**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ**

**FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES**

**DEPARTAMENTO DE INGENIERÍA DE SOFTWARE**

**GUÍA DE LABORATORIO N° 8**

Asignatura: Mantenimiento y pruebas de software Grupos: 1SF131, 1SF132

Facilitadora: Ing. Belén Bonilla M. Fecha de asignación: 28 de octubre de 2021

Estudiante: Irving Villarreal 8-970-357

1. **TÍTULO** **DE LA EXPERIENCIA**: Desarrollo de pruebas unitarias con PHPUnit.
2. **TEMAS**

* Automatización de pruebas de software
* Pruebas unitarias

1. **OBJETIVO(S):**
2. Conocer la estructura de desarrollo de una prueba unitaria en PHPUnit.
3. Conocer y emplear diferentes aserciones que brinda PHPUnit para la creación de pruebas unitarias.
4. Conocer y emplear proveedores de datos en el desarrollo de pruebas unitarias en PHPUnit.
5. **METODOLOGÍA:**
6. Trabajar de manera individual.
7. Seguir el procedimiento indicado en esta guía.
8. Subir su archivo con los resultados y la url de su proyecto en GitHub al espacio de trabajo en Teams.
9. **PROCEDIMIENTO O ENUNCIADO DE LA EXPERIENCIA:**

**Clase Calculadora.php:**

1. Agregar a una clase Calculadora.php los métodos: sumar, restar, multiplicar, dividir. Todos estos métodos reciben dos números y devuelven el resultado de la operación indicada por el método. En el caso del método dividir, controlar mediante excepción la división por cero.
2. Agregar a la clase Calculadora.php el método generarArreglo. Este método debe crear un arreglo con 5 elementos numéricos generados aleatoriamente (rango del 1-10) y devolverlo.

**Explicación docente:** Las aserciones de PHPUnit están implementadas en PHPUnit\Framework\Assert. PHPUnit\Framework\TestCase hereda de PHPUnit\Framework\Assert.

Los métodos para hacer aserciones son declarados estáticos y se pueden invocar desde cualquier contexto usando, por ejemplo, PHPUnit\Framework\Assert::assertTrue(), $this->assertTrue() o self::assertTrue() en una clase que extiende de PHPUnit\Framework\TestCase.

A continuación, vamos a emplear algunas aserciones y conocer su utilidad.

1. **assertEquals.** En la clase de prueba CalculadoraTest.php, crear un método testSumar que nos permita verificar si el valor que estamos esperando es el mismo que la función sumar de la clase Calculadora está generando (retornando). Para esto utilizaremos la aserción assertEquals. Según la documentación de PHPUnit, esta aserción se define como sigue: assertEquals(mixed $expected, mixed $actual[, string $message = ''])

Reporta un error identificado por el $message si las dos variables $expected y $actual no son iguales. assertNotEquals() es el inverso de esta aserción y toma los mismos argumentos.

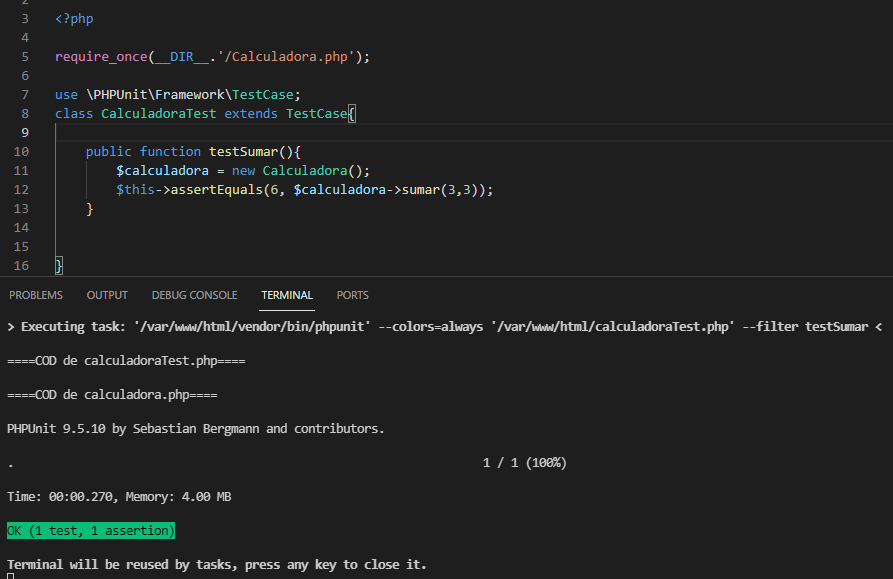
La aserción assertEquals no toma en cuenta el tipo de dato, por lo cual, si comparamos, por ejemplo: 10 con “10”, la aserción se cumple.

Vamos a enviarle un valor correcto y uno incorrecto.

