Table of Contents

Web自动化测试课程 1.1

第3章-UnitTest框架 1.2

- UnitTest基本使用 1.2.1
- Fixture 1.2.2
- 断言 1.2.3
- 参数化 1.2.4
- 跳过 1.2.5
- 生成HTML测试报告 1.2.6
- 附件-断言资料 1.2.7

Web自动化测试课程

序号	章节	知识点
1	第一章 Web自动化入门	1. 认识自动化及自动化测试 2. 自动化测试工具选择 3. 环境搭建
2	第二章 Selenium-API操作	1. 元素定位方式 2. 元素和浏览器的操作方法 3. 鼠标和键盘操作 4. 元素等待 5. HTML特殊元素处理 6. 窗口截图 7. 验证码处理
3	第三章 UnitTest框架	1. UnitTest基本使用 2. UnitTest断言 3. 参数化 4. 生成HTML测试报告
4	第四章 PO模式	1. 方法封装 2. PO模式介绍 3. PO模式实战
5	第五章 数据驱动	1. JSON读写 2. 数据驱动介绍 3. 数据驱动实战
6	第六章 日志收集	1. 日志相关概念 2. 日志的基本方法 3. 日志的高级方法
7	第七章 项目实战	1. 自动化测试流程 2. 项目实战演练

课程目标

1. 掌握使用Selenium进行Web自动化测试的流程和方法，并且能够完成自动化测试脚本的编写。
2. 掌握如何通过UnitTest管理用例脚本，并生成HTML测试报告。
3. 掌握使用PO模式来设计自动化测试代码的架构。
4. 掌握使用数据驱动来实现自动化测试代码和测试数据的分离。
5. 掌握使用logging来实现日志的收集。

第3章-UnitTest

目标

1. 掌握UnitTest框架的基本使用方法
2. 掌握断言的使用方法
3. 掌握如何实现参数化
4. 掌握测试报告的生成

UnitTest基本使用

目标

1. 掌握UnitTest框架的基本使用方法

---

1. UnitTest框架

1.1 什么是框架？

说明：
1. 框架英文单词framework
2. 为解决一类事情的功能集合

1.2 什么是UnitTest框架？

概念：UnitTest是Python自带的一个单元测试框架，用它来做单元测试。

1.3 为什么使用UnitTest框架？

1. 能够组织多个用例去执行
2. 提供丰富的断言方法
3. 能够生成测试报告

1.4 UnitTest核心要素

1. TestCase
2. TestSuite
3. TestRunner
4. TestLoader
5. Fixture

---

2. TestCase

说明：TestCase就是测试用例

2.1 案例

定义一个实现加法操作的函数，并对该函数进行测试

2.2 定义测试用例

1. 导包：import unittest
2. 定义测试类：新建测试类必须继承unittest.TestCase
3. 定义测试方法：测试方法名称命名必须以test开头

2.3 执行测试用例

方式一：
    使用pycharm在代码上点击鼠标右键，选择使用UnitTest运行
方式二：
    调用 unittest.main() 来运行

> 思考：如何同时运行多个测试用例？

3. TestSuite

说明：(翻译：测试套件)多条测试用例集合在一起，就是一个TestSuite
使用：
    1. 实例化： suite = unittest.TestSuite()
                (suite：为TestSuite实例化的名称)

    2. 添加用例：suite.addTest(ClassName("MethodName"))
                (ClassName：为类名；MethodName：为方法名)

    3. 添加扩展：suite.addTest(unittest.makeSuite(ClassName))
                (搜索指定ClassName内test开头的方法并添加到测试套件中)

> 提示：TestSuite需要配合TestRunner才能被执行

4. TextTestRunner

```
说明：TextTestRunner是用来执行测试用例和测试套件的
使用：
    1. 实例化： runner = unittest.TextTestRunner()

    2. 执行：    runner.run(suite)  # suite：为测试套件名称
```

需求

```
将test01.py..test10.py共10条用例，将这10条用例批量执行；
```

问题

```
1. 使用suite.addtest(unittest.makeSuite(className))导入10条测试类
2. .addtest()需要添加10次
```

