ESTRUCTURA DE DATOS Y PROGRAMACIÓN ORIENTADA A OBJETOS

Tema Nº13:Pruebas Unitarias

Indicador de logro Nº13:

Realiza pruebas unitarias utilizando un lenguaje de programación.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº13:**

Pruebas Unitarias

**Subtema 13.1:**

¿Qué son las pruebas unitarias y cómo llevar una a cabo?

Las **pruebas unitarias o *unit testing*** son una forma de comprobar que un fragmento de código funciona correctamente. Es un procedimiento más de los que se llevan a cabo dentro de una metodología ágil de trabajo.

Descubre cómo las pruebas unitarias pueden ayudarte a **mejorar el resultado final de tu aplicación**.

## **¿Qué son las pruebas unitarias?**

Las pruebas unitarias consisten en **aislar una parte del código y comprobar que funciona a la perfección**. Son pequeños *tests* que validan el comportamiento de un objeto y la lógica.

El *unit testing* suele realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente las llevan a cabo los **desarrolladores**, aunque en la práctica, también pueden realizarlas los **responsables de QA**.

Hay una especie de mito respecto a las pruebas unitarias. Algunos desarrolladores están convencidos de que son una pérdida de tiempo y las evitan buscando ahorrar tiempo.  
**Nada más alejado de la realidad.**

Con ellas se **detectan antes errores**que, sin las pruebas unitarias, no se podrían detectar hasta fases más avanzadas como las pruebas de sistema, de integración e incluso en la beta. Realizar pruebas unitarias con regularidad supone, al final, un ahorro de tiempo y dinero.

## **Motivos para realizar un test unitario**

Si todavía no estás del todo seguro de por qué debes incorporar las *unit testing* a tu día a día como desarrollador, te damos algunos motivos:

* Las pruebas unitarias demuestran que la **lógica del código** está en buen estado y que funcionará en todos los casos.
* Aumentan la **legibilidad del código** y ayudan a los desarrolladores a entender el código base, lo que facilita hacer cambios más rápidamente.
* Los test unitarios bien realizados sirven como **documentación** del proyecto.
* Se realizan en **pocos milisegundos**, por lo que podrás realizar cientos de ellas en muy poco tiempo.
* Las *unit testing* permiten al desarrollador **refactorizar el código** más adelante y tener la garantía de que el módulo sigue funcionando correctamente. Para ello se escriben casos de prueba para todas las funciones y métodos, para que cada vez que un cambio provoque un error, sea posible identificarlo y repararlo rápidamente.
* La **calidad final del código** mejorará ya que, al estar realizando pruebas de manera continua, al finalizar el código será limpio y de calidad.
* Como las pruebas unitarias dividen el código en pequeños fragmentos, es posible **probar distintas partes del proyecto** sin tener que esperar a que otras estén completadas.

## **Las 3 A’s del unit testing**

Para llevar a cabo buenas pruebas unitarias, deben estar estructuradas siguiendo las tres A’s del Unit Testing. Se trata de un **concepto fundamental respecto a este tipo de pruebas**, que describe un proceso compuesto de tres pasos.

* Arrange (organizar). Es el primer paso de las pruebas unitarias. En esta parte se **definen los requisitos** que debe cumplir el código.
* Act (actuar). Es el paso intermedio de las pruebas, el momento de **ejecutar** el *test* que dará lugar a los resultados que deberás analizar.
* Assert (afirmar). En el último paso, es el momento de comprobar si los resultados obtenidos son los que se esperaban. **Si es así, se valida y se sigue adelante**. Si no, se corrige el error hasta que desaparezca.

## **TDD y pruebas unitarias**

El ***test driven development* (TDD) o desarrollo guiado por pruebas** implica desarrollar las pruebas unitarias a las que se va a someter el software antes de escribirlo.  
De esta manera, el desarrollo se realiza atendiendo a los requisitos que se han establecido en la prueba que deberá pasar. El objetivo es conseguir un **código limpio** que funcione.  
Con esta metodología, se escoge un requisito de la lista y se plantea una prueba que se ejecuta para comprobar que falla. Si no falla puede ser porque no se ha planteado correctamente o porque la función ya estaba implementada.  
A continuación, se escribe el código que haga posible pasar la prueba de la manera más simple posible, se ejecutan las pruebas y, si todo es correcto, **se refactoriza el código para eliminar las partes duplicadas**.