Tome una captura de pantalla para el código desarrollado. Además, indique los valores usados para ambos casos y muestre la salida obtenida:

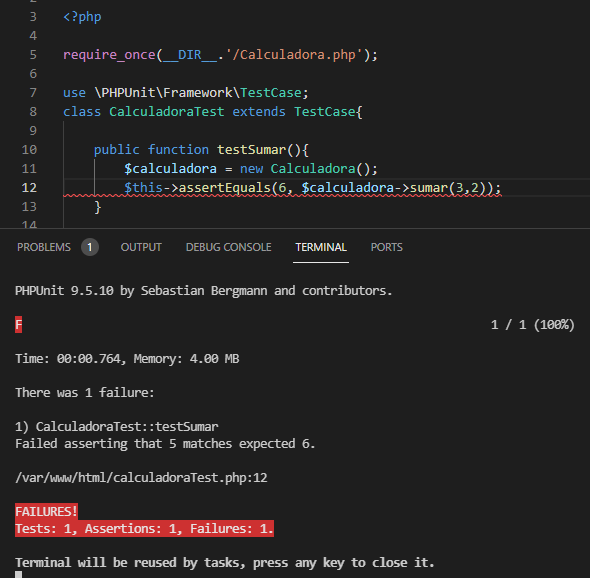
Valores Usado Correcto: 6, sumando 3 y 3

Código y Salida



Valor Usado Incorrecto: 6 sumando 3 y 2

Código y Salida

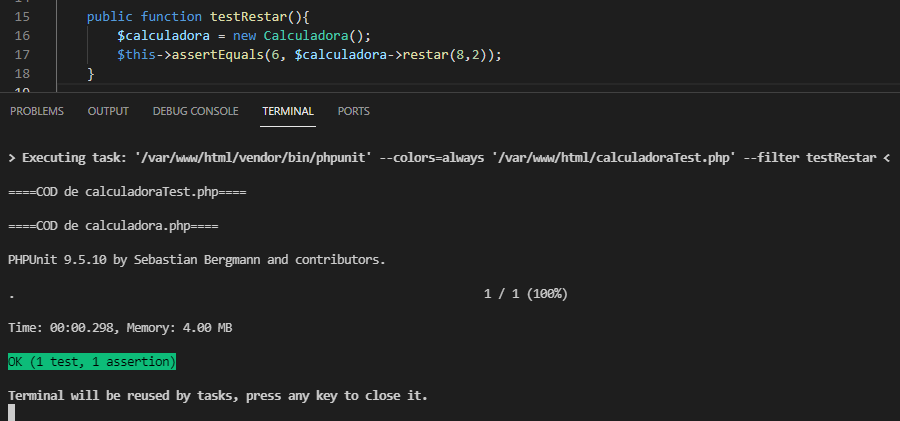


Cree el método de prueba para los métodos restar y multiplicar con esta aserción.

Métodos restar

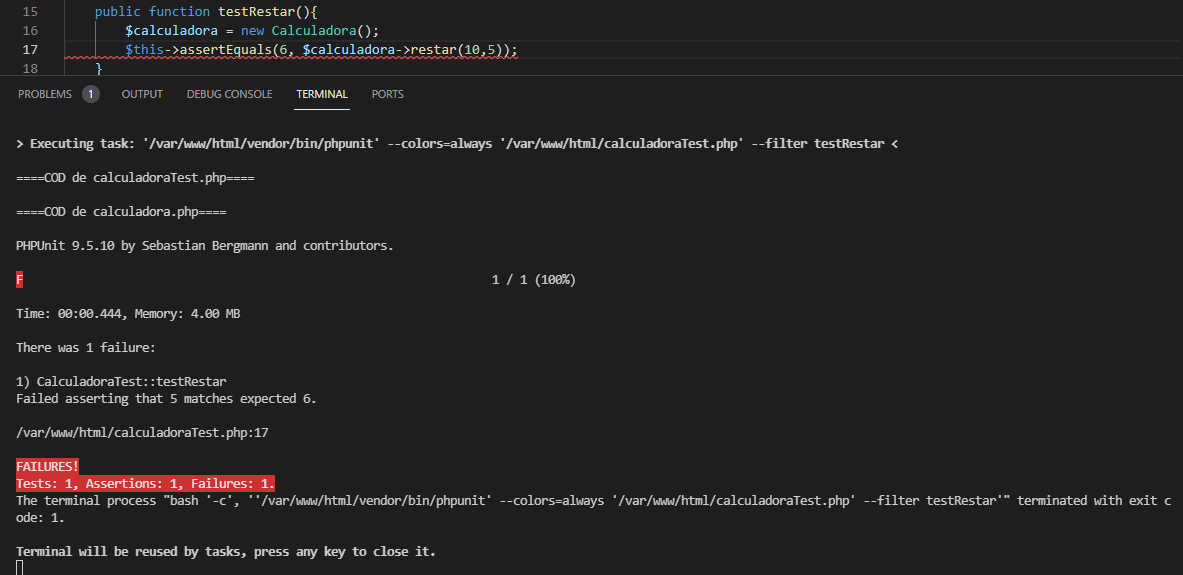
Valor usado correcto: 6, restando 8 y 2

Código y Salida:



Valor Usado Incorrecto: 6, restando 10 y 5

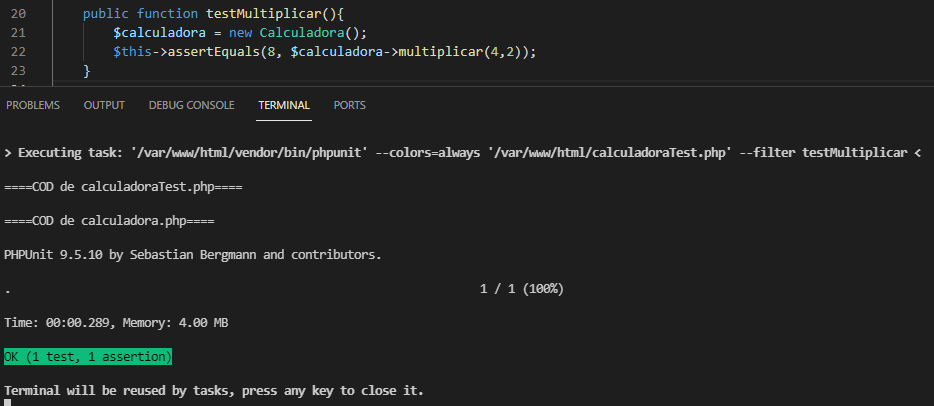
Código Y salida:



Métodos Multiplicación

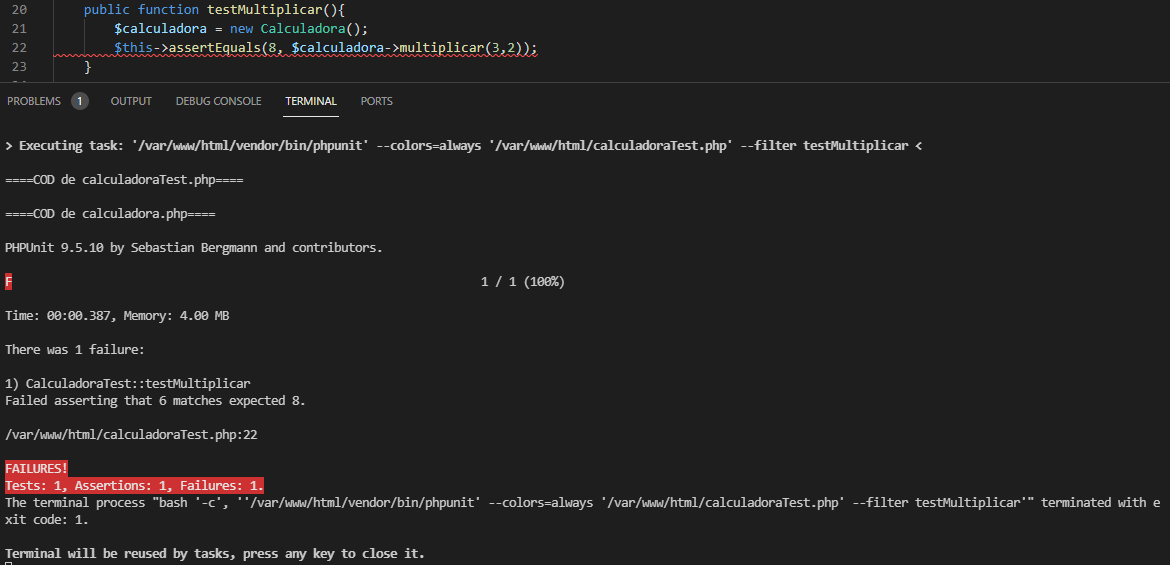
Valores usados correctos: 8, multiplicando 4 y 2

Código y Salida



Valores Usados incorrectos: 8, multiplicando 3 y 2

Código y Salida:



1. **assertEqualsWithDelta.** En la clase de prueba, vamos a crear un método que realice la prueba sobre el método dividir. Para ello usaremos la aserción assertEqualsWithDelta. Según la documentación de PHPUnit, esta aserción se define como sigue:

assertEqualsWithDelta(float $expected, float $actual, float $delta = 0,[string $message = ''])

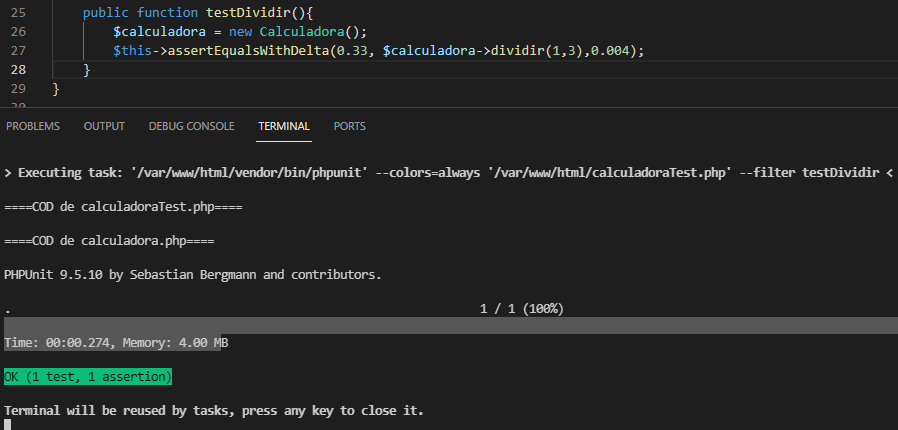
Reporta un error identificado por el $message si la diferencia absoluta entre los números de tipo flotante $expected y $actual es mayor que el $delta. Si la diferencia absoluta entre los números de tipo flotante $expected y $actual es menor o *igual a* $delta entonces la aserción pasa.

Verifiquemos entonces enviando dos números al método dividir con resultado esperado que sea un número de punto flotante (decimal en este caso). Establezcamos un delta que permita que la aserción se cumpla y otro, que no.

Tome una captura de pantalla para el código desarrollado. Además, indique los valores usados para ambos casos y muestre la salida obtenida:

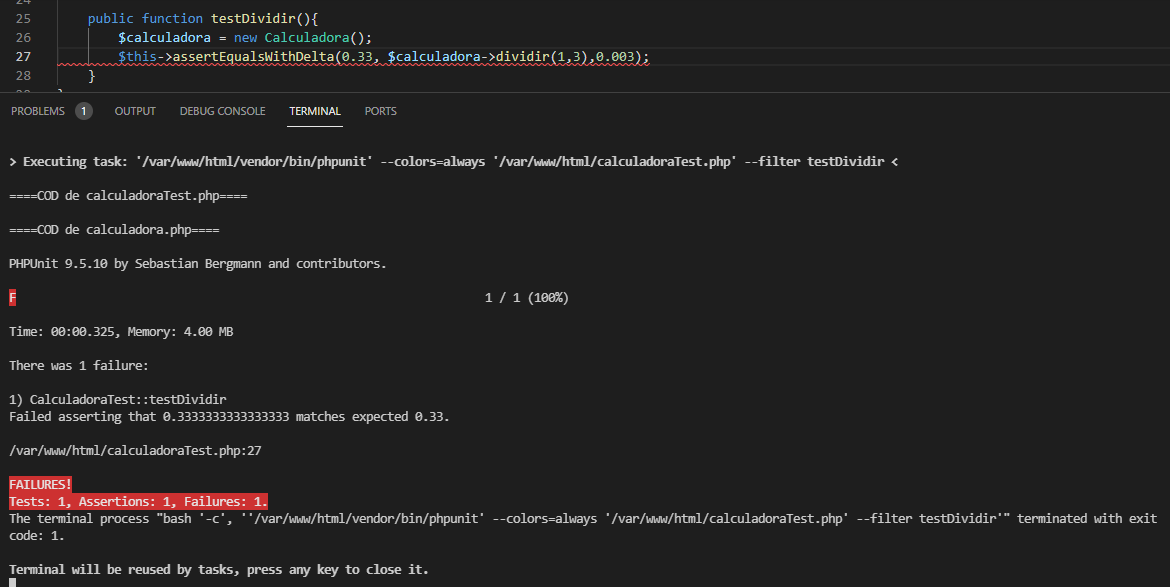
Valores usados cuando la aserción se cumple: 0.33, dividiendo 1 entre 3, con 0.004

Código y Salida:



Valores usados cuando la aserción no se cumple: 0.33, dividiendo 1 entre 3, con 0.003

Código y Salida:



1. **assertSame.** En el método testSumar vamos a agregar la aserción assertSame para verificar que el valor que estamos enviando es el mismo que está generando la función sumar en la clase Calculadora. Según la documentación de PHPUnit, esta aserción se define como sigue:

assertSame(mixed $expected, mixed $actual[, string $message = ''])

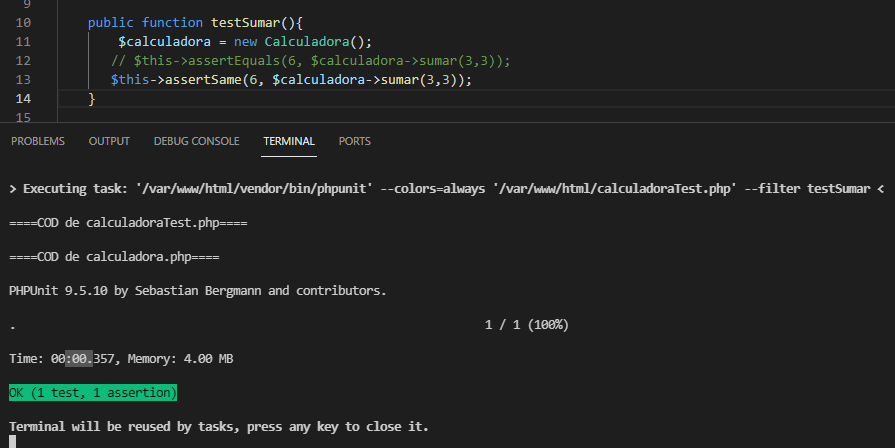
Reporta un error identificado por el $message si las dos variables $expected y $actual no tienen el mismo tipo y valor. assertNotSame() es el inverso de esta aserción y toma los mismos argumentos.

Como vemos, a esta aserción no solo le interesa el valor (como es el caso de assertEquals), también le interesa el tipo de dato. Para verificar esto, establezcamos un valor numérico como salida esperada de la función sumar y luego un valor numérico, pero en texto (usando comillas).

Tome una captura de pantalla para el código desarrollado. Además, indique los valores usados para ambos casos y muestre la salida obtenida:

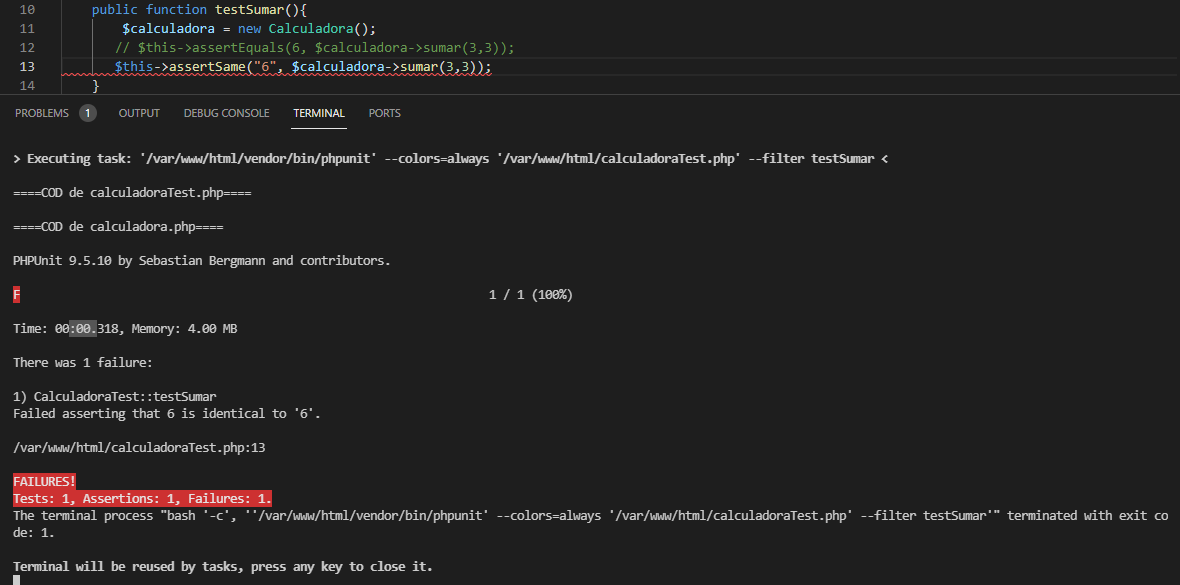
Valores usados para valor numérico correcto: 6, sumando 3 y 3