5. TestLoader

```
说明：
    用来加载TestCase到TestSuite中，即加载满足条件的测试用例，并把测试用例封装成测试套件。
    使用unittest.TestLoader，通过该类下面的discover()方法自动搜索指定目录下指定开头的.py文件
，
    并将查找到的测试用例组装到测试套件；

用法：
    suite = unittest.TestLoader().discover(test_dir, pattern='test*.py')
        自动搜索指定目录下指定开头的.py文件，并将查找到的测试用例组装到测试套件
        test_dir: 为指定的测试用例的目录
        pattern：为查找的.py文件的格式，默认为'test*.py'

    也可以使用unittest.defaultTestLoader 代替 unittest.TestLoader()

运行：
    runner = unittest.TextTestRunner()
    runner.run(suite)
```

5.1 TestLoader与TestSuite区别

```
1. TestSuite需要手动添加测试用例（可以添加测试类，也可以添加测试类中某个测试方法）
2. TestLoader搜索指定目录下指定开头.py文件，并添加测试类中的所有的测试方法，不能指定添加测试方
法;
```

6. 总结

1. UnitTest框架的作用？
2. 如何定义测试用例？
3. 如何执行测试用例？
4. 如何使用TestSuite？
5. 如何运行TestSuite？
6. 如何使用TestLoader？
7. TestLoader与TestSuite的区别？

Fixture

目标

1. 掌握方法级别和类级别的Fixture
2. 了解模块级别的Fixture

---

1. Fixture

小需求：在一个测试类中定义多个测试方法，查看每个测试方法执行完所花费的时长。

说明：Fixture是一个概述，对一个测试用例环境的初始化和销毁就是一个Fixture

Fixture控制级别：
    1. 方法级别
    2. 类级别
    3. 模块级别

1.1 方法级别

使用：
    1. 初始化(前置处理):
        def setUp(self)   --> 首先自动执行
    2. 销毁(后置处理):
        def tearDown(self)   --> 最后自动执行
    3. 运行于测试方法的始末，即：运行一次测试方法就会运行一次setUp和tearDown

1.2 类级别

使用：
    1. 初始化(前置处理):
        @classmethod
        def setUpClass(cls):   --> 首先自动执行
    2. 销毁(后置处理):
        @classmethod
        def tearDownClass(cls):   --> 最后自动执行
    3. 运行于测试类的始末，即：每个测试类只会运行一次setUpClass和tearDownClass

1.3 模块级别 [了解]

使用：
1. 初始化(前置处理):
   def setUpModule() --> 首先自动执行
2. 销毁(后置处理):
   def tearDownModule() --> 最后自动执行
3. 运行于整个模块的始末，即：整个模块只会运行一次setUpModule和tearDownModule

---

2. 案例

需求：使用UnitTest框架对tpshop项目测试
1). 点击登录，进入登录页面
2). 输入用户名和密码，不输入验证码，直接点击登录按钮
3). 获取错误提示信息

2.1 示例代码

```
import time
import unittest
from selenium import webdriver

class TestLogin(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://localhost")
        self.driver.implicitly_wait(10)
        self.driver.maximize_window()

    def test_login(self):
        # 点击登录按钮
        self.driver.find_element_by_link_text("登录").click()
        # 输入用户名
        self.driver.find_element_by_id("username").send_keys("13012345678")
        # 输入密码
        self.driver.find_element_by_id("password").send_keys("123456")
        # 点击登录按钮
        self.driver.find_element_by_css_selector("[name='sbtbutton']").click()
        # 获取错误提示信息
        msg = self.driver.find_element_by_css_selector(".layui-layer-content").text
        print("msg=", msg)
```

```
    def tearDown(self):
        time.sleep(3)
        self.driver.quit()
```

2.2 总结

1. 必须继承unittest.TestCase类，setUp、tearDown才是一个Fixture
2. setUp：一般做初始化工作，比如：实例化浏览器驱动对象、浏览器最大化、设置隐式等待等
3. tearDown：一般做结束工作，比如：关闭浏览器驱动对象、退出登录等

---

3. 总结

1. 什么是Fixture？
2. Fixture控制级别有哪些？
3. 如何定义Fixture？

断言

目标

1. 理解什么是断言
2. 掌握断言assertEqual、assertIn方法
3. 了解UnitTest其他断言方法

---

1. UnitTest断言

1.1 什么是断言？

概念：让程序代替人为判断测试程序执行结果是否符合预期结果的过程

1.2 为什么要学习断言？

自动化脚本在执行的时候一般都是无人值守状态，我们不知道执行结果是否符合预期结果，所以我们需要让程序代替人
为检测程序执行的结果是否符合预期结果，这就需要使用断言。

