Estructuras de Datos

Propuesta de Proyecto

Integrantes:

- Juan Diego Vaca
- Christian Santamaria
- Johana Duchi
- Edwin Jaramillo

Tema: Unit testing. Evaluación de algoritmos y sus tiempos de ejecución.

De forma general, nuestro proyecto se centra en elaborar un programa que reciba una función (previamente conocida y estudiada en el curso: conocimiento de los parámetros de entrada y qué retorna), se le aplique una determinada cantidad de pruebas, tanto sencillas como robustas, a fin de determinar si la implementación ha sido correcta, considerando todos los posibles casos; asimismo, se analizará el tiempo de ejecución del algoritmo proporcionado.

Herramientas necesarias:

Para el desarrollo del proyecto se necesitará aplicar *Unit Testing*, un método de chequeo de software, en el que unidades individuales de código fuente son evaluados para determinar si están aptos para ser utilizados. El objetivo es diseñar e implementar los criterios/resultados que son útiles para verificar si la unidad de código puesta a prueba es correcta. Cualquier fallo en el algoritmo evaluado será reportado dentro de un resumen al final de la inspección.

Para programar unit tests, se recomienda empezar con tests en una unidad tan pequeña como sea posible en el código, y posteriormente aumentar de tamaño estas unidades para ver como las más pequeñas interactúan con otras.

Los pasos básicos para implementar un Unit Test son:

- Importar la librería *unittest*.

```
import unittest
```

- Crear una clase que será la responsable de verificar el código, y que esta heredel el módulo *unittest*.

```
class TestCuboid(unittest.TestCase):
```

- Definir los métodos de verificación, en base a criterio de la persona que evalúa. Para esto lo métodos *assert* son especialmente útiles.

```
class TestCuboid(unittest.TestCase):
    def test_volume(self):
            self.assertAlmostEqual(cuboid_volume(2),8)
            self.assertAlmostEqual(cuboid_volume(1),1)
            self.assertAlmostEqual(cuboid_volume(0),0)
            self.assertAlmostEqual(cuboid_volume(5.5),166.373)
```

Otro punto importante es cómo "importar" y pasar como argumento los algoritmos a evaluar. Existen varias formas de hacerlo, sin embargo, una de las más sencillas es buscar y ejecutar un script de Python que contenga únicamente la función que se quiere testear.

Una librería que ayuda en buena manera es *runpy*, es una librería que permite localizar y ejecutar módulos de Python.

runpy provee dos funciones, de acuerdo a la necesidad:

- *runpy.run_module*

```
runpy.run_module(mod_name, init_globals=None, run_name=None, alter_sys=False)
```

Ejecuta el código del módulo especificado y devuelve el diccionario global del módulo resultante.

- *runpy.run_path*

```
runpy.run_path(path_name, init_globals=None, run_name=None)
```

Ejecuta el código en la ubicación del sistema de archivos indicada y devuelve el diccionario global del módulo resultante.

Nuestra meta es que mediante la aplicación y la profunda investigación de estas herramientas se logrará crear Unit Tests para los algoritmos que se han realizado en los deberes envíados a lo largo del curso.

Referencias

Pozo, L. (2019). How to Run Your Python Scripts. https://realpython.com/run-python-scripts/#:~:text=To%20run%20Python%20scripts%20with,see%20the%20phrase%20Hello%20World!

Python. (s.f). runpy — Localización y ejecución de módulos Python. https://docs.python.org/es/3/library/runpy.html

Sharma, A. (2020). Unit Testing in Python Tutorial. https://www.datacamp.com/tutorial/unit-testing-python