Código y Salida:



Valores usados para valor numérico incorrecto: “6”, sumando 3 y 3

Código y Salida:



1. **assertContains**. En la clase de prueba, crear un método testGenerarArreglo que nos permita verificar si el arreglo creado en el método generarArreglo (de la clase Calculadora) posee un elemento indicado. Para esto utilizaremos la aserción assertContains. Según la documentación de PHPUnit, esta aserción se define como sigue:

assertContains(mixed $needle, iterable $haystack[, string $message = ''])

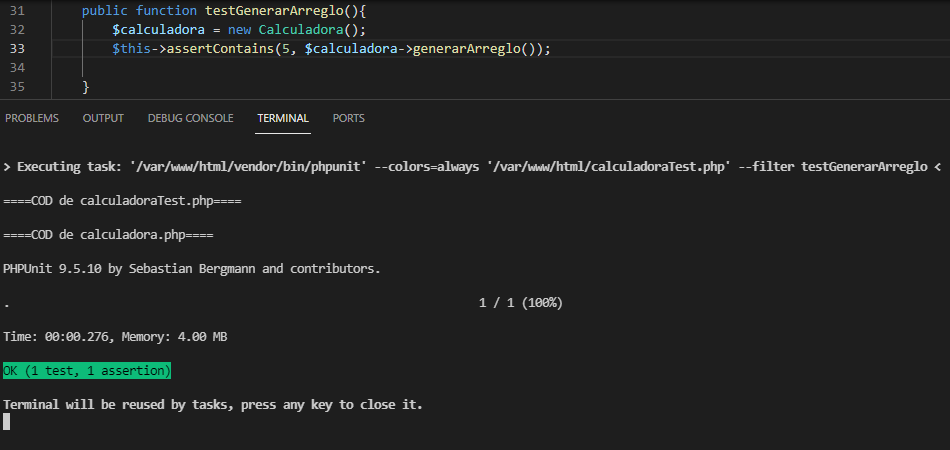
Reporta un error identificado por el $message si $needle no es un elemento de $haystack.

assertNotContains() es el inverso de esta aserción y toma los mismos argumentos.

Vamos a verificar entonces, consultando por un valor entre el 1 y el 10.

Tome una captura de pantalla para el código desarrollado. Además, indique el valor empleado y muestre la salida obtenida:

Valor empleado: 5



1. **assertCount.** En el mismo método de prueba testGenerarArreglo vamos a verificar que la cantidad de elementos del array sea la que estamos esperando. Para esto usaremos la función assertCount. Según la documentación de PHPUnit, esta aserción se define como sigue:

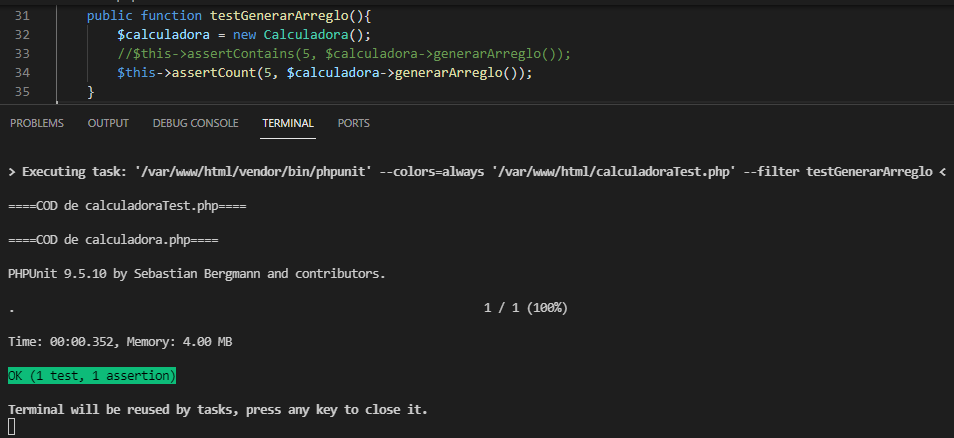
assertCount($expectedCount, $haystack[, string $message = ''])

Reporta un error identificado por el $message si el número de elementos en $haystack no es $expectedCount. assertNotCount() es el inverso de esta aserción y toma los mismos argumentos.

