

# Certificado de profesionalidad IFCD0210

# Pruebas

Es muy difícil realizar una aplicación completa sin errores.

Es necesario realizar pruebas para poder localizar errores.

Ojo con ampliaciones futuras que provocan errores en funciones correctas del pasado

El propósito de pruebas automatizadas es asegurar que el funcionamiento actual y el deseado es correcto a lo largo del tiempo.

# Pruebas

Existen varios tipos de pruebas, cada una dirigida a un aspecto específico de la aplicación:

- ✓ **Pruebas unitarias:** tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado.
- ✓ **Pruebas de integración:** el objetivo es verificar el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes.

# Pruebas

Existen varios tipos de pruebas, cada una dirigida a un aspecto específico de la aplicación:

- ✓ **Pruebas de rendimiento:** es una técnica que determina cómo la estabilidad, la velocidad, la escalabilidad y la capacidad de respuesta de una aplicación se mantiene bajo una determinada carga de trabajo.
- ✓ **Pruebas de usabilidad:** técnica usada en el diseño de interacciones centrado en el usuario para evaluar un producto mediante pruebas con los usuarios mismos.
- ✓ **Pruebas de aceptación:** valida que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.

# Test unitarios

Los test unitarios son una batería de test, es decir, fragmentos de código ejecutable que prueban automáticamente la validez de partes o unidades del software.

Comprobamos los componentes de forma individual, a nivel de cada método de la clase, permitiéndonos aislar de forma eficiente el comportamiento erróneo de la aplicación.

En programación, una **prueba unitaria** es una forma de comprobar el correcto funcionamiento de una unidad de código.

# Test unitarios

```
<?php
$directores = [];
echo count($directores); // 0
echo "<br>";
$directores[] = "Martin Scorsese";
echo count($directores); // 1
?>
```

EL RESULTADO NECESITA  
INTERPRETACIÓN

```
<?php
$directores = [];
echo (count($directores) == 0)? "ok":"not ok"; // ok
echo "<br>";
$directores[] = "Martin Scorsese";
echo (count($directores) == 1)? "ok":"not ok"; // ok
?>
```

EL RESULTADO **NO** NECESITA  
INTERPRETACIÓN

# PHPUnit

PHPUnit es el framework para implementar las pruebas unitarias en lenguaje PHP.

PHPUnit se ejecuta desde una consola o Shell

Llamamos “casos de prueba” a las clases que contienen la lógica necesaria para probar otras clases.

Habitualmente tendremos una carpeta /tests que tendrá la misma distribución de carpetas y archivos que nuestro proyecto. La diferencia es que estos serán los casos de prueba que utilizaremos para probar las clases del proyecto

# PHPUnit

Se siguen las siguientes convenciones:

- ✓ Los test de una clase llamada **NombreClass** se incluyen en una clase llamada **NombreClassTest**
- ✓ En la mayoría de ocasiones, esta clase **NombreClassTest** hereda de **PHPUnit\Framework\TestCase**
- ✓ Los test se definen como métodos públicos sin parámetros y con la cadena test como prefijo en su nombre
- ✓ En los test, se usan asserts, como **assertEquals()**, para comprobar si el valor esperado es igual (en este caso).
- ✓ Los test se lanzan por línea de comandos.



# PHPUnit

Acceder a la Shell del servidor mediante el terminal de MAC:

```
ssh nascorXX@nascorXX.md360.es -p 50050
```

```
Last login: Sun Aug 29 18:29:50 on ttys000
iMac-de-Borja:~ borja$ ssh nascor16@nascor16.md360.es -p 50050
The authenticity of host '[nascor16.md360.es]:50050 ([185.23.118.163]:50050)' can't be established.
ECDSA key fingerprint is SHA256:0WdXbLIJAo+0m80Jl7YBK4lVWlCXRXIr2bcq5rsNRdo.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[nascor16.md360.es]:50050,[185.23.118.163]:50050' (ECDSA) to the list of known hosts.
nascor16@nascor16.md360.es's password:
Linux dns118163.phdns25.es 5.9.0-0.bpo.5-cloud-amd64 #1 SMP Debian 5.9.15-1~bpo10+1 (2020-12-31) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Aug 18 17:35:58 2021 from 37.18.134.208
nascor16@dns118163:~$
nascor16@dns118163:~$
```

Para windows

<https://docs.ovh.com/es/dedicated/introduccion-ssh/>

# PHPUnit

nascor16.md360.es
● Activos

Sitio web en [httpdocs/](#) Dirección IP: 185.23.118.163 Usuario del sistema: nascor16

[Configuración de hosting](#)
[Abrir en web](#)
[Descripción](#)
[Transferir dominio](#)

Empiece a crear su sitio web mediante una de las siguientes formas:

WordPress  
Cree su sitio web con WordPress.  

Instalar WordPress

Instalar una aplicación  
Cree su sitio instalando una aplicación web como Joomla o Drupal.  

