# 实验三 单元测试实验

# 引言

## 标识

本文档测试于以下测试环境

系统：Windows 11

## 实验要求

1. 通过实验，理解单元测试原理，熟悉单元测试工具的使用。
2. 编写四则运算程序以及学生管理系统，确定测试单元，设计测试用例，借助某单元测试工具做单元测试。
3. 具体用什么单元测试工具，根据自己情况自选，如xUnit、TestNG、gtest、pytest、unittest等。 （本实验使用unnittest）
4. 在实验报告中，给出测试需求、测试设计、测试用例集、测试执行结果及分析。

# 测试需求

测试一：对于如下四则运算程序进行unittest单元测试，其测试需求如下：

1. 测试加法方法add(a, b)的计算准确性，包括正整数、负整数的相加。
2. 测试减法方法subtract(a, b)的计算准确性，包括正整数、负整数的相减。
3. 测试乘法方法multiply(a, b)的计算准确性，包括正整数、负整数的相乘。
4. 测试除法方法divide(a, b)的计算准确性，包括正整数、负整数的相除，以及除数为零的情况。
5. 测试异常情况，如除数为零时是否抛出异常，以及计算结果越界时是否抛出异常。
6. 保证语句覆盖率达到100%。

# calculator.py

def add(a, b):  
 return a + b  
  
  
def subtract(a, b):  
 return a - b  
  
  
def multiply(a, b):  
 return a \* b  
  
  
def divide(a, b):  
 if b == 0:  
 raise ValueError("除数不能为零！")  
 return a / b

测试二：对于如下学生管理系统进行unittest单元测试，其测试需求如下：

1. 测试Student类的测试用例，验证student\_id和name属性的正确赋值。
2. 测试Course类的测试用例，验证course\_name和course\_code属性的正确赋值。
3. 测试Enrollment类的测试用例，验证student、course、grade属性的正确赋值。

# student\_management.py

class Student:  
 def \_\_init\_\_(self, student\_id, name):  
 self.student\_id = student\_id  
 self.name = name  
  
  
class Course:  
 def \_\_init\_\_(self, course\_code, course\_name):  
 self.course\_code = course\_code  
 self.course\_name = course\_name  
  
  
class Enrollment:  
 def \_\_init\_\_(self, student, course, grade=None):  
 self.student = student  
 self.course = course  
 self.grade = grade  
  
 def set\_grade(self, grade):  
 self.grade = grade

# 测试设计

测试一：要对每个方法进行单元测试，包括add，subtract，multiply和divide方法。对于每个方法，测试以下几个方面：

1. 正常情况下的整数输入值测试：输入合法的整数，验证计算结果是否正确。
2. 正常情况下的浮点数输入值测试：输入合法的浮点数，验证计算结果是否正确。
3. 异常输入值测试：对于divide方法，测试除数为0的情况。
4. 异常情况下的输出值测试：对于会导致异常的输入出值，验证是否能捕获并正确抛出异常。

# 测试用例集

测试一：

|  |  |  |  |
| --- | --- | --- | --- |
| 待测试方法 | 用例序号 | 输入 | 预期输出 |
| add | 1 | a = 2, b = 3 | 5 |
|  | 2 | a = -1, b = -1 | -2 |
|  | 3 | a = 0, b = 0 | 0 |
|  | 4 | a = 0.1, b = 0.2 | 0.3 |
| subtract | 5 | a = 5, b = 3 | 2 |
|  | 6 | a = -1, b = -1 | 0 |
|  | 7 | a = 0, b = 5 | -5 |
|  | 8 | a = 0.3, b = 0.1 | 0.2 |
| multiply | 9 | a = 2, b = 3 | 6 |
|  | 10 | a = -1, b = -1 | 1 |
|  | 11 | a = 0, b = 5 | 0 |
|  | 12 | a = 0.1, b = 0.2 | 0.02 |
| divide | 13 | a = 6, b = 3 | 2 |
|  | 14 | a = 1, b = 0 | “除数不能为0！” |
|  | 15 | a = 0.3, b = 0.1 | 3.0 |