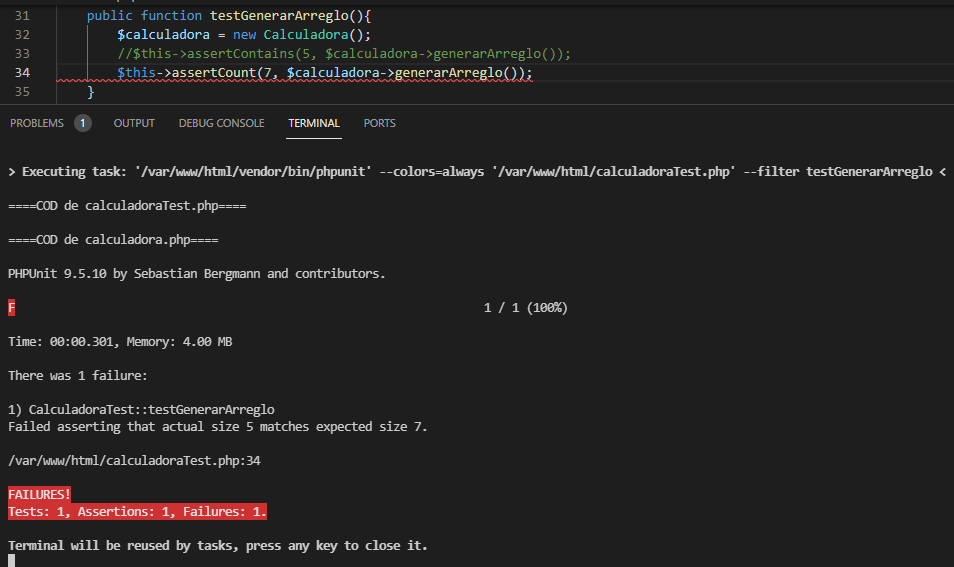
Vamos a verificar entonces, consultando por el tamaño actual del arreglo y por un tamaño incorrecto.

Tome una captura de pantalla para el código desarrollado. Además, indique los valores usados para ambos casos y muestre la salida obtenida:

Valor usado para tamaño correcto: 5



Valor usado para tamaño incorrecto: 7



1. **assertEmpty.** En el mismo método de prueba testGenerarArreglo vamos a verificar que el array devuelto esté vacío. Para esto usaremos la función assertEmpty. Según la documentación de PHPUnit, esta aserción se define como sigue:

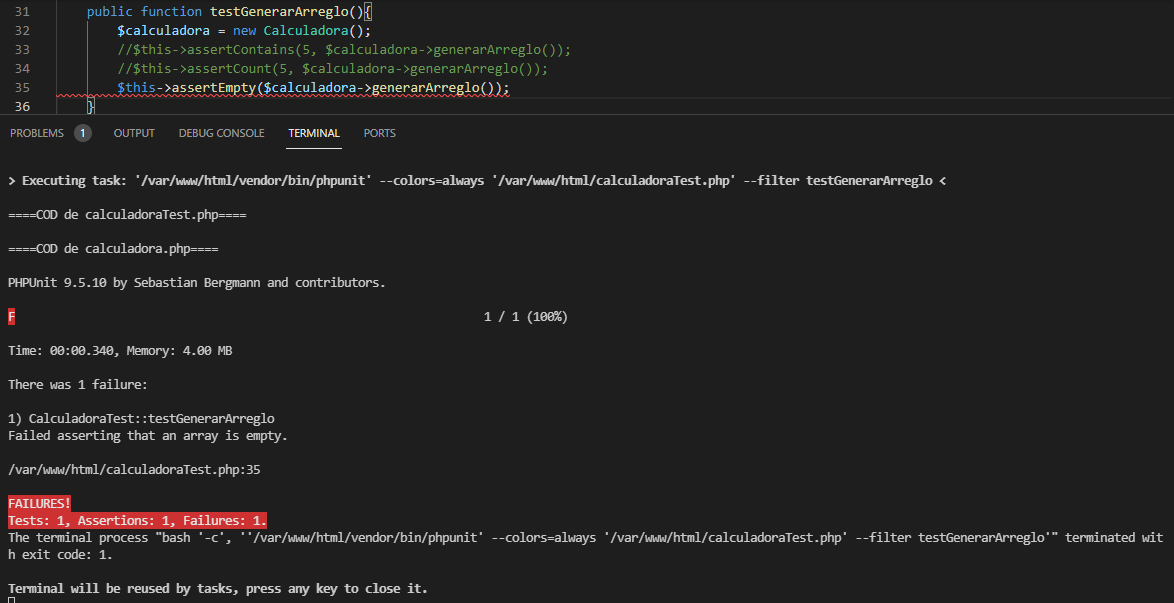
assertEmpty(mixed $actual[, string $message = ''])

Reporta un error identificado por el $message si $actual no está vacío. assertNotEmpty() es el inverso de esta aserción y toma los mismos argumentos.

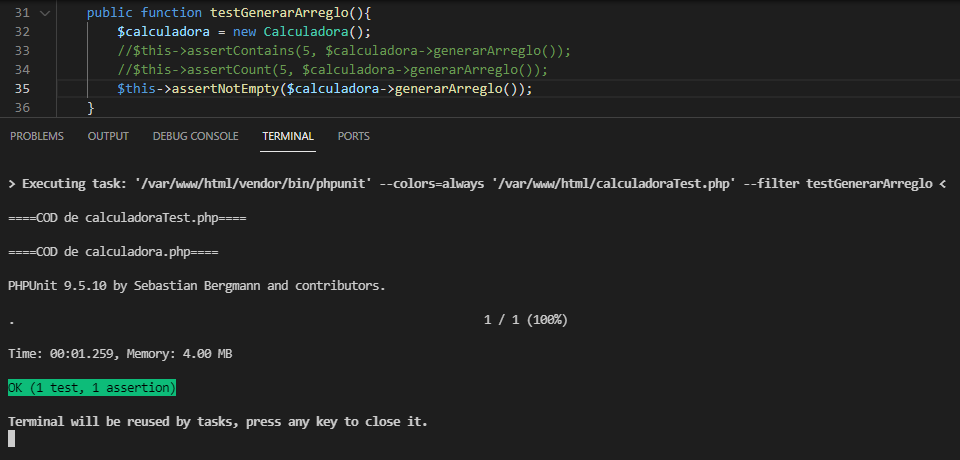
Verifiquemos entonces esperando como resultado que esté vacío y, por otro lado, que no esté vacío.