1.3 UnitTest断言方法

说明：

1. UnitTest中提供了非常丰富的断言方法，请参考附件资料
2. 复杂的断言方法在自动化测试中几乎使用不到，所以我们只需要掌握几个常用的即可

---

2. 常用的UnitTest断言方法

序号	断言方法	断言描述
1	assertTrue(expr, msg=None)	验证expr是true，如果为false，则fail
2	assertFalse(expr, msg=None)	验证expr是false，如果为true，则fail
3	assertEqual(expected, actual, msg=None)	验证expected==actual，不等则fail 【掌握】

4	assertNotEqual(first, second, msg=None)	验证first != second, 相等则fail
5	assertIsNone(obj, msg=None)	验证obj是None，不是则fail
6	assertIsNotNone(obj, msg=None)	验证obj不是None，是则fail
7	assertIn(member, container, msg=None)	验证是否member in container【掌握】
8	assertNotIn(member, container, msg=None)	验证是否member not in container

2.1 使用方式

断言方法已经在unittest.TestCase类中定义好了，而且我们自定义的测试类已经继承了TestCase，所以在测试方法中直接调用即可。

```
import unittest

def add(x, y):
    return x + y

class TestAssert(unittest.TestCase):

    def test01(self):
        num = add(1, 2)
        self.assertEqual(3, num)

    def test02(self):
        num = add(1, 2)
        is_ok = num == 3
        self.assertTrue(is_ok)
```

3. 案例

```
需求：使用UnitTest框架对tpshop项目测试
1). 点击登录，进入登录页面
2). 输入用户名和密码，不输入验证码，直接点击登录按钮
3). 获取错误提示信息
4). 断言错误提示信息是否为“验证码不能为空!”，如果断言失败则保存截图

扩展：
    1. 图片名称为动态-时间
```

断言主要代码

```
# 获取错误提示信息
msg = self.driver.find_element_by_css_selector(".layui-layer-content").text
print("msg=", msg)
try:
    # 断言
    self.assertIn("验证码不能为空", msg)
except AssertionError as e:
    # 保存截图
    img_path = "./imgs/img{}.png".format(time.strftime("%Y%m%d-%H%M%S"))
    self.driver.get_screenshot_as_file(img_path)
    raise e
```

案例代码

```
import time
import unittest
from selenium import webdriver

class TestLogin(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.get("http://localhost")
        self.driver.implicitly_wait(10)
        self.driver.maximize_window()

    def test_login(self):
        # 点击登录按钮
        self.driver.find_element_by_link_text("登录").click()
        # 输入用户名
        self.driver.find_element_by_id("username").send_keys("13012345678")
        # 输入密码
        self.driver.find_element_by_id("password").send_keys("123456")
        # 点击登录按钮
        self.driver.find_element_by_css_selector("[name='sbtbutton']").click()
        # 获取错误提示信息
        msg = self.driver.find_element_by_css_selector(".layui-layer-content").text
        print("msg=", msg)
        try:
            # 断言
            self.assertIn("验证码不能为空", msg)
        except AssertionError as e:
            # 保存截图
            img_path = "./imgs/img{}.png".format(time.strftime("%Y%m%d-%H%M%S"))
            self.driver.get_screenshot_as_file(img_path)
            raise e
```

```
    def tearDown(self):
        self.driver.quit()

if __name__ == '__main__':
    unittest.main()
```

4. 断言总结

1. 什么是断言？
2. 需要掌握哪个断言？
3. 断言异常类？

参数化

目标

1. 掌握如何实现参数化

---

1. 参数化

1.1 小需求

需求：定义一个实现加法操作的函数，并对该函数进行测试

示例代码

```
import unittest

# 求和
def add(x, y):
    return x + y

class TestAdd(unittest.TestCase):
    def test_add_01(self):
        result = add(1, 1)
        self.assertEqual(result, 2)

    def test_add_02(self):
        result = add(1, 0)
        self.assertEqual(result, 1)

    def test_add_03(self):
        result = add(0, 0)
        self.assertEqual(result, 0)

    def test_add(self):
        test_data = [(1, 1, 2), (1, 0, 1), (0, 0, 0)]
        for x, y, expect in test_data:
            print("x={} y={} expect={}".format(x, y, expect))
            result = add(x, y)
            self.assertEqual(result, expect)
```

