安装：pip install pytest

优点：单元测试，复杂的功能测试，UI自动化，接口(requests)

第三方插件比较丰富，pytest-html 完美生成报告

Pytest-rerunfailures 失败case重复执行，

**可以很好的和jenkins进行结合**

Case的管理：运行指定的一个或者多个case

@pytest.mark.name //对类或者方法进行标记

运行执行的case,-----///pytest + -m + name

方法二：使用pytest-xdist分布式执行

方法三：使用特定的case进行

编写规则：文件名字test开头或者\_test结尾

测试类属性Test开头，不能带有def \_\_init\_\_方法

测试函数test开头

断言使用基本的assert即可

其识别文件的方式用的是test的命名方式

· pytest # run all tests below current dir

· pytest test\_mod.py # run tests in module file test\_mod.py

· pytest somepath # run all tests below somepath like ./tests/

· pytest -k stringexpr # only run tests with names that match the  
# the "string expression", e.g. "MyClass and not method"  
# will select TestMyClass.test\_something  
# but not TestMyClass.test\_method\_simple

· pytest test\_mod.py::test\_func # only run tests that match the "node ID",  
# e.g "test\_mod.py::test\_func" will be selected  
# only run test\_func in test\_mod.py

Pytest和unittest的不同特点：

1. 启动方式不同
2. 断言方式不同
3. 配置方式不同

框架常见名词解释：

Setup:在测试函数之前执行，完成准备工作；

Teardown:测试函数之后执行，完成收尾，例如断开链接，并且回收内存

(夹具)Fixture的使用方法

使用yiled完成setup,teardown：

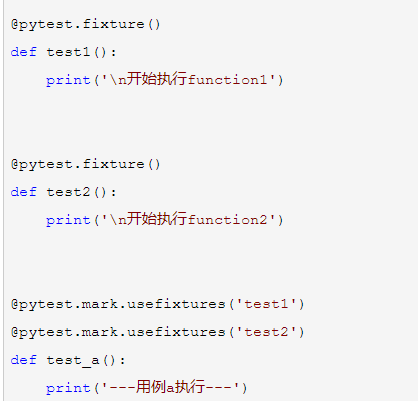
session>moudle>class>function

usefixtures和fixture的区别：

如果有返回值，即加上fixture装饰器的函数有return返回出来参数的话，必须使用fixture直接将名称或者name参数传入下一个需要使用的case，如果没有return参数的话，两种方式都可以；

注：usefixtures可以完成叠加fixture的传递：

@pytest.mark.usefixture()进行叠加，**先执行的放底层，后执行的放在上层；**



如此示例，先执行test2,后执行test1的fixture

Fixture 类似于unittest的setup,teardown

Fieture的参数：scope,params,autous,ids,name

Autous = true的话，那么不需要每个Testcase都传递一边，只需要开启，则会自动按照scope级别传递；

设立参数的传递,使用的方式@pytest.Fixture(scope=’’,name=)

**scope参数有四种，分别是'function','module','class','session'，默认为function。**

* **function：每个test都运行，默认是function的scope**
* **class：每个class的所有test只运行一次**
* **module：每个module的所有test只运行一次,并且实现多个.py文件共享前置**
* **session：实现多个.py文件，使用同一个session来完成多个用例**

Name 可以直接设定此装饰函数的名字，直接进行调用；例如：

@pytest.Fixture(scope = ,name=aa)

Def xx(self)

如果想要调用的话，需要使用xx,但是有name 参数的化，可以直接使用aa

Conftest的用法

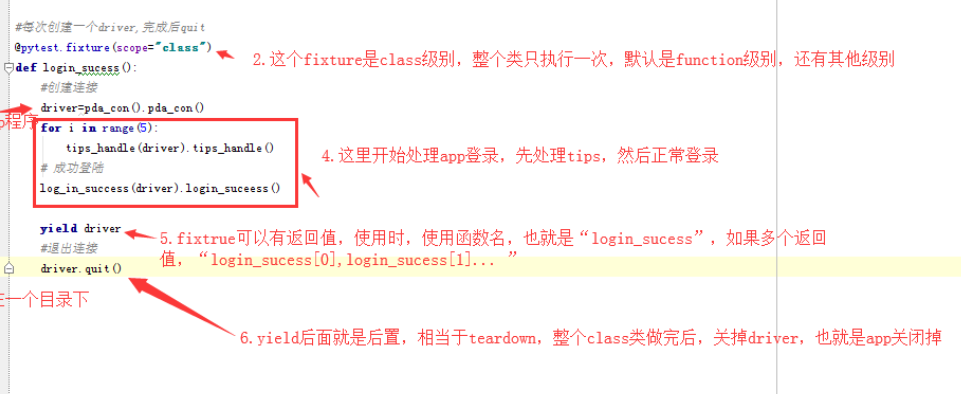
**直接试着全局范围的fixture，并且进行多个调用：**  
1.直接使用Conftest.py进行配置，数据共享，不需要import进行导入