Tome una captura de pantalla para el código desarrollado y muestre las salidas obtenidas:

Salida cuando el arreglo tiene 5 elementos con AssertEmpty



Salida cuando el arreglo tiene 5 elementos con AssertNotEmpty



**Proveedores de datos**

**Explicación docente:** Un método de prueba puede aceptar argumentos arbitrarios. Estos argumentos son provistos por un método proveedor de datos. El método proveedor de datos que queremos usar se especifica con la anotación @dataProvider. Un método proveedor de datos debe ser público y retornar un arreglo de arreglos o un objeto que implementa la interfaz Iterator que produce un arreglo en cada paso de la iteración. Para cada arreglo que es parte de la colección se llama al método de prueba y el contenido del arreglo constituye sus argumentos.

Empleando la clase Calculadora y la clase de tipo test para esta vamos a crear un proveedor de datos para el método de prueba testSumar().

1. Lo primero que haremos es crear nuestro método proveedor de datos. Para esto creamos el siguiente método:

**public function** sumarProveedor()

{

**return** [

**'Caso 1'**=> [-1, -1, -2],

**'Caso 2'** => [0, 0, 0],

**'Caso 3'** => [0, -1, -1],

**'Caso 4'**=> [-1, 0, -1]

   ];

}

Con esto estamos creando un método que nos va a proveer de los datos de prueba para nuestra prueba unitaria sobre el método sumar. De esta manera, tenemos una sola codificación que toma los diferentes datos de prueba según el diseño por clases de equivalencia y combinación por pares (por ejemplo).

1. Lo siguiente será agregar justo antes de la declaración del método testSumar() la siguiente anotación:

*/\*\**

*\** ***@dataProvider sumarProveedor***

*\*/*

De esta manera, le estamos diciendo al método testSumar() que sus argumentos los tome del proveedor de datos.

1. Luego al método testSumar() le añadimos los tres parámetros que recibirá del proveedor de datos, en el mismo orden como son definidos en éste, y realizamos la aserción con estos parámetros:

*/\*\**

*\** ***@dataProvider sumarProveedor***

*\*/*

**public function** testSumar($numero1, $numero2, $resultado\_esperado)

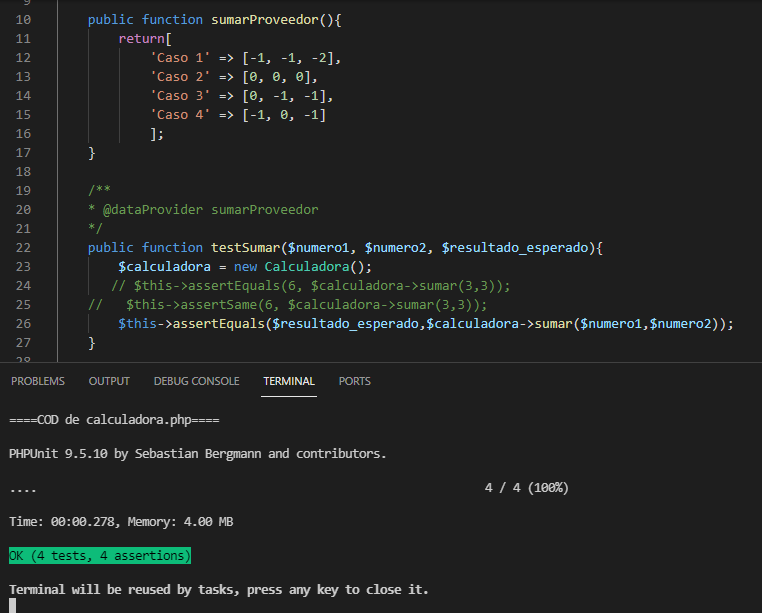
{

$calculadora = **new** Calculadora();

$this->assertEquals($resultado\_esperado,$calculadora->sumar($numero1,$numero2));

