values()))
    return login_data
```

二、提高题

代码题

1.按下列要求day06代码题引入数据驱动

```
"""
1.定义多个测试场景测试数据：密码错误、账户不存在、不输入验证码；
2.将测试数据读取方法封装到工具类中；
"""
```

```
"""修改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

    @classmethod
    def get_json_data(cls,json_file_path):
        # 打开json文件
        with open(json_file_path, encoding='utf-8') as f:
            # 定义一个空列表用来存储最终需要传递给参数parameterized
            login_data = []
            # 读取json并文件类容并转换为python字典
            dict_str = json.load(f)
            # 遍历字典，取键值
            for case_data in dict_str.values():
                login_data.append(list(case_data.values()))
        return login_data
```

```
"""修改测试用例文件"""
# ~/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(DriverUtils.get_json_data())
    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.find_element_by_css_selector(".error").text)
```

```
"""新增~/data/login.json文件"""
{
   "login_password_error": {
        "username": "admin",
        "password": "error",
        "code": "8888",
        "expect": "账号密码不正确"
  },
    "login_account_not_exist": {
        "username": "admin1",
        "password": "admin123",
        "code": "8888",
        "expect": "账号密码不正确"
  },
    "login_code_null": {
        "username": "admin1",
        "password": "admin123",
        "code": " ",
        "expect": "验证码错误"
  }
}
```