1. 同目录下需要有init.py文件
2. Conftest.py名称不可变化
3. 需要和fixture同时使用

5.此文件需要配置在和testcase.py相同的目录下

如下：

===================================================================



**想要执行哪些case,并且控制相应的开关：**

**@pytest.mark控制执行哪些特征的testcase**

**例如：@pytest.mark.website**

**可以通过-m,’website’执行有website标记的**

可以在用例中直接传递fixture名称 传递参数，直接拿到fixture的返回值；

例如：



直接调用装饰器下面的方法名称作为参数，传入下一个函数；另外fixture允许提供参数，参数可以是list,该List中有几条数据，那么fixture就会运行几次





此时的user是list类型，可以直接将参数传到fixture中；

简单的https 接口测试可以使用pytest（）

二.

在用例结束的时候需要清理一些数据，或者清理创建的对象：



Yield 关键字 返回了 smtp

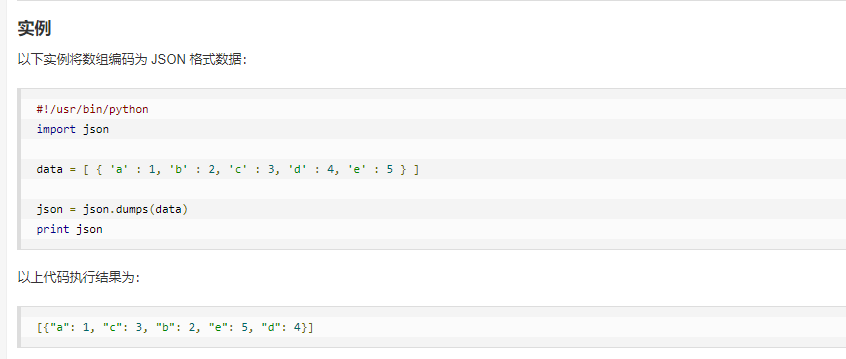
Module 中的 用例执行完成后，close（）方法会执行；

注释：python中json字符串的使用方式：

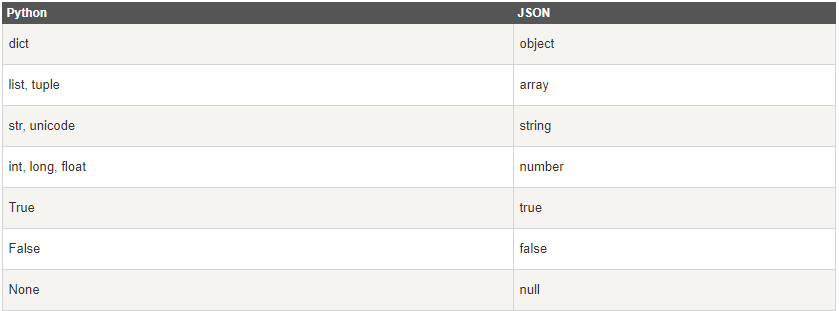
Json.dumps 将python对象转换成Json字符串；

Json.loads 将json的对象转换成python;