发现问题

```
1. 一条测试数据定义一个测试函数，代码冗余度太高
2. 一个测试函数中测试多条数据，最终只会有一个测试结果
```

1.2 参数化

通过参数的方式来传递数据，从而实现数据和脚本分离。并且可以实现用例的重复执行。
unittest测试框架，本身不支持参数化，但是可以通过安装unittest扩展插件parameterized来实现。

安装

```
pip install parameterized
```

使用方式

- 导包：from parameterized import parameterized
- 使用@parameterized.expand装饰器可以为测试函数的参数进行参数化

```
# 方式一
@parameterized.expand([(1, 1, 2), (1, 0, 1), (0, 0, 0)])
def test_add(self, x, y, expect):
    pass

# 方式二
data = [(1, 1, 2), (1, 0, 1), (0, 0, 0)]
@parameterized.expand(data)
def test_add(self, x, y, expect):
    pass

# 方式三
def build_data():
    return [(1, 1, 2), (1, 0, 1), (0, 0, 0)]

@parameterized.expand(build_data)
def test_add(self, x, y, expect):
    pass
```

示例代码

```
import unittest
from parameterized import parameterized

# 求和
```

```python
def add(x, y):
    return x + y

# 构建测试数据
def build_data():
    return [(1, 1, 2), (1, 0, 1), (0, 0, 0)]

class TestAdd(unittest.TestCase):
    @parameterized.expand([(1, 1, 2), (1, 0, 1), (0, 0, 0)])
    def test_add_1(self, x, y, expect):
        print("x={} y={} expect={}".format(x, y, expect))
        result = add(x, y)
        self.assertEqual(result, expect)

    data = [(1, 1, 2), (1, 0, 1), (0, 0, 0)]

    @parameterized.expand(data)
    def test_add_2(self, x, y, expect):
        print("x={} y={} expect={}".format(x, y, expect))
        result = add(x, y)
        self.assertEqual(result, expect)

    @parameterized.expand(build_data)
    def test_add_3(self, x, y, expect):
        print("x={} y={} expect={}".format(x, y, expect))
        result = add(x, y)
        self.assertEqual(result, expect)
```

跳过

目标

1. 掌握如何把测试函数标记成跳过

---

1. 跳过

对于一些未完成的或者不满足测试条件的测试函数和测试类，可以跳过执行。

使用方式

```
# 直接将测试函数标记成跳过
@unittest.skip('代码未完成')

# 根据条件判断测试函数是否跳过
@unittest.skipIf(condition, reason)
```

示例代码

```
import unittest

version = 35

class TestSkip(unittest.TestCase):

    @unittest.skip("代码未完成")
    def test_01(self):
        print("test_01")

    @unittest.skipIf(version <= 30, "版本大于30才会执行")
    def test_02(self):
        print("test_02")

@unittest.skip("代码未完成")
class TestSkip2(unittest.TestCase):

    def test_a(self):
        print("test_a")

    def test_b(self):
```

```
        print("test_b")
```

