day10 课堂笔记

课程之前

复习和反馈

作业

```
import unittest

from tools import login


class TestLogin(unittest.TestCase):
    def test_username_password_ok(self):
        """正确的用户名和密码"""
        if "登录成功" == login('admin', '123456'):
            print('用例通过')
        else:
            print('用例不通过')

    def test_username_error(self):
        """错误的用户名"""
        if "登录失败" == login('root', '123456'):
            print('用例通过')
        else:
            print('用例不通过')

    def test_password_error(self):
        """错误的密码"""
        if "登录失败" == login('admin', '123123'):
            print('用例通过')
        else:
            print('用例不通过')

    def test_username_password_error(self):
        """错误的用户名和密码"""
        if "登录失败" == login('aaa', '123123'):
            print('用例通过')
        else:
            print('用例不通过')
```

今日内容

用例脚本中
    断言(使用代码自动的判断预期结果和实际结果是否相符)
    参数化(将测试数据定义到 json 文件，使用)
    跳过(某些用例由于某种原因不想执行，设置为跳过)

生成测试报告 (suite 和 runner(第三方))

断言

使用代码自动的判断预期结果和实际结果是否相符
assertEqual(预期结果，实际结果)
-   判断预期结果和实际结果是否相等，如果相等， 用例通过，如果不相等，抛出异常， 用例不通过

assertIn(预期结果，实际结果)
-   判断预期结果是否包含在 实际结果中， 如果存在，用例通过， 如果不存在，抛出异常，用例不通过

```
import unittest


class TestAssert(unittest.TestCase):
    def test_equal_1(self):
        self.assertEqual(10, 10)  # 用例通过

    def test_assert_2(self):
        self.assertEqual(10, 11)  # 用例不通过

    def test_in(self):
        # self.assertIn('admin', '欢迎 admin 登录')  # 包含 通过
        # self.assertIn('admin', '欢迎 adminnnnnnnn 登录')  # 包含 通过
        # self.assertIn('admin', '欢迎 aaaaaadminnnnnnnn 登录')  # 包含 通过
        # self.assertIn('admin', '欢迎 adddddmin 登录')  # 不包含 不通过
        self.assertIn('admin', 'admin')  # 包含 通过
```

```
import unittest

from hm_02_assert import TestAssert

suite = unittest.TestSuite()

suite.addTest(unittest.makeSuite(TestAssert))
unittest.TextTestRunner().run(suite)
```

练习

```
1. 定义一个 tools 模块，在这个模块中 定义 add 的方法,可以对两个数字求和,返回求和结果
2. 书写用例，对 add()  函数进行测试
1, 1, 2
1, 2, 3
3, 4, 7
4, 5, 9
```

```python
import unittest

from tools import add


class TestAdd(unittest.TestCase):
    def test_1(self):
        self.assertEqual(2, add(1, 1))

    def test_2(self):
        self.assertEqual(3, add(1, 2))

    def test_3(self):
        self.assertEqual(7, add(3, 4))

    def test_4(self):
        self.assertEqual(9, add(4, 5))
```

参数化

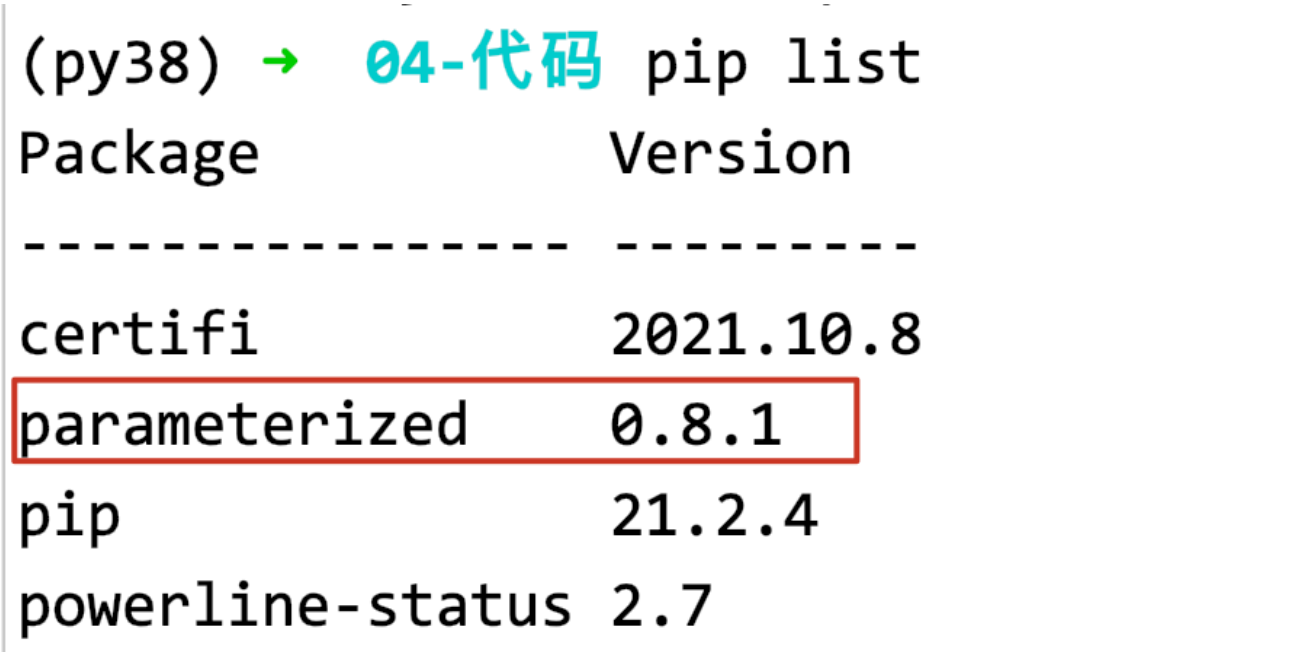
```
- 通过参数的方式来传递数据，从而实现数据和脚本分离。并且可以实现用例的重复执行。(在书写用例方法的时候，测试数据使用变量代替，在执行的时候进行据说传递)
- unittest 测试框架，本身不支持参数化，但是可以通过安装unittest扩展插 件 parameterized 来实现。
```