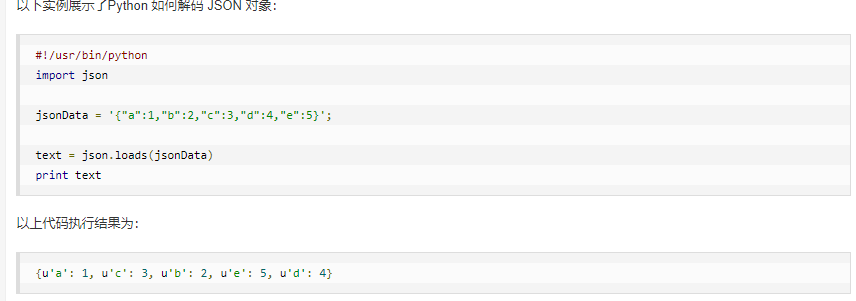
实例1：

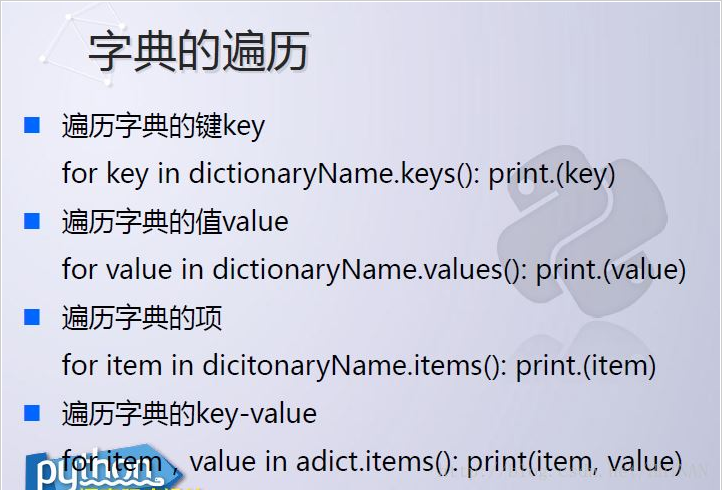


转换表：



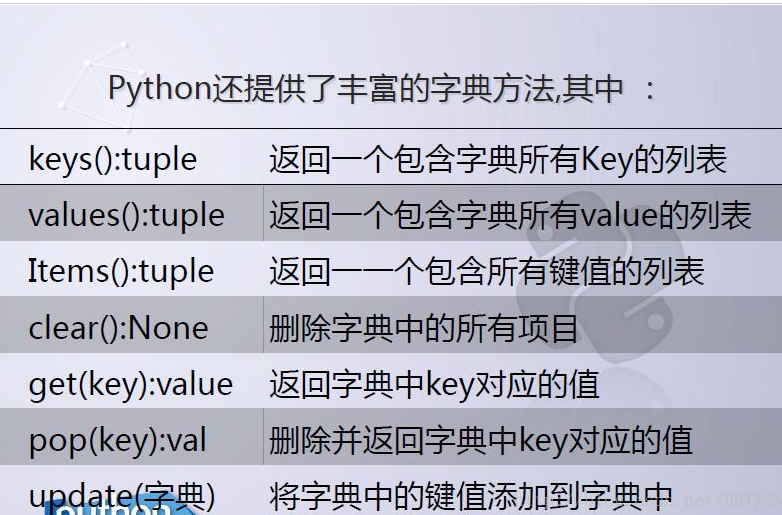
实例2





API： Key 获取键值， value 获取所有的值

Items 获取项

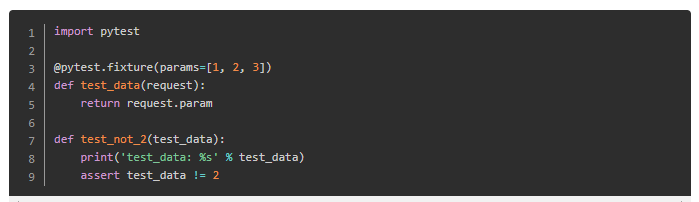


方法+返回值，列在上图；

Fixture 在pytest中的使用：

编写测试函数的时候，将此函数的名称传入参数，pytest会以依赖注入的方式，将该函数的返回值作为测试函数的传入参数

每个fixture的名字都可以触发一个函数，自身也可以调用其他fixture；**fixture主要的目的是为了提供一种可靠和可重复性的手段去运行那些最基本的测试内容**



