Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Guadalajara

Maestría en Ciencias de la Computación

TC4002.1 Análisis, diseño y construcción de software

Dr. Gerardo Padilla Zarate

Integrantes

A00354877	Héctor Gabriel Olagues Torres
A01220356	Paul Iván Gallegos Bernal

Marzo de 2021

Lab 3.2 – Unit Testing with Python

Note: For all exercises in Lab 3.2 use the unittest module in Python.

Generic test frame for all exercises (run_tests.py)

```
import sys
import os
import unittest

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), "../src")))

loader = unittest.TestLoader()
start_dir = "test/"
suite = loader.discover(start_dir)

runner = unittest.TextTestRunner(verbosity=2)
```

Programming Exercise 1

Given the math.ceil function:

- Define a set of unit test cases that exercise the function (Remember Right BICEP)

Test cases (test_math_ceil.py)

```python
import unittest
import math


class TestMathCeil(unittest.TestCase):
    def test_positive_number(self):
        self.assertEqual(math.ceil(101.96), 102)

    def test_almost_zero(self):
        self.assertEqual(math.ceil(0.1), 1)
        self.assertEqual(math.ceil(-0.1), 0)

    def test_is_zero(self):
        self.assertEqual(math.ceil(0.0), 0)

    def test_negative_number(self):
        self.assertEqual(math.ceil(-13.1), -13)
```

Programming Exercise 2

Given the filecmp.cmp function:

- Define a set of test cases that exercise the function (Remember Right BICEP)

Test cases (test_filecmp_cmp.py)

```
import unittest
import filecmp


class TestFilecmpCmp(unittest.TestCase):
    def test_different_content_shallow(self):
        # file0 = os.path.join(os.path.dirname(__file__),
'../resources/file0.txt')
        # file1 = os.path.join(os.path.dirname(__file__),
'../resources/file1.txt')
        file0 = "resources/file0.txt"
        file1 = "resources/file1.txt"

        # Compare the os.stat() signature i.e the metadata
        # of both files
        comp = filecmp.cmp(file0, file1)
        self.assertFalse(comp)

    def test_same_content_shallow(self):
        file0 = "resources/file0.txt"
        file1 = "resources/file0_copy.txt"

        # Compare the os.stat() signature i.e the metadata
        # of both files
        comp = filecmp.cmp(file0, file1)
        self.assertTrue(comp)

    def test_different_content_no_shallow(self):
        file0 = "resources/file0.txt"
        file1 = "resources/file1.txt"

        # Compare the contents of both files
        comp = filecmp.cmp(file0, file1, False)
        self.assertFalse(comp)

    def test_same_content_no_shallow(self):
        file0 = "resources/file0.txt"
        file1 = "resources/file0_copy.txt"

        # Compare the contents of both files
        comp = filecmp.cmp(file0, file1, False)
        self.assertTrue(comp)
```

Programming Exercise 3

Implement a class that manages a directory that is saved in a text file. The data saved includes:

a. Name
b. Email
c. Age
d. Country of Origin

The class should have capabilities to:

- Add new record
- Delete a record
- Look for a record by email and age
- List on screen all record information

Source code (records_handler.py)

```
import sqlite3
from sqlite3 import Error


class RecordsHandler:
    def __init__(self, db_file):
        self.sql_create_projects_table = """ CREATE TABLE IF NOT EXISTS Records (
                                        id integer PRIMARY KEY,
                                        name text NOT NULL,
                                        email text NOT NULL,
                                        age text NOT NULL,
                                        origin text NOT NULL
                                    ); """

        self.create_connection(db_file)

    def create_connection(self, db_file):
        """ create a database connection to a SQLite database """
        self.conn = None
        try:
            self.conn = sqlite3.connect(db_file)
        except Error as e:
            print(e)

        self.create_table(self.sql_create_projects_table)

    def close_connection(self):
        if self.conn:
            self.conn.close()

    def create_table(self, create_table_sql):
        """create a table from the create_table_sql statement
        :param conn: Connection object
        :param create_table_sql: a CREATE TABLE statement
        :return:
        """
        try:
            c = self.conn.cursor()
            c.execute(create_table_sql)
            self.conn.commit()
        except Error as e:
            print(e)

    def add_record(self, name="", email="", age="", origin=""):
        sqlquery = "INSERT INTO Records (name, email, age, origin) VALUES ('%s', '%s', '%s', '%s')"
        val = (name, email, age, origin)
        query = sqlquery % val
        try:
            c = self.conn.cursor()
            c.execute(query)
            self.conn.commit()
        except Error as e:
            print(e)
```

```
def delete_record(self, record_id=-1):
    sqlquery = "DELETE FROM Records WHERE id='%s'"
    val = record_id
    query = sqlquery % val
    try:
        c = self.conn.cursor()
        c.execute(query)
        self.conn.commit()
    except Error as e:
        print(e)

def look(self, email="", age=""):
    sqlquery = "SELECT * FROM Records WHERE email='%s' AND age='%s'"
    val = (email, age)
    query = sqlquery % val

    results = []
    try:
        c = self.conn.cursor()
        c.execute(query)
        results.extend(c.fetchall())
    except Error as e:
        print(e)

    return results

def list_all(self):
    query = "SELECT * FROM Records"
    results = []
    try:
        c = self.conn.cursor()
        c.execute(query)
        results.extend(c.fetchall())
    except Error as e:
        print(e)

    # loop through the rows
    for row in results:
        print(row)

def delete_all(self):
    sqlquery = "DELETE FROM Records"
    try:
        c = self.conn.cursor()
        c.execute(sqlquery)
        self.conn.commit()
    except Error as e:
        print(e)
```

