

**EE308FZ Software Engineering**

**LAB 11 Software Testing**

**Preparation**

Learn related knowledge about Junit Test for Java and Unittest for python. You can finish the following assignment or test Your Project.

**Task**

1. Here are Junit Test task and Unittest task for choose. If you are good at Java , we suggest you choose the first one, if you are good at Python, then, the latter one.
2. Follow the requests and tips, complete the test task .

**Requirements**

1. You should be familiar with Junit Test or Unittest and the usage of these tools. Such as ‘assert, suite, timeout’, check problems for detail.
2. Fix the missing codes according to requirement for every problem.

**Tips:**

* We list ‘related knowledge’ for some problems, read it very carefully.

Junit Test

1. Junit assert

Related Knowledge： you can check the expected result and 5the real result of the method you test with the help of org.junit.Assert.



task：given a assert test class, complete assert test code

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

String obj1 = "junit";

String obj2 = "junit";

String obj3 = "test";

String obj4 = "test";

String obj5 = null;

int var1 = 1;

int var2 = 2;

int[] arithmetic1 = { 1, 2, 3 };

int[] arithmetic2 = { 1, 2, 3 };

@Test

public void test() {

// add assert test code between Begin and End, no other change allowed

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//assertTrue(var1 < var2); for example

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

}

1. Junit time test

if a test case takes longer time you set, then Junit will mark it failed.

tip: use ‘timeout’ and ‘@Test’ togeter

import org.junit.Test;

public class TestTimeOut {

// Fix timeout assert in Test function below. Test fail if running 1000ms longer

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

@Test()

public void test() {

while(true){}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

1. Junit parameterized test

Related knowledge：Junit parameterized test allows you test the same test with different parameters. Learning Junit parameterized test with five steps bellow.

1. Annotate test class with @RunWith(Parameterized.class).
2. Build a static method annotated by @Parameters, which retrun a set or an array of Objects as test data.
3. Build a public construct method, which receives a parameter equals with test data.
4. Build instance variable for every colum of test data.
5. Build your test case with the instance as test data source.

In Junit, you can use @RunWith and @parameter to pass parameters.

@RunWith：When a class annotated by @RunWith or when a class extends a base class which annotated by @RunWith，then Junit will run test through a runner pointed by the annotation.

@Parameters：Add this annotation to every method if it provide data. By the way, these methods must be static, return a Collection and receive no parameter.

P.S.: You must assign value for every field in the class no matter used or unused!

import static org.junit.Assert.assertEquals; // static import

import java.util.Arrays;

import java.util.Collection;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Parameterized;

import org.junit.runners.Parameterized.Parameters;

import step1.Calculator;

/\*\*

\* JUnit4 parameterized test

\*/

@RunWith(Parameterized.class)

public class ParameterTest {

private int input11;

private int input22;

private int expected;

public ParameterTest(int input11, int input22, int expected){

this.input11 = input11;

this.input22 = input22;

this.expected = expected;

}

@Parameters

public static Collection prepareData(){

/\*\*

\*the type of the two-dimension array must be Object.

\*data in the two-dimension array is ready of test sub() in Calculator

\* every element in the two-dimension array should corresponds to position of parameters in construct method ParameterTest

\*let the third element equals the first subtract the second element according to parameters’ postion

\*fix missing codes under ‘Begin’ and above ‘End’，pass 4 groups of parameters to test sub method in Calculator is right or not

\*tip：only two lines of codes

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

@Test

public void testSub(){

Calculator cal = new Calculator();

assertEquals(cal.sub(input11, input22), expected);

}

}

// Calculator.java Junit Parameterized Test

/\*\*

\* Mathematical Calculation 🡪 subtract

\*/

public class Calculator {

public int sub(int a, int b) {

return a - b;

}

}

1. **Junit Exception Test**

Related knowledge：you can check codes if throw expected exception or not by using ‘expected’ attribute in @Test meta data. value of the ‘expected attribute’ is a kind of Exception, if codes throw the expected exception, then test successfully, otherwise, failed.

import static org.junit.Assert.\*;

import org.junit.Rule;

import org.junit.Test;

import org.junit.rules.ExpectedException;

import step2.Person;

public class JunitException {

/\*\*

\*add a line of annotation in Begin/End，check the age of Person Object is legal or not. \*throw IllegalArgumentException exception

