Tests-clase_1

December 12, 2020

1 Tests

Testear el código es muy importante. Es un muy buen hábito escribir tests en paralelo al código.

Los Unit tests son un método de testing en el cual cada unidad de código es puesto bajo test para determinar que funcionan tal cómo se se lo espera y que nuevos cambios no han modificado su funcionamiento.

Se espera que los desarrolladores escriban tests automáticos para asegurar que todas las partes del código y los tests en sí, tienen el comportamiento esperado.

El framework que viene dentro del estándar de Python es `unittest`.

Algunas reglas básicas de testing: * Un unit test se debe centrar en una pequeña parte del código. * Cada test unitario debe ser independiente, a veces necesitamos que todos los tests realizan tareas inciales y luego finales, en Python lo hacemos con setUp() y tearDown() * Intentar que los tests corran rápido. * Siempre es buena idea crear un hook para que se ejecuten los tests antes de realizar un push. * Cuando se debuggea es buena idea escribir un test unitario que reproduzca el bug. * Usar nombres largos y descriptivos en las funciones. Por ejemplo si estamos testeando la función `sqrt()` un test se podría llamar. `test_square_of_number()` o `test_square_negative_number()`.

1.1 `unittest`

Módulo que viene incluído en la biblioteca estándar de Python.

Para crear un Caso de Test creamos una clase que herede de `unittest.TestCase`

[17]:
```
import unittest


class MyTest(unittest.TestCase):
    def test(self):
        pass

if __name__ == '__main__':
    # unittest.main()
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.
----------------------------------------------------------------------
```

```
Ran 1 test in 0.002s

OK
```

La clase TestCase provee varios assert: * assertEqual(a, b) * assertNoEqual(a, b) * assertTrue(x) * assertFalse(x) * assertIs(a, b) * assertIsNot(a, b) * assertIsNone(a, b) * assertIsNotNote(a, b) * assertIn(a, b) * assertNotIn(a, b) * assertIsInstance(a, b) * assertIsNotInstance(a, b)

También es posible chear excepciones, warnings y mensajes de logs. * assertRaises(exe, func, *args, **kwargs) * assertRaisesRegex(exe, r, func, *args, **kwargs) * assertWarns(exe, func, *args, **kwargs) * assertWarnsRegex(exe, r, func, *args, **kwargs) * assertLogs(logger, level)

[3]:
```
# Espacio para ejemplos.
```

También tenemos otros métodos que permiten hacer checkeos más específicos.

- assertAlmostEqual(a, b) -> round(a-b, 7) == 0
- assertNotAlmostEqual(a, b) -> round(a-b, 7) != 0
- assertGreater(a, b) -> a > b
- assertGreaterEqual(a, b) -> a >= b
- assertLess(a, b) -> a < b
- assertLessEqual(a, b) -> a <= b

[4]:
```
# Espacio para ejemplos.
```