一、练习题

选择题

1.[多选]下面对测试加载器TestLoader理解正确的是：

- [x] A.测试加载器可以加载指定目录下符合条件的文件中使用unittest组织测试用例并组装成测试套件；
- [ ] B.测试加载器可以加载指定目录下符合条件的文件，且能执行非unittest组织的测试脚本；
- [x] C.使用TestLoader加载器组织的测试套件也需要使用TextTestRunner执行；
- [x] D.TestLoader加载器通常用于版本更新后，大批量测试用例统一执行；

2.[多选]以下对于fixture理解正确选项是：

- [x] A.对测试用例环境的初始化和销毁表示一个fixture；
- [x] B.fixture通过不同的控制级别来决定初始化和销毁动作的作用范围；
- [x] C.unittest提供了3种不同的fixutre控制级别:模块级别、类级别、方法级别；
- [x] D.通过fixture可以更好的组织测试用例执行，减少部分代码冗余；

3.[多选]以下对于fixture编写方法错误的是：

- [ ] A.类级别的firture必须加上@classmetod装饰器，方法名为setUpClass(cls) tearDownClass(cls)；
- [ ] B.方法级别的fixture方法名为:setUp(self) tearDown(self)；
- [x] C.方法级别的fixture方法名可以为:setup(self) teardown(self)；
- [x] D.类级别的firture可用不使用@classmetod装饰器；

4.[多选]对于下面的测试用例说法错误的是：

```
# 导包
import unittest
password = "passwordA"
# 定义测试类
class TestDemo(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.username = "testA"
        password = "passwordB"
        print(password)

    @classmethod
    def tearDownClass(self):
        print("------>tearDownClass")

    def test_print(self):
        print(self.username)
        print(password)
```

- [ ] A.test_print方法中打印的信息为:testA 和 passwordA；

- [ ] B.setUpClass初始化fixture方法打印的信息为：passwordB；
- [ ] C.fixture方法的cls和测试方法的self都是指的当前测试类：TestDemo；
- [x] D.test_print方法可以直接使用setUpClass中定义局部变量password；

5.[多选]对于断言说法正确的是：

- [x] A.在测试用例执行完成后，需要获取页面上的某些信息来和需求或测试所期望结果做对比，而对比的结果是自动化测试用例执行是否通过依据；
- [x] B.在执行大批量测试用例时，可能是定时在服务器上面执行，需要通过脚本能自动判断用例执行结果；
- [x] C.断言时校对的期望结果需要测试工程师提前设计好；
- [x] D.自动化断言和手工测试判断是否通过的校验过程其实类似，都是判断期望结果和实际结果是否一致的过程；

6.[多选]对于unittest提供的断言方法使用正确的是：

- [x] A.断言方法由unittest.TestCase内部方法，测试用例TestDemo类继承了TestCase类，所以TestDemo可以通过self直接调用父类的方法；
- [x] B.当TestDemo中包含多个测试方法，其中一个断言失败后会抛出异常:AssertionError，但是不会影响后续测试方法的运行；
- [ ] C.自动化脚本编写语法错误，应该使用断言来判断；
- [x] D.不要用太多断言以至于让代码很晦涩；

7.[多选]：以下对参数化理解错误的是：

- [ ] A.unittest本身不支持参数化，可以通过其扩展插件parameterized来实现；
- [ ] B.@parameterized.expand()能读取的数据类型可以是以下几种格式[[]]、[()]、(())、([])；
- [ ] C.使用参数化可以将测试数据和测试脚本分离，减少代码冗余，提高代码重用性；
- [x] D.所有测试用例都应该使用参数化技术；

8.[多选]：查看下面代码选择错误说明正确的选项：

```
"""以下代码主要是为了理解参数化使用中的注意点，测试方法的设计不要纠结"""
import unittest
from parameterized import parameterized
def divide(x, y):
    return x / y

def build_data():
    [(1, 1, 1), (20, 8, 10), (200, 202, 2)]

class TestDivide(unittest.TestCase):
    @parameterized.expand((1, 1, 1), (20, 10, 8), (200, 2, 202))
    def test_001(self, x, y,expect):
        result = divide(x, y)
        self.assertEqual(expect, result)

    data = [(1, 1, 1), (20, 10, 38)]
    @parameterized.expand(data)
    def test_002(self, x, y):
        result = divide(x, y)
        self.assertEqual(expect, result)

    @parameterized.expand(build_data())
```

```
    def test_003(self, x, y, expect):
        result = divide(x, y)
        self.assertEqual(expect, result)
```

- [x] A.test_001方法参数静态数据不是数组或列表，直接以多个元组进行了传递；
- [x] B.test_002方法参数个数和数据元素中的元素个数不匹配；
- [x] C.test_003方法中构造数据方法没有返回，且数据的顺序和方法参数的顺序不对应；
- [x] D.如测试用例没有错误，运行测试类总共会运行8条测试用例；