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Begin\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void checkage() {

Person person = new Person();

person.setAge(-1);

}

}

//Person.java

public class Person {

private String name;

private int age;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

if (age < 0 ) {

throw new IllegalArgumentException("age is invalid");

}

this.age = age;

}

}

1. Junit Suite Test

Related knowledge：Suite Test means test a couple of test cases together. Precisely speaking, Using @RunWith and @Suite.

import static org.junit.Assert.\*;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import step3.Calculate;

import step3.CalculateTest;

import step3.Car;

import step3.CarTest;

/\*

add two lines of annotations. Implement Suite Test of CalculateTest and CarTest

Suite Test codes must next to Class SuiteTest, no shift allowed!

\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class SuiteTest {

}

//Calculate.java

public class Calculate {

public int add(int a, int b) {

return a + b;

}

}

//CalculateTest.java

public class CalculateTest {

Calculate calculate;

@Before

public void setUp() throws Exception {

calculate = new Calculate();

}

@Test

public void testAdd() {

int result = calculate.add(12, 12);

assertEquals(24, result);

}

}

//CarTest.java

public class CarTest {

Car car;

@Before

public void setUp() throws Exception {

car = new Car();

}

@Test

public void testGetWheels() {

int result = car.getWheels();

assertEquals(4, result);

}

}

//Car.java

public class Car {

public int getWheels() {

return 4;

}

}

**unittest**

1. unittest assert test

related knowledge：

assertEqual(first, second, msg=None): return False if first not equals second, msg could be anything you want if exception be triggered.

similar method:

assertEqual(first, second, msg=None): check first == second

assertNotEqual(first, second, msg=None): check first != second

assertIn(member, container, msg=None): check member in container or not

assertNotIn(member, container, msg=None):check member not in container or yes

assertTrue(expr, msg=None): expr is True

assertFalse(expr, msg=None): expr is False

assertIsNone(self, obj, msg=None): None, obj is None

assertIsNotNone(self, obj, msg=None): None, obj is not None

import unittest, random

# Test Class

class MyClass(object):

@classmethod

def sum(self, a, b):

return a + b

@classmethod

def div(self, a, b):

return a / b

@classmethod

def retrun\_None(self):

# Unit Test Class

class MyTest(unittest.TestCase):

# assertEqual()

def test\_assertEqual(self):

# test if a+b equals sum or not

try:

a, b = 1, 2

sum = 3

self.assertEqual(a + b, sum, 'assert failed!，%s + %s != %s' %(a, b, sum))

except AssertionError as e:

print (e)

# assertNotEqual()

def test\_assertNotEqual(self):

# fix missing three lines of codes below ‘try’, test if b-a equals res or not

try:

except AssertionError as e:

print (e)

# assertTrue()

def test\_assertTrue(self):

try:

self.assertTrue(1 == 1, "False expression")

except AssertionError as e:

print (e)

# assertFalse()

def test\_assertFalse(self):

# fix missing codes below ‘try’, only a line of codes needed

try:

except AssertionError as e:

print (e)

# assertIs()

def test\_assertIs(self):

# test a and b are totally same

try:

a = 12

b = a

self.assertIs(a, b, "%s and %s are not same" %(a, b))

except AssertionError as e:

print (e)

# assertIsInstance()

def test\_assertIsInstance(self):

# fix missing codes below ‘y=object’ to test type(x) != y, only a line of codes needed

try:

x = MyClass

y = object

except AssertionError as e:

print (e)

if \_\_name\_\_ == '\_\_main\_\_':

# run unittest

unittest.main()

2. unittest test groups

class Calc(object):

def add(self, \*d):

#

result = 0

for i in d:

result += i

return result

def mul(self, \*d):

#

result =1

for i in d:

result = result\*i

return result

def sub(self,a, \*d):

#

result =a

for i in d:

result = result-i

return result

def div(self, a, \*d):

#

result =a

for i in d:

result = result/i

return result

#TestCalc.py

import unittest

import random

from Calc import Calc

class TestCalcFunctions(unittest.TestCase):

def setUp(self):

self.c=Calc()

print ("setup completed!")

def test\_sum(self):

self.assertTrue(self.c.add(1,2,3,4)==10)

def test\_sub(self):