环境准备

```
因为参数化的插件 不是 unittest 自带的，所以想要使用 需要进行安装
Python 中 包(插件，模块) 的安装，使用 pip 工具
pip install parameterized
pip install -i https://pypi.douban.com/simple/  parameterized
# 在终端(cmd)中执行
```

```
(py38) ➜  04-代码 pip install parameterized
Looking in indexes: https://pypi.douban.com/simple/
Collecting parameterized
  Downloading https://pypi.doubanio.com/packages/31/13/fe468c8c7400a8eca2.8.1-py2.py3-none-any.whl (26 kB)
Installing collected packages: parameterized
Successfully installed parameterized-0.8.1
(py38) ➜  04-代码
```

```
pip  list  # 查看安装的所有的插件
```




```
from pa.... import pa....

from parameterized import parameterized
```

使用

1. 导包 from para... import para...
2. 修改测试方法，将测试方法中的测试数据使用 变量表示
3. 组织测试数据，格式 [(), (), ()]，一个元组就是一组测试数据
4. 参数化，在测试方法上方使用装饰器 @parameterized.expand(测试数据)
5. 运行(直接 TestCase 或者 使用 suite 运行)

```
import unittest

from tools import add
```

```
from parameterized import parameterized

data = [(1, 1, 2), (1, 2, 3), (2, 3, 5), (4, 5, 9)]


class TestAdd(unittest.TestCase):
    @parameterized.expand(data)
    def test_add(self, a, b, expect):
        print(f'a:{a}, b:{b}, expect: {expect}')
        self.assertEqual(expect, add(a, b))


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

练习

将测试数据 定义为 json 文件， 读取 json 文件，完成参数化

- json 文件

```
[
  [1, 1, 2],
  [1, 2, 3],
  [2, 3, 5],
  [4, 5, 9],
  [10, 20, 30]
]
```

- 读取 json 文件

```
import json


def build_add_data():
    with open('add_data.json') as f:
        data = json.load(f)  # [[], [], []]  ---> [(), ()]

    return data
```

- 代码文件

```
import unittest

from read_data import build_add_data

from tools import add
```

```python
from parameterized import parameterized

data = [(1, 1, 2), (1, 2, 3), (2, 3, 5), (4, 5, 9)]


class TestAdd(unittest.TestCase):
    @parameterized.expand(build_add_data())
    def test_add(self, a, b, expect):
        print(f'a:{a}, b:{b}, expect: {expect}')
        self.assertEqual(expect, add(a, b))


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

```json
[
  {
    "a": 1,
    "b": 2,
    "expect": 3
  },
  {
    "a": 11,
    "b": 22,
    "expect": 33
  },
  {
    "a": 12,
    "b": 23,
    "expect": 35
  },
  {
    "a": 14,
    "b": 25,
    "expect": 39
  }
]
```

```python
def build_add_data_1():
    with open('add_data_1.json') as f:
        data_list = json.load(f)  # [{}, {}, {}]  ----> [(), ()]

        new_list = []
        for data in data_list:  # data 字典
            # 字典中的值，是否都需要
            a = data.get('a')
            b = data.get('b')
            expect = data.get('expect')
            new_list.append((a, b, expect))
```

```
        return new_list


def build_add_data_2():
    with open('add_data_1.json') as f:
        data_list = json.load(f)  # [{}, {}, {}]  ----> [(), ()]

        new_list = []
        for data in data_list:  # data 字典
            # 字典中的值，是否都需要
            new_list.append(tuple(data.values()))

        return new_list
```