Así se puede tachar ese requisito de la lista y seguir avanzando con el desarrollo.

## **Buenas prácticas para los tests unitarios**

Una vez te acostumbres a realizar este tipo de pruebas irás descubriendo todas sus ventajas. Sin embargo, si todavía no tienes experiencia, vamos a ver algunos **ejemplos de buenas prácticas en las pruebas unitarias de software**.

* Las pruebas unitarias deberían ser independientes. Si se produce cualquier tipo de mejora o cambio en los requerimientos, las pruebas unitarias no deberían verse afectados.
* Prueba sólo un código a la vez.
* Sigue un esquema claro. Puede parecer algo secundario, pero no lo es. Sé también consistente a la hora de nombrar tus *unit tests*.
* Cualquier cambio necesita pasar el test. En el caso de producirse un cambio en el código de cualquier módulo, asegúrate de que hay una prueba unitaria que se corresponda con ese módulo y que este pasa las pruebas antes de cambiar la implementación.
* Corrige los bugs identificados durante las pruebas antes de continuar. Asegúrate de realizar esta corrección antes de proseguir con la siguiente fase del ciclo de vida del desarrollo de software.
* Acostúmbrate a realizar pruebas regularmente mientras programas. Cuanto más código escribas sin testar, más caminos tendrás que revisar para encontrar errores.

No puedes esperar que las pruebas unitarias descubran todos los errores de un software, pero sí que **ahorran mucho tiempo** al facilitar localizar los errores de una manera más sencilla.  
Ya ves las múltiples ventajas de incorporar las pruebas unitarias a tu rutina de desarrollo. **¿Estás preparado?**

**Subtema 13.2:**

Practico ejemplo de prueba unitaria en Python

Python es sin duda alguna uno de los lenguajes más usados del mercado y como un potente lenguaje incluye su propio framework de pruebas, en este ejemplo veremos solo una parte de las funcionalidades que incluye el framework.

Esta es una clase muy sencilla (Promedio.py) la cual incluye un método o función que calcula el promedio entre una lista de números y retorna el resultado.

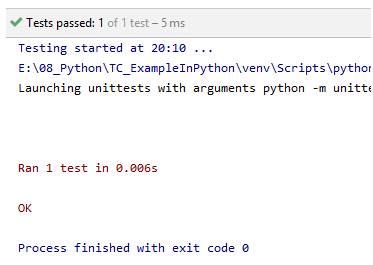
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def promedio (li):      promedio = 0      sum = 0      for el in li:          sum += el      promedio: float = sum / len(li)      return promedio |

Vamos a crear una prueba que ejecute una prueba sobre la clase Promedio.py, pasando como parámetro una lista de números y al final en la prueba vamos a ver como se valida el resultado de la prueba.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import unittest  import Promedio    class PromedioTestCase(unittest.TestCase):      def setUp(self):          self.el1 = &#91;1,2,3,4,5,6,7,8,9]        def test\_promedioInt(self):          result = Promedio.promedio(self.el1)          self.assertEqual(result, 5) |

Vamos a ver linea a linea que hace nuestra clase de prueba:

1. **import unittest e import Promedio:** el primero para indicar que vamos a usar el framework de pruebas que incluye python y el segundo para acceder a la clase que deseamos probar
2. **class PromedioTestCase(unittest.TestCase)**: indica el nombre de la clase de pruebas y especifica el tipo de contenido, en este ejemplo casos de pruebas.
3. **Setup**: como en muchos otros frameworks de pruebas es la sección donde se pueden crear objetos comunes a varios casos de pruebas o configuraciones iniciales de las pruebas
4. **test\_promedioInt**: es el nombre del test propiamente, hace una llamada al método en la clase promedio que queremos probar y guardara el valor de la operación en la variable resultado
5. **self.assertEqual(result, 5)**: finalmente se valida el resultado contra el valor esperado de nuestra prueba



Como vemos en la anterior imagen después de ejecutar el test en un IDE para python (pycharm) vemos el resumen de la ejecución, así queda visto que es muy sencillo empezar a escribir test unitarios para código escrito en python.