三日精通：自动化必备之Pytest测试框架训练营

第一天：pytest测试框架快速上手

主讲：北凡老师

1. 测试框架

测试框架：**抽象**出来的**工具集合**，提供大量组件、工具、功能

- 用例发现
- 用例管理
- 环境管理
- 用例执行
- 测试报告

大部分变成语言，都有测试框架：

- java：junit，testng
- **python：unittest，pytest**

	unittest	**pyetst**
安装、卸载	无需安装	手动安装
升级、降级	无法改变版本	可以指定版本
代码风格	Java语言	Python语言
插件生态	只有几个插件	1400+插件涵盖各方面
备注	由python官方维护	完全兼容unittest

2. 快速上手

安装很简单

```
pip install pytest   # 安装
pip install pytest -U  # 升级到最新版
```

pytest有三种启动方式：

1. 命令： `pytest`
2. 代码：

   ```
   import pytest

   pytest.main()
   ```

3. 鼠标【不推荐】
   1. 是由pycharm提供的，不是pytest
   2. 行为前两种方式，不一致，不适合负债项目

pytest在简单的基础上，对断言进行高级封装（AST），对python数据结构断言，非常友好

1. pytest遵循了python简单的学习方式
2. pytest实现了很多高级特性
3. 鼓励（课程要求）大家积极使用断言

3. 看懂结果

```
============================= test session starts =============================
platform win32 -- Python 3.12.0, pytest-8.3.3, pluggy-1.5.0
rootdir: E:\PyProject\my_pytest
plugins: result-log-1.2.2
collected 4 items

test_data.py FFFF                                                        [100%]

================================== FAILURES ===================================
```

```
_____________________________________ test_number ______________________________________

    def test_number():
>       assert 1 == 2
E       assert 1 == 2

test_data.py:3: AssertionError
test_data.py:24: AssertionError
========================== short test summary info ==========================
FAILED test_data.py::test_number - assert 1 == 2
============================== 4 failed in 0.07s ===============================
```

1. 执行环境：版本、根目录、用例数量
2. 执行过程：文件名称、**用例结果**、执行进度
3. 失败详情：用例内容、断言提示
4. 整体摘要：结果情况、结果数量、花费时间

用例结果缩写

缩写	单词	含义
.	passed	通过
F	failed	失败（用例执行时报错）
E	error	出错（fixture执行报错）
s	skipped	跳过
X	xpassed	预期外的通过（不符合预期）
x	xfailed	预期内的失败（符合预期）

4. 用例规则

1. 用例发现规则

测试框架在识别、加载用例的过程，称之为：**用例发现**

pytest的用例发现步骤：

1. 遍历所有的目录，例外：`venv`，`.`**开头**的目录
2. 打开python文件，`test_`**开头** 或者 `_test`**结尾**
3. 遍历所有的`Test`**开头**类

4. 收集所有的`test_`**开头的函数 或者 方法**

2. 用例内容规则

> pytest 8.4 增加了一个强制要求

pytest对用例的要求：

1. 可调用的（函数、方法、类、对象）
2. 名字`test_`开头
3. 没有参数（参数有另外含义）
4. 没有返回值 （默认为None）

3. 练习

有函数`add` 接收两个参数，并返回它们相加的结果

请为此编写测试用例

```
def add(a, b):
    return a+b


class TestAdd:

    def test_int(self):
        res = add(1,3)
        assert res == 4

    def test_str(self):
        res = add("1","3")
        assert res == "13"


    def test_list(self):
        res = add([1],[2,3,4])
        assert res == [1,2,3,4]
```

> 在Python中，类是函数的进一步抽象，由需求驱动而来

5. 配置框架

配置 可以改变pytest 默认的规则：

1. 命令参数
2. ini配置文件

所有的配置方式，可以一键获取

```
pytest -h
```

- 有哪些配置
- 分别是什么方式
  - `-`开头：参数
  - 小写字母开头：ini配置
  - 大写字母开头：环境遍历
- 配置文件: `pytest.ini`

常用参数：

- `-v` ： 增加详细程度
- `-s`： 在用例中正常的使用**输入输出**
- `-x`：快速退出，当遇到失败的用例停止执行
- `-m`： 用例筛选

6. 标记mark

标记 可以让用例与众不同，进而可以让用被区别对待

1. 用户自定义标记

用户自定义标记 只能实现用例筛选

步骤：

1. 先注册
2. 再标记
3. 后筛选

```
class TestAdd:

    @pytest.mark.api
    def test_int(self):
        res = add(1,3)
        assert res == 4

    @pytest.mark.ui
    def test_str(self):
        res = add("1","3")
        assert res == "13"

    @pytest.mark.pay
    def test_list(self):
        res = add([1],[2,3,4])
        assert res == [1,2,3,4]
#
```

```
pytest -m web
```

2. 框架内置标记

框架内置标记 为用例增加特殊执行效果

和用户自定义标记区别：

1. 不需注册，可以直接使用
2. 不仅可以筛选，还可以增加特殊效果
3. 不同的标记，增加不同的特殊效果
    - **skip： 无条件跳过**
    - **skipif：有条件跳过**
    - **xfail：预期失败**
    - **parametrize：参数化**
    - usefixtures：使用fixtures

数据驱动测试 = 参数化测试 + 数据文件

根据数据文件的内容，动态决定用例的数量、内容

7. 数据驱动测试参数

数据文件，驱动用例执行数量、内容

```
a,b,c
1,1,2
2,3,5
3,3,6
4,4,7
```

```
    @pytest.mark.ddt
    @pytest.mark.parametrize(
        "a,b,c",
        read_csv("data.csv")
    )
    def test_ddt(self,a,b,c):
        res = add(int(a),int(b))
        assert res == int(c)
```