Instalar aplicaciones

Crear un sitio web personalizado  
Cargue su contenido web y añada bases de datos.  

Archivos

Bases de datos

Mostrar menos

Acceso a hosting web

Cambie la configuración de la cuenta del usuario del sistema usada para acceder a Plesk de forma remota a través de SSH o RDP y para utilizar archivos y carpetas en el administrador de archivos.

Acceso FTP

Certificados SSL/TLS  
Webmail no protegido

Configuración de hosting

Configuración de PHP (v7.3.29)

Administrador de archivos

Cuentas de correo

Firewall para aplicaciones web

Registros

Estadísticas web SSL/TLS

Configuración de correo

Directorios protegidos con contraseña

Usuarios web

Aplicaciones

Configuración DNS

Limitar correos salientes

Copia de sitio web

# PHPUnit

[Inicio](#) > [Dominios](#) >

## Acceso a hosting web para **nascor16.md360.es** ...

Aquí puede ver las direcciones IP asociadas con su suscripción y editar el nombre de usuario y la contraseña de su usuario del sistema.

### Direcciones IP

Dirección IP

185.23.118.163 (compartida) ▾

La dirección IP donde se aloja el sitio web es una dirección de red del host virtual del sitio web.

### Usuario del sistema

Aquí puede especificar un usuario del sistema operativo para la gestión de los archivos y carpetas de la suscripción mediante FTP o a través de también se utiliza para acceder a Plesk mediante SSH (en Linux) o RDP (en Windows) si se conceden los permisos correspondientes.

Nombre de usuario \*

nascor16

Contraseña

Generar

Mostrar

Confirmar contraseña

Acceder al servidor vía SSH

- Forbidden
- /bin/sh
- ✓ /bin/bash
- /usr/bin/bash
- /bin/rbash
- /usr/bin/rbash
- /bin/dash
- /usr/bin/dash
- /bin/bash (chrooted)

SSH usando las credenciales del usuario del sistema.

### Cuota máxima de espacio

La cuota máxima de espacio en disco que puede usarse. Si establece la cuota máxima y se usa todo el espacio en disco, no se podrán realizar más acciones en archivos.

# PHPUnit: Composer

**Composer** es un sistema de gestión de paquetes para programar en PHP el cual provee los formatos estándar necesarios para manejar dependencias y librerías de PHP.

DEDE ESTAR INSTALADO EN LA **CARPETA DEL PROYECTO**

**ACCESO SSH**

<https://es.wikipedia.org/wiki/Composer>

# PHPUnit: Instalación

## PASO 1

Creamos una carpeta del proyecto: proyecto

## PASO 2

Creamos las carpetas src y test dentro de proyecto

## PASO 3

Creamos el archivo composer.json en la raíz del proyecto

```
{
    "autoload": {
        "classmap": [
            "src/"
        ]
    },
    "require-dev": {
        "phpunit/phpunit": "^9"
    }
}
```

# PHPUnit: Instalación

## PASO 4

Instalamos composer

`composer install`

## PASO 5

Preparamos el uso de la carga automática

`composer dump-autoload`

## PASO 6

Ejecutamos las pruebas test

`./vendor/bin/phpunit --colors test`

# PHPUnit: Patrón AAA

## **ARRANGE (Preparar)**

Se inicializa el test unitario.

## **ACT (Actuar)**

Lanzamos la acción que queremos probar.

## **ASSERT (Afirmar)**

Comprobación que la acción tiene el resultado deseado.

<https://phpunit.readthedocs.io/es/latest/assertions.html>

# PHPUnit: Fixtures

## **SetUp()**

Antes de la ejecución de cada método de test se llama a este método.  
Inicializamos los objetos para las pruebas.

## PREPARACIÓN

## **tearDown() (Limpieza)**

Cuando termina la ejecución del método de test se invoca esta función que  
nos permite limpiar el sistema de los objetos creados.

## **ASSERT (Afirmar)**

Comprobación que la acción tiene el resultado deseado.