Test cases (test_records_handler.py)

```python
import unittest
import os
import io
import sys

from src.records_handler import RecordsHandler


class TestRecordsHandler(unittest.TestCase):
    def setUp(self):
        # Not mocking any object because it is time consuming to
        # discover mocking for this Lab.
        self.recordsHandler = RecordsHandler("resources/lab3DB.db")
        # Making sure that there are no records in each test case
        self.recordsHandler.delete_all()

    def test_add_record(self):
        capturedOutput = io.StringIO()  # Create StringIO object
        sys.stdout = capturedOutput  # and redirect stdout.

        self.recordsHandler.list_all()
        # No records found
        self.assertEqual(capturedOutput.getvalue(), "")

        self.recordsHandler.add_record("name", "email", "age", "origin")
        self.recordsHandler.list_all()  # Call unchanged function.
        sys.stdout = sys.__stdout__  # Reset redirect.

        # Added record found
        self.assertEqual(
            capturedOutput.getvalue(), "(1, 'name', 'email', 'age', 'origin')\n"
        )

    def test_list_all(self):
        self.recordsHandler.add_record("name", "email", "age", "origin")

        capturedOutput = io.StringIO()  # Create StringIO object
        sys.stdout = capturedOutput  # and redirect stdout.
        self.recordsHandler.list_all()  # Call unchanged function.
        sys.stdout = sys.__stdout__  # Reset redirect.

        self.assertEqual(
            capturedOutput.getvalue(), "(1, 'name', 'email', 'age', 'origin')\n"
        )

def test_look(self):
    record = self.recordsHandler.look("email2", "age2")
    self.assertEqual(record, [])

    self.recordsHandler.add_record("name2", "email2", "age2", "origin2")
    record = self.recordsHandler.look("email2", "age2")
    self.assertEqual(record[0], (1, "name2", "email2", "age2", "origin2"))
```

```
def test_delete_record(self):
    capturedOutput = io.StringIO()  # Create StringIO object
    sys.stdout = capturedOutput  # and redirect stdout.

    self.recordsHandler.list_all()
    self.assertEqual(capturedOutput.getvalue(), "")

    self.recordsHandler.add_record("name1", "email1", "age1", "origin1")
    self.recordsHandler.add_record("name2", "email2", "age2", "origin2")
    self.recordsHandler.add_record("name3", "email3", "age3", "origin3")

    self.recordsHandler.list_all()

    self.assertEqual(
        capturedOutput.getvalue(),
        "(1, 'name1', 'email1', 'age1', 'origin1')\n"
        "(2, 'name2', 'email2', 'age2', 'origin2')\n"
        "(3, 'name3', 'email3', 'age3', 'origin3')\n",
    )

    self.recordsHandler.delete_record(1)

    capturedOutput = io.StringIO("")  # Clearing buffer
    sys.stdout = capturedOutput

    self.recordsHandler.list_all()

    self.assertEqual(
        capturedOutput.getvalue(),
        "(2, 'name2', 'email2', 'age2', 'origin2')\n"
        "(3, 'name3', 'email3', 'age3', 'origin3')\n",
    )

    sys.stdout = sys.__stdout__  # Reset redirect.

def tearDown(self):
    self.recordsHandler.close_connection()
```

Test case evidence for all exercises