**Convención de Nombres para Pruebas Unitarias — Unit Test**

Esté artículo se ha basado en las opiniones de la comunidad y pretende listar las formas más populares de nombrar a las pruebas unitarias junto con ejemplos de las pruebas, con información recogida de otros artículos y fuentes. Para saber más y visitarlas:

* [TDD by Example](https://g.co/kgs/u9A5u9)
* [Clean code](https://g.co/kgs/4htXwc)
* [Unit test naming best practices](https://stackoverflow.com/questions/155436/unit-test-naming-best-practices/1594049#1594049)
* [7 popular unit test naming](https://dzone.com/articles/7-popular-unit-test-naming)
* [Unit test naming conventions](https://medium.com/@stefanovskyi/unit-test-naming-conventions-dd9208eadbea)

Es recomendable programar en inglés para favorecer que distintas personas puedan aprender y colaborar en los proyectos.

**Convenciones**

Las convenciones son básicamente estándares que recogen las reglas mínimas para hacer algo.

Ventajas:

* Facilitan la lectura y entendimiento del código.
* Reducen el coste del mantenimiento del código.
* Agilizan el desarrollo cuando colaboran varias/os desarrolladoras/os.
* Facilitan agregar o modificar funcionalidades.

Para adoptar convenciones de equipo, es recomendable que todos los miembros estén de acuerdo con la elección, priorizando las decisiones de equipo frente a dogmas personales porque es **una decisión que deberán seguir todas las personas del equipo** actuales y las futuras.

Una vez **elegida la convención**, una buena práctica es **dejar constancia** de los acuerdos sobre las convenciones sobre el proyecto.

Por ejemplo, se puede añadir un archivo dentro del proyecto con un nombre auto-explicativo, que tenga una referencia desde el “README” para facilitar su contexto y localización.

**Función para probar**

1 MIN\_AGE\_TO\_BE\_ADULT = 18

2

3 def is\_adult(age):

4 assert isinstance(age, int), "Age should be a number"

5

6 return age < MIN\_AGE\_TO\_BE\_ADULT

**Estrategia “Should…When”**

El enfoque te tiene ésta estrategia es decir expresamente que se espera y cuando se espera, es una estrategia muy popular y es la precursora de la estrategia “Roy Osherove’s” que se comenta en el siguiente punto.

Patrón: [UnitOfWork\_ShouldStateUnderTest\_WhenExpectedBehavior]

Ejemplo:

1 # Prueba es adulto... debe ser falso.. cuando tiene menos de 18 años

2 def test\_is\_adult\_should\_be\_false\_when\_age\_is\_less\_than\_18(self):

3 age = 17

4

5 adult = is\_adult(age)

6

7 self.assertEqual(adult, False)

**Variante “When…Should”**

Esta variante se utiliza con el enfoque de que la lectura del test es menos forzada que la variante “Should…When”

Patrón: [UnitOfWork\_ShouldStateUnderTest\_WhenExpectedBehavior]

Ejemplo:

1 # Prueba es adulto... debe ser falso.. cuando tiene menos de 18 años

2 def test\_is\_adult\_when\_age\_is\_less\_than\_18\_should\_be\_false(self):

3 age = 17

4

5 adult = is\_adult(age)

6

7 self.assertEqual(adult, False)

* **Ventaja:** posee toda la información que necesitamos saber sobre el método de una forma estructurada.
* **Desventaja:** La palabra “should” y “when” se vuelve repetitiva.

**Adaptación de “When y Should” aplicado al Contexto**

Podemos variar las palabras “Should” y “When” adaptándolas al contexto de la prueba, por ejemplo, en vez de “when” usar un **“with” o "without"**

Ejemplo:

1 # Prueba es adulto… debe lanzar una excepción… sin edad

2 def test\_is\_adult\_should\_throws\_exception\_without\_age(self):

3 age = '17'

4

5 with self.assertRaises(Exception):

6 is\_adult(age)

* **Ventaja**: posee toda la información que necesitamos saber sobre el método de una forma estructurada.
* **Desventaja**: La palabra “should” se vuelve repetitiva y dificulta ver que se está probando.

**Estrategia “Roy Osherove’s” State-Expected**

Es una alternativa al uso de “Should…When” que pretende que se exprese mejor el objetivo específico.

Frecuentemente se usa el nombre del método o el objeto a probar en la parte de “UnitOfWork”.

Puedes consultar en la web del creador Roy Osherove’s ([visitar blog](https://osherove.com/blog/2012/5/15/test-naming-conventions-with-unit-of-work.html)).

Patrón: [UnitOfWork\_StateUnderTest\_ExpectedBehavior]

Ejemplo:

1 # Prueba es adulto… menor de 18 años… es falso

2 def test\_is\_adult\_age\_is\_less\_than\_18\_is\_false(self):

3 age = 17

4

5 adult = is\_adult(age)

6

7 self.assertEqual(adult, False)

* **Ventaja**: posee toda la información que necesitamos saber sobre el método de una forma estructurada.
* **Desventaja**: si cambia el nombre del método hay que cambiar el nombre del test.