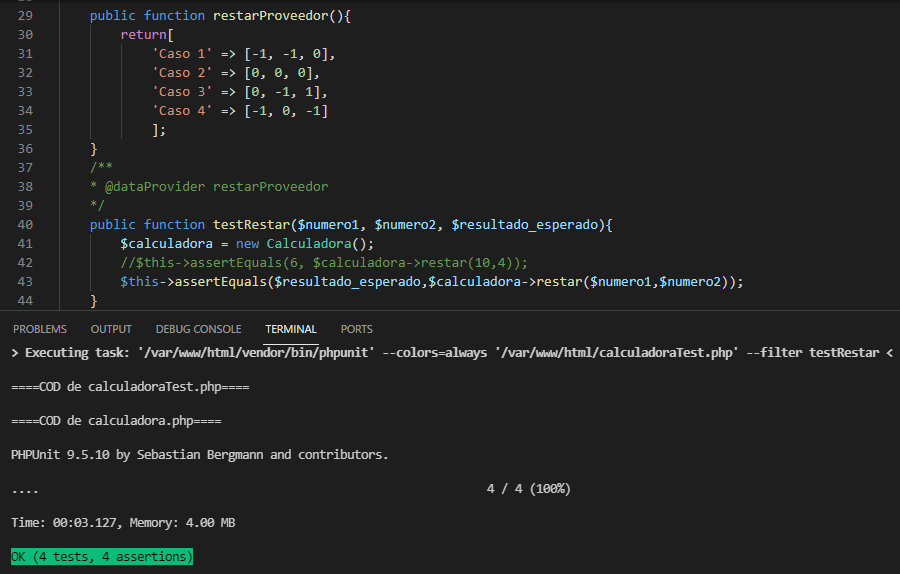
}

1. Corremos nuestro método de prueba testSumar().Tome una captura de pantalla y colóquela aquí.

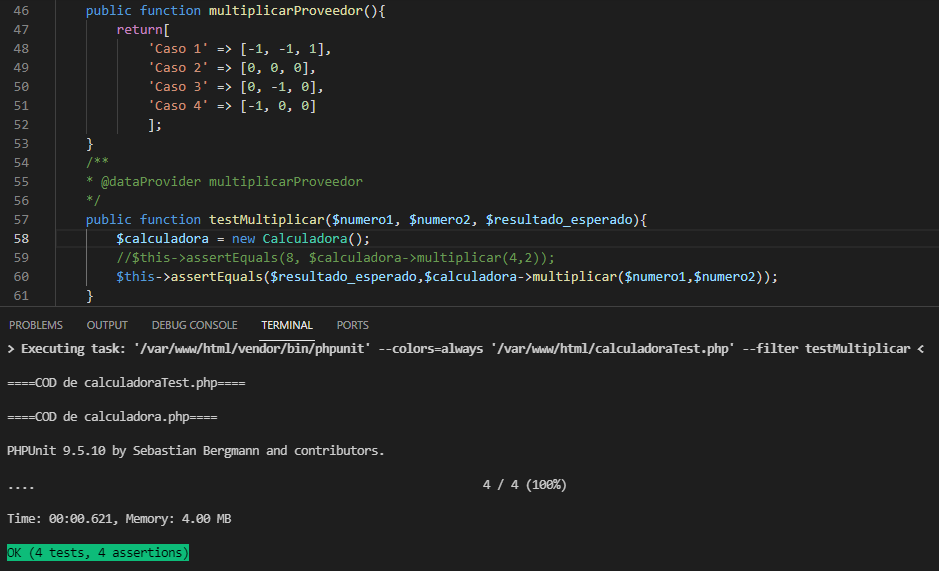


1. Crear un proveedor de datos para los métodos de prueba testRestar y testMultiplicar. Corra las pruebas y coloque las capturas de pantalla con los resultados.

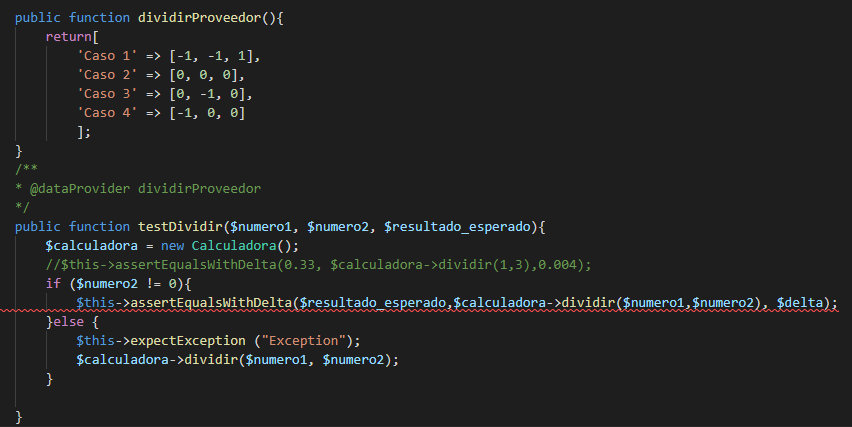
Test restar

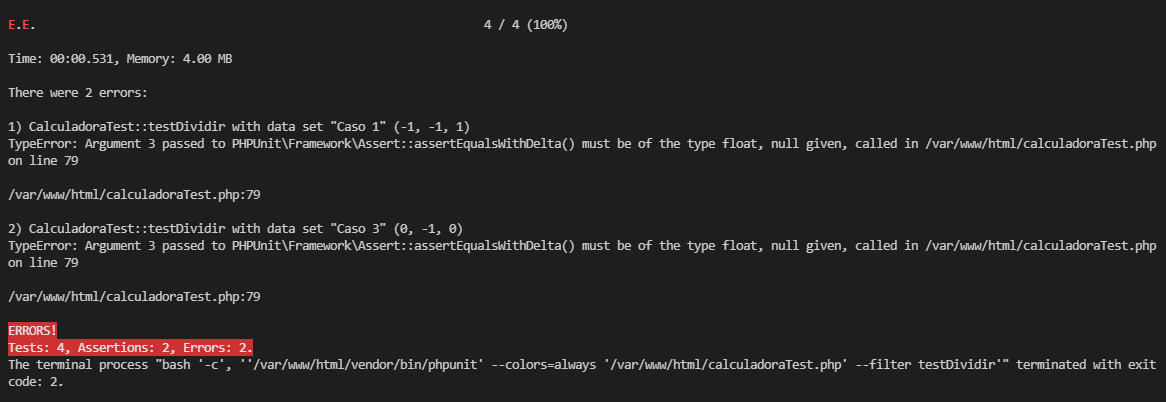


Test Multiplicar



1. Crear un proveedor de datos para el método testDividir contemplando el valor de delta, así como el manejo de la excepción cuando se intenta dividir por cero. Corra las pruebas y coloque las capturas de pantalla con los resultados.





1. **RECURSOS:** Computadora con: VSCode, XAMPP o un Docker con PHP, PHPUnit.
2. **RESULTADOS:** Proyecto PHP con el desarrollo realizado, laboratorio resuelto.