测试报告

使用第三方的报告模版，生成报告 HTMLTestReport，本质是 TestRunner

- 安装
pip install -i https://pypi.douban.com/simple/ HTMLTestReport
- 使用
1. 导包 unittest、HTMLTestReport
2. 组装用例(套件， loader)
3. 使用 HTMLTestReport 中的 runner 执行套件
4. 查看报告

```
import unittest

from htmltestreport import HTMLTestReport
from hm_04_pa1 import TestAdd

# 套件

suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestAdd))

# 运行对象
# runner = HTMLTestReport(报告的文件路径后缀.html，报告的标题，其他的描述信息)
runner = HTMLTestReport('test_add_report.html', '加法用例测试报告', 'xxx')
runner.run(suite)
```

使用绝对路径

将来的项目是分目录书写的，使用相对路径,可能会出现找不到文件的情况,此时需要使用 绝对路径

方法:
1. 在项目的根目录,创建一个 Python 文件(app.py 或者 config.py)
2. 在这个文件中 获取项目的目录,在其他代码中使用 路径拼接完成绝对路径的书写

获取当前文件的绝对路径: abspath = os.path.abspath(__file__)

获取文件路径的目录名称: dirname = os.path.dirname(filepath)

```
import os

# __file__ 特殊的变量,表示当前代码文件名
# path1 = os.path.abspath(__file__)
# print(path1)
# path2 = os.path.dirname(path1)
# print(path2)

# BASE_DIR = os.path.dirname(os.path.abspath(__file__))
BASE_DIR = os.path.dirname(__file__)


if __name__ == '__main__':
    print(BASE_DIR)
```

案例

1, 对登录函数进行测试, 登录函数 定义在 tools.py 中
2, 在 case 目录中书写用例对login 函数进行测试, 使用断言
3, 将 login 函数的测试数据定义在 json 文件中,完成参数化, data 目录中
4, 生成测试报告 report 目录中

```
def login(username, password):
    if username == 'admin' and password == '123456':
        return '登录成功'
    else:

        return '登录失败'
```

- 测试数据

```
[
  {
    "desc": "正确的用户名和密码",
    "username": "admin",
    "password": "123456",
    "expect": "登录成功"
  },
  {
    "desc": "错误的用户名",
    "username": "root",
    "password": "123456",
    "expect": "登录失败"
  },
  {
    "desc": "错误的密码",
    "username": "admin",
    "password": "123123",
    "expect": "登录失败"
  },
  {
    "desc": "错误的用户名和密码",
    "username": "root",
    "password": "123123",
    "expect": "登录失败"
  }
]
```

- 读取测试数据的方法

```
def build_login_data():
    with open(BASE_DIR + '/data/login_data.json', encoding='utf-8') as f:
        data_list = json.load(f)  # [{}, {}] ---> [()]
        new_list = []
        for data in data_list:
            # 字典中的 desc 不需要
            username = data.get('username')
            password = data.get('password')
            expect = data.get('expect')
            new_list.append((username, password, expect))

        return new_list
```

- 测试用例代码

```
import unittest

from common.read_data import build_login_data
from tools import login
from parameterized import parameterized


class TestLogin(unittest.TestCase):
    @parameterized.expand(build_login_data())
    def test_login(self, username, password, expect):
        print(f'username: {username}, password: {password}, expect: {expect}')
        self.assertEqual(expect, login(username, password))
```

- Suite 报告代码

```
import unittest

from app import BASE_DIR
from case.test_login import TestLogin
from htmltestreport import HTMLTestReport

suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestLogin))

runner = HTMLTestReport(BASE_DIR + '/report/login_report.html', '登录测试报告', 'V1.0')
runner.run(suite)
```

跳过

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(简单来说，不想执行的测试方法,可以设置为跳过)

- 直接将测试函数标记成跳过
  `@unittest.skip('跳过的原因')`

- 根据条件判断测试函数是否跳过

```
@unittest.skipIf(判断条件, reason='原因')   # 判断条件为 True, 执行跳过
```

```
import unittest

version = 29


class TestSkip(unittest.TestCase):
    @unittest.skip('没什么原因,就是不想执行')
    def test_1(self):
        print('方法一')

    @unittest.skipIf(version >= 30, '版本号大于等于 30, 测方法不用执行')
```

```
    def test_2(self):
        print('方法二')

    def test_3(self):
        print('方法三')


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