使用params设定参数，可以将所有的参数结果读取，

传入到下一个函数；

allure装饰器的功能点：

Feature：用于定义测试的功能，被测产品的需求

Story:定义被测试功能的用户场景，即子功能

Step:测试用例分的几个步骤

Attach:报告中输入一些附加的信息，测试数据类的解释

@allure.feature( 输入功能？)

pytest生成测试报告：

安装python,allure依赖库

Pip install pytest

Pip install allure-python

Pip install allure-python-commons

安装命令行工具https://pan.baidu.com/s/1YkgYpvfmH\_I26ZPAJ1OF0A

将allure设置到环境变量之下bin目录下

通过allure-version进行验证

测试报告输出方式：

Pip install pytest -cov //计算pytest覆盖率

Pip install pytest-randomly //测试顺序随机

Pip install pytest-xdisl //分布式测试

Pip intall pytest-html //生成测试报告插件

安装的第三方插件：前缀pytest

多重断言pytest.assume()-------------api

Sugar //进度条功能可以直接用

Cov //覆盖率支持

Picked //测试未提交到git的代码

Instafail //修改默认行为，展示失败错误

其中主要包含的断言方法：断言方式

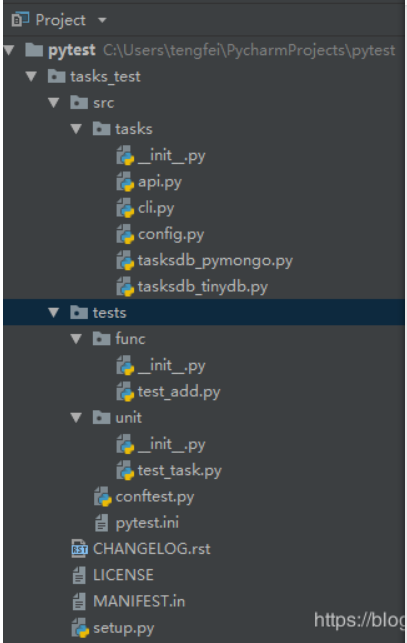
Assert xx判断xx的bool值为真

Assert not xx 判断xx的bool值不为真

Assert a in b 判断a被b包含

Assert a == b 判断a 等于b

Assert a!=b 判断a不等于b



\_\_init\_\_.py文件的作用

1. 使得整个文件夹编程一个Python模块
2. 通常这个py文件为空，但是可以增加其他的功能，可以将其他需要导入的包全部保存在同一个文件下
3. 可以配合\_\_all\_\_进行使用

\_\_all\_\_\_(包名+函数名+类名)只有其中包含的才可以使用

执行命令的方法：

**py.test --genscript=runtests.py 通过此命令生成一个执行所有脚本的py文件，但是此命令并不一定能触发**

执行testdemo.py下的测试功能

Pytest testdemo.py::test\_function1

Pytest testdemo.py::testclass::method

配合allure插件执行后生成完整的测试报告：

第一次运行方式，生成数据：pytest test\_report.py   --alluredir report

第二次把数据生成报告：allure generate report/ -o report/html --clean

此命令可以配合pytest正常命令使用

allure generate [xml\_report\_path] -o [html\_report\_path]

//[html\_report\_path]根据自己需要定义文件夹，作者定义为：/report/html

这个也是命令，需要测试相应的结果

1. pytest -s -q --alluredir allure-xml
2. allure generate allure-xml -o allure-report --clean

安装：pip install pytest

优点：单元测试，复杂的功能测试，UI自动化，接口(requests)

第三方插件比较丰富，pytest-html 完美生成报告

Pytest-rerunfailures 失败case重复执行，

可以很好的和jenkins进行结合

Case的管理：运行指定的一个或者多个case

@pytest.mark.name //对类或者方法进行标记

运行执行的case,-----///pytest + -m + name

方法二：使用pytest-xdist分布式执行

方法三：使用特定的case进行

编写规则：文件名字test开头或者\_test结尾

测试类属性Test开头，不能带有def \_\_init\_\_方法

测试函数test开头

断言使用基本的assert即可

其识别文件的方式用的是test的命名方式

 pytest # run all tests below current dir

 pytest test\_mod.py # run tests in module file test\_mod.py

 pytest somepath # run all tests below somepath like ./tests/

 pytest -k stringexpr # only run tests with names that match the  
# the "string expression", e.g. "MyClass and not method"  
# will select TestMyClass.test\_something  
# but not TestMyClass.test\_method\_simple

 pytest test\_mod.py::test\_func # only run tests that match the "node ID",  
# e.g "test\_mod.py::test\_func" will be selected  
# only run test\_func in test\_mod.py

