实验三 单元测试实验

1. 引言

1.1 标识

本文档测试于以下测试环境

系统：Windows 11

1.2 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。

2. 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%。

```python
# 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
```

3. 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。

4. 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"

5. 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：

```
============================= test session starts ==============================
collecting ... collected 4 items

test_calculator.py::TestCalculator::test_add PASSED                      [ 25%]
test_calculator.py::TestCalculator::test_divide PASSED                   [ 50%]
test_calculator.py::TestCalculator::test_multiply PASSED                 [ 75%]
test_calculator.py::TestCalculator::test_subtract PASSED                 [100%]

============================== 4 passed in 0.01s ===============================
```

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：

```
============================= test session starts ==============================
collecting ... collected 4 items

test_student_management.py::TestStudentManagement::test_course_creation PASSED [ 25%]
test_student_management.py::TestStudentManagement::test_enrollment_creation PASSED [ 50%]
test_student_management.py::TestStudentManagement::test_set_grade PASSED [ 75%]
test_student_management.py::TestStudentManagement::test_student_creation PASSED [100%]

============================== 4 passed in 0.01s ===============================
```

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。

6. 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