生成HTML测试报告

目标

1. 掌握如何生成HTML测试报告方法

---

1. 什么是HTML测试报告

说明：HTML测试报告就是执行完测试用例后，以HTML(网页)方式将执行结果生成报告

1.1 为什么要生成测试报告

1. 测试报告是本次测试结果的体现形态
2. 测试报告内包含了有关本次测试用例的详情

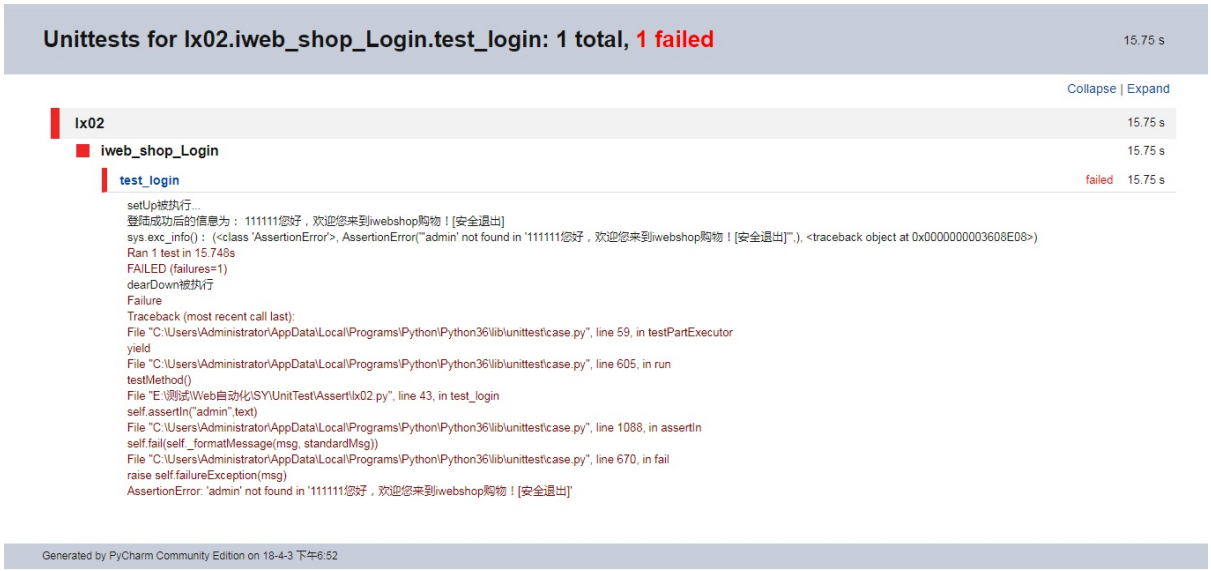
---

2. HTML生成报告方式

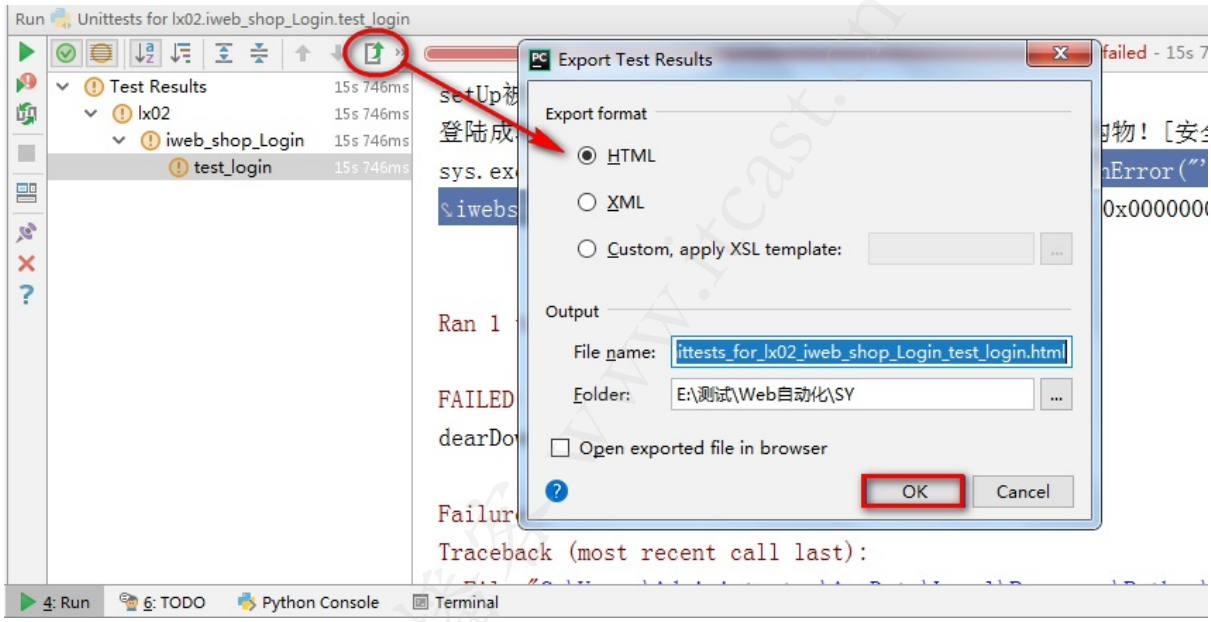
1. Export Test Results （UnitTest 自带）
2. HTMLTestRunner（第三方模板）【重点】

3. Export Test Results（自带）

3.1 测试报告截图



3.2 自带报告生成操作图



4. HTMLTestRunner【重点】

4.1 测试报告截图

iweb_shop项目Web自动化测试报告

Start Time: 2018-03-17 23:42:27
Duration: 0:01:02.024547
Status: Pass 3

测试用例共计2条

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
iweb_testCase01.TestLoginOut	2	2	0	0	Detail
iweb_testCase02.TestLogin	1	1	0	0	Detail
Total	**3**	**3**	**0**	**0**	