# fix a line of codes to test c.sub(self, a, \*b) method

def test\_mul(self):

# fix a line of codes to test c.mul(self, \*b) method

def test\_div(self):

# fix a line of codes to test c.div(self, a, \*b) method

def tearDown(self):

print ("test completed!")

def tearDown(self):

print ("tearDown completed")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

# unittest\_suite.py

import random

import unittest

from TestCalc import TestCalcFunctions

class TestSequenceFunctions(unittest.TestCase):

def setUp(self):

self.seq = list(range(10))

def tearDown(self):

pass

def test\_choice(self):

# chose an element from seq randomly

element = random.choice(self.seq)

# check element is truly in the sequence

self.assertTrue(element in self.seq)

def test\_sample(self):

# if codes raise exception

with self.assertRaises(ValueError):

random.sample(self.seq, 20)

for element in random.sample(self.seq, 5):

self.assertTrue(element in self.seq)

class TestDictValueFormatFunctions(unittest.TestCase):

def setUp(self):

self.seq = list(range(10))

def tearDown(self):

pass

def test\_shuffle(self):

# shuffle sequence

random.shuffle(self.seq)

self.seq.sort()

self.assertEqual(self.seq, list(range(10)))

# check TypeError exception

self.assertRaises(TypeError, random.shuffle, (1, 2, 3))

if \_\_name\_\_ == '\_\_main\_\_':

# get all test methods start with ‘test’ and return a suite

suite1 = unittest.TestLoader().loadTestsFromTestCase(TestSequenceFunctions)

# please fix another two suite, suite2 of TestCalcFunctions and suite3 of TestDictValueFormatFunctions

# put more test class into suite

# you can change suites’ order, like [suite1, suite2, suite3]

suite = unittest.TestSuite([suite2, suite1,suite3])

# set verbosity = 2 you could get more detailed information

unittest.TextTestRunner(verbosity = 2).run(suite)

1. unittest skip test

#encoding=utf-8

import random,sys,unittest

class TestSeqFunctions(unittest.TestCase):

a = 1

def setUp(self):

self.seq = list(range(20))

@unittest.skip("skipping") # skip this method anyway

def test\_shuffle(self):

random.shuffle(self.seq)

self.seq.sort()

self.assertEqual(self.seq,list(range(20)))

self.assertRaises(TypeError,random.shuffle,(1,2,3))

# add a line of annotation that skip this method if a>5

def test\_choice(self):

element = random.choice(self.seq)

self.assertTrue(element in self.seq)

# add a line of annotation that skip if not in linux platform

def test\_sample(self):

with self.assertRaises(ValueError):

random.sample(self.seq, 20)

for element in random.sample(self.seq, 5):

self.assertTrue(element in self.seq)

if \_\_name\_\_=="\_\_main\_\_":

# unittest.main()

suite = unittest.TestLoader().loadTestsFromTestCase(TestSeqFunctions)

suite = unittest.TestSuite(suite)

unittest.TextTestRunner(verbosity = 2).run(suite)

1. unittest Run test under numerical order or alpha order.

#encoding=utf-8

import unittest

from Calc import Calc

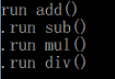
class MyTest(unittest.TestCase):

@classmethod

def setUpClass(self):

print ("init Calc before unittest")

self.c = Calc()

# rename the four methods bellow, make sure print queue will be :  


# P.S.: test case must starts with ‘test’

def add(self):

print ("run add()")

self.assertEqual(self.c.add(1, 2, 12), 15, 'test add fail')

def sub(self):

print ("run sub()")

self.assertEqual(self.c.sub(2, 1, 3), -2, 'test sub fail')

def mul(self):

print ("run mul()")

self.assertEqual(Calc.mul(2, 3, 5), 30, 'test mul fail')

def div(self):

print ("run div()")

self.assertEqual(Calc.div(8, 2, 4), 1, 'test div fail')

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

1. unittest Time Test

Related Knowledge：import time\_decorator and use it before test methods

import time

import timeout\_decorator

class timeoutTest(unittest.TestCase):

# set a time decorator(annotation) which will be triggered after 5s’ running

def testtimeout(self):

print "Start"

for i in range(1,10):

time.sleep(1)

print "%d seconds have passed" % i

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()