Pytest和unittest的不同特点：

启动方式不同

断言方式不同

配置方式不同

框架常见名词解释：

Setup:在测试函数之前执行，完成准备工作；

Teardown:测试函数之后执行，完成收尾，例如断开链接，并且回收内存

(夹具，钩子函数)Fixture的使用方法

使用yiled完成setup,teardown：

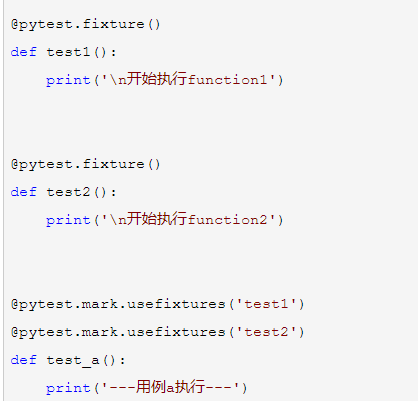
session>moudle>class>function

usefixtures和fixture的区别：

如果有返回值，即加上fixture装饰器的函数有return返回出来参数的话，必须使用fixture直接将名称或者name参数传入下一个需要使用的case，如果没有return参数的话，两种方式都可以；

注：usefixtures可以完成叠加fixture的传递：

@pytest.mark.usefixture()进行叠加，先执行的放底层，后执行的放在上层；



如此示例，先执行test2,后执行test1的fixture

Fixture 类似于unittest的setup,teardown

Fieture的参数：scope,params,autous,ids,name

Autous = true的话，那么不需要每个Testcase都传递一边，只需要开启，则会自动按照scope级别传递；

设立参数的传递,使用的方式@pytest.Fixture(scope=’’,name=)

scope参数有四种，分别是'function','module','class','session'，默认为function。

function：每个test都运行，默认是function的scope

class：每个class的所有test只运行一次

module：每个module的所有test只运行一次,并且实现多个.py文件共享前置

session：实现多个.py文件，使用同一个session来完成多个用例

Name 可以直接设定此装饰函数的名字，直接进行调用；例如：

@pytest.Fixture(scope = ,name=aa)

Def xx(self)

如果想要调用的话，需要使用xx,但是有name 参数的化，可以直接使用aa

Conftest的用法

直接试着全局范围的fixture，并且进行多个调用：  
1.直接使用Conftest.py进行配置，数据共享，不需要import进行导入

同目录下需要有init.py文件

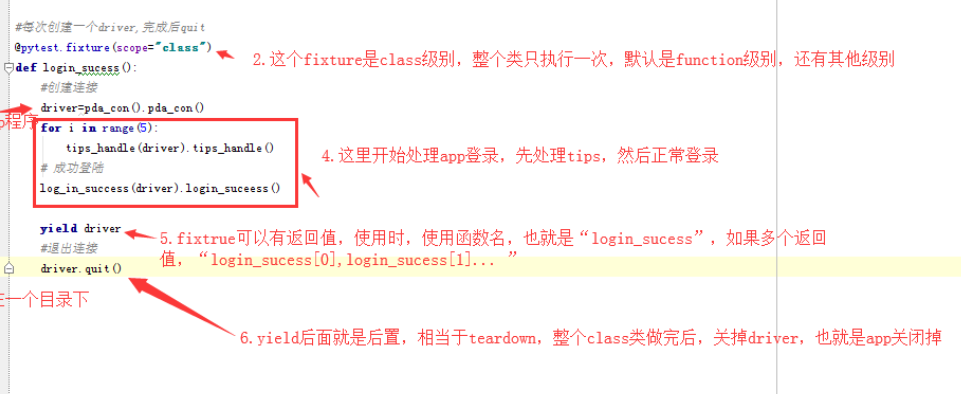
Conftest.py名称不可变化

需要和fixture同时使用

5.此文件需要配置在和testcase.py相同的目录下

如下：

===================================================================



想要执行哪些case,并且控制相应的开关：

@pytest.mark控制执行哪些特征的testcase

例如：@pytest.mark.website

可以通过-m,’website’执行有website标记的

可以在用例中直接传递fixture名称 传递参数，直接拿到fixture的返回值；

例如：



直接调用装饰器下面的方法名称作为参数，传入下一个函数；另外fixture允许提供参数，parm = [ ],是一个列表，不过可以只传入一个参数,根据scope的值进行调用





此时的user是list类型，可以直接将参数传到fixture中；

简单的https 接口测试可以使用pytest（）

二.