**Variante “Roy Osherove’s” Expected-State**

Al igual que pasa en la propuesta “Should-When” está la forma contraria de Roy Osherove’s donde primero se declara lo que se espera y luego el estado.

Patrón: [UnitOfWork\_ExpectedBehavior\_StateUnderTest]

Ejemplo:

1 # Prueba es adulto… es falso… tiene menos de 18 años

2 def test\_is\_adult\_is\_false\_less\_than\_18(self):

3 age = 17

4

5 adult = is\_adult(age)

6

7 self.assertEqual(adult, False)

* **Ventaja**: posee toda la información que necesitamos saber sobre el método de una forma estructurada.
* **Desventaja**: si cambia el nombre del método hay que cambiar el nombre del test.

**Estrategia “Unidad a testear”**

Se declara que es lo que se está probando, la característica que se va a probar forma parte del nombre de la prueba.

Patrón: [UnitOfWork]

Ejemplo:

1 # Prueba es adulto

2 def test\_is\_an\_adult(self):

3 age = 18

4

5 adult = is\_adult(age)

6

7 self.assertTrue(adult)

* **Ventaja**: Refleja de forma concisa y breve el nombre de la unidad que se prueba.
* **Desventaja**: No da detalles de la funcionalidad.

**Estrategia “Característica que se está probando”**

Aquí se pretende mostrar la funcionalidad, este enfoque es asumiendo que el método que se prueba se conoce por otros medios que no sean el nombre del test.

Patrón: [featureBeingTested]

Ejemplo:

1 # Prueba es un menor si tiene menos de 18 años

2 def test\_is\_minor\_when\_there\_are\_less\_than\_18(self):

3 age = 17

4 expected\_response = False

5

6 minor = is\_adult(age)

7

8 self.assertEqual(minor, expected\_response)

* **Ventaja**: Refleja de forma concisa y breve la funcionalidad.
* **Desventaja**: No necesariamente tiene que referenciar al método que se está probando.

Estas dos modalidades anteriores son las que recomienda Ken Beck en el libro Test Driven Development ([TDD](https://g.co/kgs/MAJgn9)): By Example escrito en el 2002 y también pueden verse en el libro de “Clean Code” de Robert C. Martin, usadas en el framework de pruebas [JUnit](https://junit.org/junit5/" \t "_blank)

“Programar haciendo TDD” (Test Driven Development), como coloquialmente decimos, consiste en programar haciendo pruebas.

Eso quiere decir que primero haces la prueba, luego la implementación y luego modificas el código para mejorarlo, con la tranquilidad que hay pruebas que te avisan si algo ha fallado.

Personalmente me ha dado muchos beneficios programar así.

**Estrategia “When…Expect”**

Al parecer guarda similitud con la forma de testear recomendada por Google Test (visitar)

Patrón: [WhenXXX\_ExpectYYYY]

Ejemplo:

1 # cuando… es adulto… esperamos… que sea falso" (o algo así literal)

2 def test\_when\_is\_less\_than\_18\_expected\_to\_be\_false(self):

3 age = 17

4

5 adult = is\_adult(age)

6

7 self.assertEqual(adult, False)

* **Ventaja**: Ayuda a generar documentación automática con TestDox
* **Desventaja**: Repetición de la palabra “when” y “Expect”.

**Estrategia “Given-When-Then”**

Esta forma es diferente a las anteriores, la codificación en el test se reemplaza por lenguaje natural.

Patrón: [Given-When-Then]

Ejemplo:

1 Scenario 1: is Adult less than 18 can not buy

2 Given he is and adult,

3 When he is less than 18,

4 Then can not buy .

* **Ventaja**: Personas no técnicas pueden escribir los test.
* **Desventaja**: Requiere el compromiso de la compañía para favorecer una estructura agilista.

Segun wikipedia, [Behavior Driven Development (BDD)](https://es.wikipedia.org/wiki/Desarrollo_guiado_por_comportamiento" \t "_blank) es un proceso de desarrollo que surgió a partir de [TDD](https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas#:~:text=Desarrollo%20guiado%20por%20pruebas%20de%20software%2C%20o%20Test%2Ddriven%20development). Combina las técnicas generales del desarrollo guiado por comportamiento, el [diseño guiado por el dominio](https://es.wikipedia.org/wiki/Dise%C3%B1o_guiado_por_el_dominio), el análisis y [diseño orientado a objetos](https://es.wikipedia.org/wiki/An%C3%A1lisis_y_dise%C3%B1o_orientado_a_objetos) y a los equipos de administración con un proceso utilizando herramientas colaborativas con un procedimiento colaborativo en el desarrollo de software.

Existen librerías como “[cucumber](https://cucumber.io/" \t "_blank)” que ayudan a que una persona no técnica pueda escribir las pruebas.