9.[多选]：对于测试报告说法正确的是：

- [x] A.测试报告是把测试的过程和结果写成文档，对发现的问题和缺陷进行分析，为纠正软件的存在的质量问题提供依据，同时为软件验收和交付打下基础；
- [x] B.测试报告内包含了有关本次测试用例的详情，包含每条测试用例的执行结果、执行时长，测试报告可以说是一款产品质量好坏的直接体现；
- [ ] C.如测试报告中出现某测试用例不通过，则一定是系统出了BUG；
- [x] D.在测试用例编写调试过程中，可以先不用引入测试报告；

简答题

1.请写出利用HTMLTestRunner生成测试报告的基本步骤和对应代码：

```
# 需要将下载好HTMLTestRunner放置到当前项目工程下
# 1.导包
import unittest
from HTMLTestRunner import HTMLTestRunner
from test_skip import TestDivide

# 2.组织测试套件，可以使用TestSuite也可以使用TestLoader来组织测试套件
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestDivide))
# 3.定义测试报告文件路径和名称
report_path = "./report/{}-result.html".format(time.strftime("%Y%m%d%H%M%S"))
# 4.打开测试文件
with open(report_path, 'wb') as f:
    # 5.实例化HTMLTestRunner
    runner = HTMLTestRunner(f, title="v1.01版本的测试报告", description="Chrome")
    # 6.运行测试的套件
    runner.run(suite)
```

2.请写出以下代码执行后控制台打印结果信息(不能将代码拷贝到pycharm运行后进行作答)：

```
# 导包
import unittest
# 定义测试类
class TestDemo(unittest.TestCase)
    @classmethod
    def setUpClass(cls):
        print("------>setUpClass")
    @classmethod
    def tearDownClass(cls):
        print("------>tearDownClass")
    def setUp(self)
```

```
        print("------>setUp")
    def tearDown(self)
        print("------>tearDown")
    def test_1_demo01(self)
        print("------>test_1_demo01")
    def test_2_demo02(self)
        print("------>test_2_demo02")
```

二、提高题

1.使用参数化技术对day04新增地址的代码进行调整，添加断言，执行该测试用例并输出测试报告：

```
# 文件一：test_address.py
import unittest

from parameterized import parameterized
from selenium import webdriver
from day05.utils import DriverUtils

class TestAddAdress(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        # 实例化浏览器驱动对象，初始化方法
        cls.driver = webdriver.Chrome()
        cls.driver.maixmize_window()
        cls.driver.implicitly_wait(10)

    @classmethod
    def tearDownClass(cls):
        cls.driver.quit()

    # 定义测试方法
    def test_login(self):
        self.driver.get('http://localhost/')
        # 执行业务操作
        self.driver.find_element_by_css_selector('.red').click()
        self.driver.find_element_by_id('username').send_keys('15800000003')
        self.driver.find_element_by_id('password').send_keys('123456')
        self.driver.find_element_by_id('verify_code').send_keys('8888')
        self.driver.find_element_by_name('sbtbutton').click()
        self.driver.find_element_by_xpath("//*[text()='地址管理']").click()

    address_001 = [("test-custemer001", "1", "2", "3", "4", "address",
"15800000092"),
                   ("test-custemer002", "338", "339", "340", "341", "address",
"15800000093")]

    @parameterized.expand(address_001)
    def test_2_add_address(self, custemer, province, city, district, twon,
address_info, phone):
        address = [custemer, province, city, district, twon, address_info,
phone]
```

```
        print(address)
        num_old = self.driver.find_elements_by_css_selector(".gp_num2 .red")
[0].text

        # 执行新增地址
        self.driver.find_element_by_css_selector("a .co_blue").click()
        self.driver.switch_to.frame(self.driver.find_element_by_css_selector("
[id*='layui-layer-iframe']"))
        self.driver.find_element_by_name("consignee").send_keys(custemer)

        province = Select(self.driver.find_element_by_id("province"))
        province.select_by_value(province)

        city = Select(self.driver.find_element_by_id("city"))
        city.select_by_value(city)

        district = Select(self.driver.find_element_by_id('district'))
        district.select_by_value(district)

        twon = Select(self.driver.find_element_by_id("twon"))
        twon.select_by_value(twon)

        self.driver.find_element_by_id('address').send_keys(address_info)
        self.driver.find_element_by_name('mobile').send_keys(phone)
        self.driver.find_element_by_css_selector("[type='submit']").click()
        # 获取新增后的地址条数
        num_new = self.driver.find_elements_by_css_selector(".gp_num2 .red")
[0].text
        # 断言
        self.assertNotEqual(num_new,num_old)
```

```
#文件二：test_runner
# 1.导包
import unittest
from HTMLTestRunner import HTMLTestRunner
from test_address import TestAddAdress

# 2.组织测试套件，可以使用TestSuite也可以使用TestLoader来组织测试套件
suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestAddAdress))
# 3.定义测试报告文件路径和名称
report_path = "./report/{}-result.html".format(time.strftime("%Y%m%d%H%M%S"))
# 4.打开测试文件
with open(report_path, 'wb') as f:
    # 5.实例化HTMLTestRunner
    runner = HTMLTestRunner(f, title="v1.01版本的测试报告", description="Chrome")
    # 6.运行测试的套件
    runner.run(suite)
```