在用例结束的时候需要清理一些数据，或者清理创建的对象：



Yield 关键字 返回了 smtp

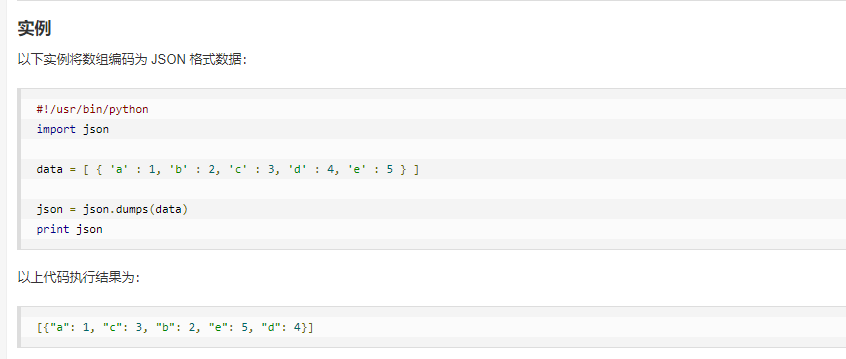
Module 中的 用例执行完成后，close（）方法会执行；

注释：python中json字符串的使用方式：

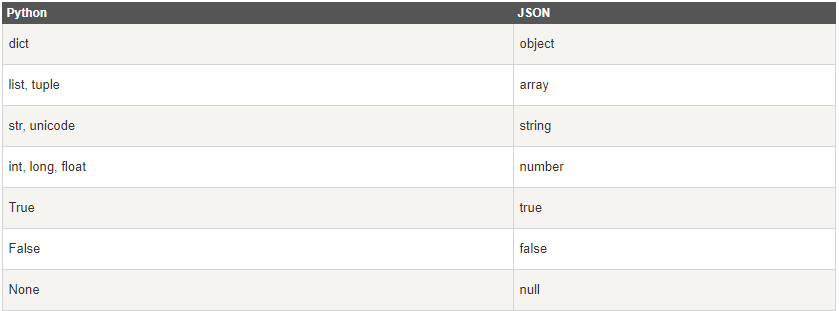
Json.dumps 将python对象转换成Json字符串；

Json.loads 将json的对象转换成python;