4.2 测试报告 生成步骤分析

1. 复制HTMLTestRunner.py文件到项目文件夹
2. 导入HTMLTestRunner、unittest包
3. 生成测试套件

```
suite = unittest.TestSuite()
suite.addTest(TestLogin("test_login"))

suite = unittest.defaultTestLoader.discover(test_dir, pattern="test*.py")
```

4. 设置报告生成路径和文件名

```
file_name = "./report/report.html"
```

5. 打开报告 `with open(file_name,'wb') as f:`
6. 实例化HTMLTestRunner对象：

```
runner = HTMLTestRunner(stream=f,[title],[description])
```

参数说明：

stream：文件流，打开写入报告的名称及写入编码格式)
title：[可选参数]，为报告标题，如XXX自动化测试报告
description：[可选参数]，为报告描述信息；比如操作系统、浏览器等版本

7. 执行：runner.run(suite)

4.3 实现代码

```
import time
import unittest
from day05.tools.HTMLTestRunner import HTMLTestRunner

# 加载指定目录下的测试用例文件
suite = unittest.defaultTestLoader.discover("./case/", "test*.py")

# 报告文件存放路径
report_path = "./report/report{}.html".format(time.strftime("%Y%m%d%H%M%S"))
with open(report_path, "wb") as f:
    # 实例化HTMLTestRunner对象，传入报告文件流f
    runner = HTMLTestRunner(stream=f, title="自动化测试报告", description="FireFox浏览器"
)
```

```
    runner.run(suite)
```

5. 总结

1. 如何使用HTMLTestRunner生成测试报告？

附件-断言资料

UnitTest断言方法

序号	断言方法	断言描述
1	assertEqual(arg1, arg2, msg=None)	验证arg1=arg2，不等则fail 【常用】
2	assertNotEqual(arg1, arg2, msg=None)	验证arg1 != arg2, 相等则fail
3	assertTrue(expr, msg=None)	验证expr是true，如果为false，则fail 【常用】
4	assertFalse(expr,msg=None)	验证expr是false，如果为true，则fail 【常用】
5	assertIs(arg1, arg2, msg=None)	验证arg1、arg2是同一个对象，不是则fail
6	assertIsNot(arg1, arg2, msg=None)	验证arg1、arg2不是同一个对象，是则fail
7	assertIsNone(expr, msg=None)	验证expr是None，不是则fail
8	assertIsNotNone(expr, msg=None)	验证expr不是None，是则fail
9	assertIn(arg1, arg2, msg=None)	验证arg1是arg2的子串，不是则fail
10	assertNotIn(arg1, arg2, msg=None)	验证arg1不是arg2的子串，是则fail
11	assertIsInstance(obj, cls, msg=None)	验证obj是cls的实例，不是则fail
12	assertNotIsInstance(obj, cls, msg=None)	验证obj不是cls的实例，是则fail
13	assertAlmostEqual (first, second, places = 7, msg = None, delta = None)	验证first约等于second。 palces: 指定精确到小数点后多少位，默认为7
14	assertNotAlmostEqual (first, second, places, msg, delta)	验证first不约等于second。 palces: 指定精确到小数点后多少位，默认为7 注： 在上述的两个函数中，如果delta指定了值，则first和second之间的差值必须≤delta
15	assertGreater (first, second, msg = None)	验证first > second，否则fail
16	assertGreaterEqual (first, second, msg = None)	验证first ≥ second，否则fail
17	assertLess (first, second, msg = None)	验证first < second，否则fail

18	assertLessEqual (first, second, msg = None)	验证first ≤ second，否则fail
19	assertRegexpMatches (text, regexp, msg = None)	验证正则表达式regexp搜索匹配的文本text。 regexp：通常使用re.search()
20	assertNotRegexpMatches (text, regexp, msg = None)	验证正则表达式regexp搜索不匹配的文本text。 regexp：通常使用re.search() 说明：两个参数进行比较（＞、≥、＜、≤、约等、不约等）
21	assertListEqual(list1, list2, msg = None)	验证列表list1、list2相等，不等则fail，同时报错信息返回具体的不同的地方
22	assertTupleEqual (tuple1, tuple2, msg = None)	验证元组tuple1、tuple2相等，不等则fail，同时报错信息返回具体的不同的地方
23	assertSetEqual (set1, set2, msg = None)	验证集合set1、set2相等，不等则fail，同时报错信息返回具体的不同的地方
24	assertDictEqual (expected, actual, msg = None	验证字典expected、actual相等，不等则fail，同时报错信息返回具体的不同的地方