测试二：

|  |  |  |  |
| --- | --- | --- | --- |
| 待测试方法 | 用例序号 | 输入 | 预期输出 |
| Student | 1 | student = Student(1, "张三") | student.student\_id == 1 |
|  | 2 | student = Student(1, "张三") | student.student\_name == “张三” |
| Course | 3 | course = Course("CS101", "计算机科学导论") | course.course\_code == “CS101” |
|  | 4 | course = Course("CS101", "计算机科学导论") | course.course\_name == “计算机科学导论” |
| Enrollment | 5 | enrollment = Enrollment(student, course) | enrollment.student == student |
|  | 6 | enrollment = Enrollment(student, course) | enrollment.course == course |
| Set\_grade | 7 | enrollment.set\_grade("A") | enrollment.grade == "A" |

# 测试执行结果与分析

测试一的测试代码如下：

# test\_calculator.py

import unittest

from calculator import add, subtract, multiply, divide

class TestCalculator(unittest.TestCase):

def test\_add(self):

self.assertEqual(add(2, 3), 5)

self.assertEqual(add(-1, -1), -2)

self.assertEqual(add(0, 0), 0)

self.assertAlmostEqual(add(0.1, 0.2), 0.3, places=1) # 浮点数

def test\_subtract(self):

self.assertEqual(subtract(5, 3), 2)

self.assertEqual(subtract(-1, -1), 0)

self.assertEqual(subtract(0, 5), -5)

self.assertAlmostEqual(subtract(0.3, 0.1), 0.2, places=1) # 浮点数

def test\_multiply(self):

self.assertEqual(multiply(2, 3), 6)

self.assertEqual(multiply(-1, -1), 1)

self.assertEqual(multiply(0, 5), 0)

self.assertAlmostEqual(multiply(0.1, 0.2), 0.02, places=2) # 浮点数

def test\_divide(self):

self.assertEqual(divide(6, 3), 2)

self.assertEqual(divide(-6, -2), 3)

with self.assertRaises(ValueError):

divide(1, 0) # 分母为零

self.assertAlmostEqual(divide(0.3, 0.1), 3.0, places=1) # 浮点数

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

unittest.main()

执行结果如下：

文本

描述已自动生成

所有测试用例结果的实际输出都与预期输出相同。说明代码的正确性和质量良好，程序健壮性强，能有效处理异常输出。

测试二的测试代码如下：

# test\_student\_management.py

import unittest

from student\_management import Student, Course, Enrollment

class TestStudentManagement(unittest.TestCase):

def test\_student\_creation(self):

student = Student(1, "张三")

self.assertEqual(student.student\_id, 1)

self.assertEqual(student.name, "张三")

def test\_course\_creation(self):

course = Course("CS101", "计算机科学导论")

self.assertEqual(course.course\_code, "CS101")

self.assertEqual(course.course\_name, "计算机科学导论")

def test\_enrollment\_creation(self):

student = Student(1, "张三")

course = Course("CS101", "计算机科学导论")

enrollment = Enrollment(student, course)

self.assertEqual(enrollment.student, student)

self.assertEqual(enrollment.course, course)

self.assertIsNone(enrollment.grade) # grade默认为None

def test\_set\_grade(self):

student = Student(1, "张三")

course = Course("CS101", "计算机科学导论")

enrollment = Enrollment(student, course)

enrollment.set\_grade("A")

self.assertEqual(enrollment.grade, "A")

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

unittest.main()

执行结果如下：

文本

描述已自动生成

所有测试用例结果的实际输出都与预期输出相同。说明代码的正确性和质量良好，程序健壮性强，能有效处理异常输出。

# 测试记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试项目 | 测试日期 | 测试时间 | 测试人员 | 测试环境 |
| 功能测试 | 2024.12.9 | 16:00 | 赵乐君 | Windows 11 |