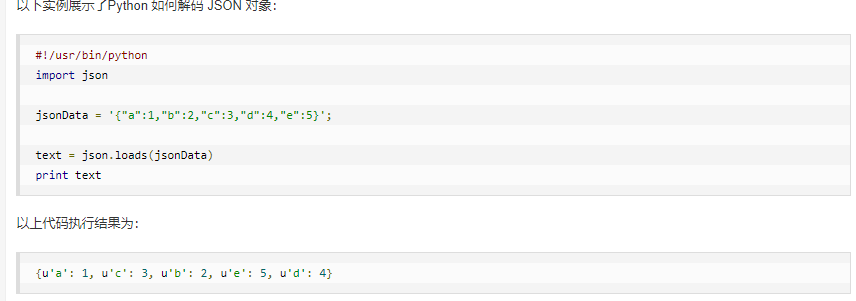
实例1：

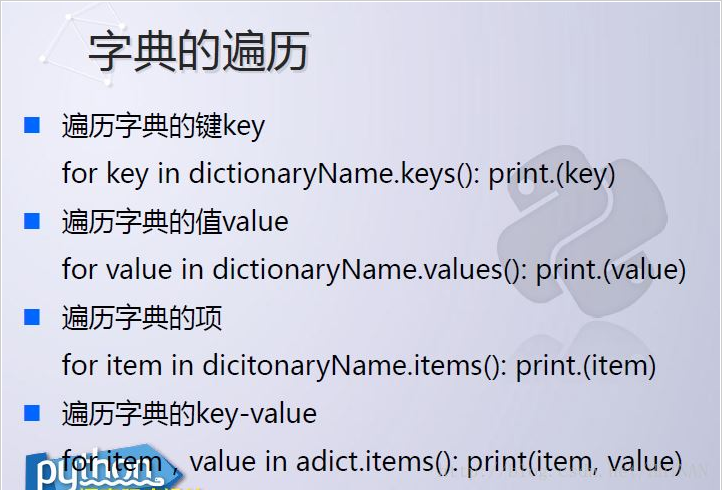


转换表：



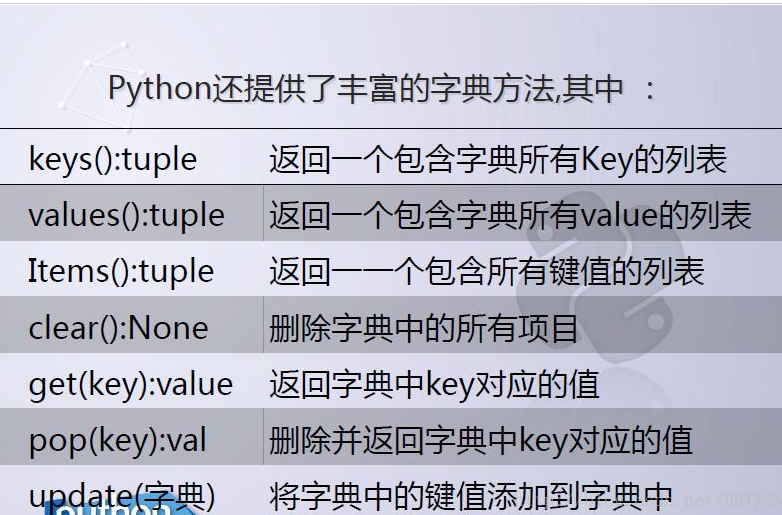
实例2





API： Key 获取键值， value 获取所有的值

Items 获取项



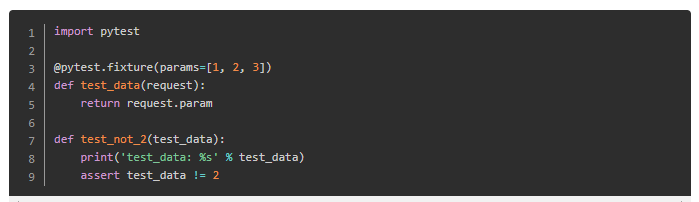
方法+返回值，列在上图；

Fixture 在pytest中的使用

：

编写测试函数的时候，将此函数的名称传入参数，pytest会以依赖注入的方式，将该函数的返回值作为测试函数的传入参数

每个fixture的名字都可以触发一个函数，自身也可以调用其他fixture；fixture主要的目的是为了提供一种可靠和可重复性的手段去运行那些最基本的测试内容



使用params设定参数，可以将所有的参数结果读取，

传入到下一个函数；

allure装饰器的功能点：

Feature：用于定义测试的功能，被测产品的需求

Story:定义被测试功能的用户场景，即子功能

Step:测试用例分的几个步骤

Attach:报告中输入一些附加的信息，测试数据类的解释

@allure.feature( 输入功能？)

pytest生成测试报告：

安装python,allure依赖库

Pip install pytest

Pip install allure-python

Pip install allure-python-commons

安装命令行工具https://pan.baidu.com/s/1YkgYpvfmH\_I26ZPAJ1OF0A

将allure设置到环境变量之下bin目录下

通过allure-version进行验证

pytest -s -q --alluredir allure-xml

allure generate allure-xml -o allure-report --clean

测试报告输出方式：

Pip install pytest -cov //计算pytest覆盖率

Pip install pytest-randomly //测试顺序随机

Pip install pytest-xdisl //分布式测试

Pip intall pytest-html //生成测试报告插件

运行pytest测试：

Pytest test\_pytest.py //运行这个文件

Pytest -m asserttest --disable-pytest-warnings

//不展示警告

如果想要展示在html上，在后面加上--html

Pytest --html=./log/report.html

在当前目录下生成一个report.html的文件

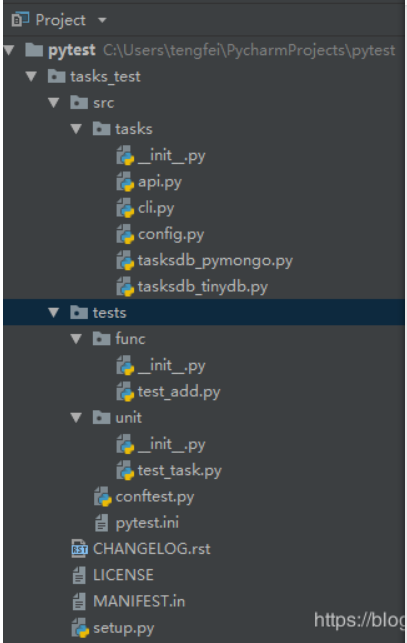
文件目录结构：

Tests 文件夹

Conftest.py

Pytest.Assume（）

进行多重断言



\_\_init\_\_.py文件的作用

使得整个文件夹编程一个Python模块

通常这个py文件为空，但是可以增加其他的功能，可以将其他需要导入的包全部保存在同一个文件下

可以配合\_\_all\_\_进行使用

\_\_all\_\_\_(包名+函数名+类名)只有其中包含的才可以使用

执行单个模块下的某个方法：

Pytest test\_mod.py::test\_function1

Pytest test\_mod.py::Testclass::method

Pytest –v 路径，执行某个包下面的测试用例

Pytest装饰器使用

Pytest.mark.parametrize(‘x,y’,[(1,2),(3,4)])

Pytest.fixture(scope = ‘’, params= date())

Date获取数据，完成外部数据驱动

添加上名字的会进行执行，同时只会执行相应的case  
@pytest.mark.smoke

Def test\_1()

执行标记为smoke,test的两个case

Pytest –v –m smoke,test

插件：前缀 pytest

Sugar //增加进度条 不需要配置

Cov //增加对覆盖率的支持

Picked //仅仅测试未提交给git的代码测试

Instafail //修改默认行为，直接展示失败，错误

Tldr //输出限制为自己需要的东西

默认输出限制为失败的case, 添加—v会显示更详细的输出

Xdist //允许通过-n标志运行多个测试

例如：pytest –n 2 将在两个CPU上运行测试

Django //为Django应用程序和项目支持pytest

Django-test-plus //对pytest进行支持

Pytest.mark.skip( ) //直接跳过某个case

Pytest.mark.skipif(1==1) //当1==1的时候跳过case

Pytest.mark.xfaili() //对case进行标记，如果通过显示xpass，如果运行失败的化展示xfail

预计展示的测试case为失败

参数为xfail(strict = True) 即标记的一定失败的

以上几个装饰器都可以使用skipif条件

其中主要包含的断言方法：

Assert xx判断xx的bool值为真

Assert not xx 判断xx的bool值不为真

Assert a in b 判断a被b包含

Assert a == b 判断a 等于b

Assert a!=b 判断a不等于b

异常值pytest.rasies()

@pytest.mark.parametrize()装饰器接收两个参数，一个以字符串形式，一个以元组或者列表(序列)形式传递测试数据

Session //所有测试文件执行之前都会执行一次

Module // 每个测试文件在执行之前都会执行一次

Class //每个测试类即，Test类都会执行一次

Function //默认类型，在每个case执行前都会执行

运行测试的时候如果要展示 pytest <测试目录> --alluredir <测试结果存放目录>

py.test --genscript=runtests.py 通过此命令生成一个执行所有脚本的py文件

allure – feature 特性

大的功能定义为一个feature

Allure – story 特性

子功能，为上一个文件的子功能

Allure-severity 特性

测试case的优先级，blocker 中等缺陷，critical临界缺陷

Normal 普通曲线 计算数值错误

Minor级别：次要缺陷 界面UI和需求不符合

Trivial级别：轻微缺陷（必须输入）

Step 执行的步骤

Issue特性，将相应的值进行处理

Allure.testcase(url) 使用用例库中的url

Allure.attach(‘xx’) 需要展示的文件内容

1.将生成的测试报告更改成你想要的格式命令

allure generate 测试结果数据所在目录 -o 测试报告保存的目录 –clean

2 .生成相关文件所在路径，首先执行的命令。通过此命令进行文件的支持

pytest 测试文件所在路径 --alluredir 生成的测试结果数据保存的目录

allure generate DIr+ -o + report +-clean

allure generate [xml\_report\_path] -o [html\_report\_path]

//[html\_report\_path]根据自己需要定义文件夹，作者定义为：/report/html

第一次运行方式，生成数据：pytest test\_report.py   --alluredir report

第二次把数据生成报告：allure generate report/ -o report/html

添加相应的测试结果，并且将相应的结果展示