<https://phpunit.readthedocs.io/es/latest/assertions.html>

## VER EJEMPLO1

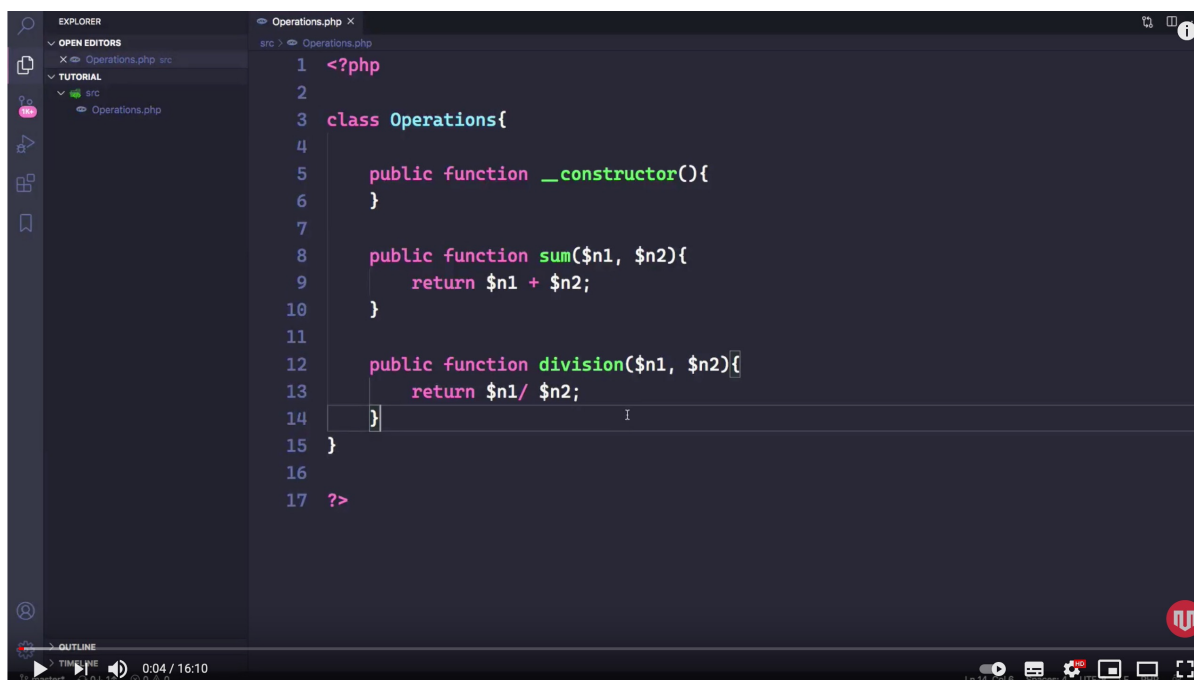


# PHPUnit: Actividad 1

Visualizar vídeo

<https://www.youtube.com/watch?v=khhz8RAoSww>

Reproducir el ejemplo en vuestro servidor



```
1 <?php
2
3 class Operations{
4
5     public function __constructor(){
6     }
7
8     public function sum($n1, $n2){
9         return $n1 + $n2;
10    }
11
12    public function division($n1, $n2){
13        return $n1/ $n2;
14    }
15 }
16
17 ?>
```

# PHPUnit: Unit Test are FIRST

## FIRST

Fast (Rápido)

Isolated (Independiente)

Repeateable (Repetible)

Selfverifying (Autoverificable)

Timely (Oportuno)

## Característica principal de TDD

### Test Driven Development

Sistema de realizar pruebas unitarias donde se crean los test antes que el código

# PHPUnit: Unit Test are FIRST

## **FAST (Rápido)**

Gran número de test

Los test han de ser rápidos

Si tardan mucho en ejecutarse estamos utilizando mal la herramienta.

## **ISOLATED (Independiente)**

Cada test debe tener una sola razón para fallar

Todos los test deben ser independientes entre si

Los test deben ser independientes de factores externos

# PHPUnit: Unit Test are FIRST

## **REPEATABLE (Repetible)**

Los test deben ofrecer siempre el mismo resultado.

Eliminar los datos de la memoria (tearDown)

Volatibilidad de recursos externos

Dependencia de clases no inicializadas

## **SELFVERIFYING (Autoverificable)**

O Falla o Éxito. No es posible ambigüedad.

## **TIMELY (Oportuno)**

Si escribimos el test antes que el código,

determinamos lo que queremos que haga nuestra función.

Determinamos los resultados de los ejemplos y crearemos el código necesario para que funcionen nuestro test antes.

# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

El objetivo de TDD es código limpio que funcione ya que como sabemos el resultado de las pruebas, sabemos cuando hemos terminado de implementar.

- ✓ Implementación de las funciones que el cliente necesita.
- ✓ La minimización de los errores/defectos que llegan a producción
- ✓ Producción de software modular, altamente reutilizables y preparado para el cambio

# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

Reglas básicas:

- ✓ Se escribe nuevo código solo si un test ha fallado
- ✓ Eliminamos el código duplicado

Tareas de programación:

### **ROJO**

Escriba un test que en principio no va a funcionar

### **VERDE**

Corrija el test para que se ponga en verde

### **REFACTORICE**

Optimice el código y elimine el duplicado

# PHPUnit: TDD

**Test-Driven Development**  
**(Desarrollo guiado por pruebas)**

<https://www.youtube.com/watch?v=YuRdaR6wwWU&t=381s>



# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

### VENTAJAS

- ✓ La calidad del software aumenta
- ✓ Conseguimos código altamente reutilizable
- ✓ El trabajo en equipo es más fácil
- ✓ Mejora la comunicación entre los miembros del equipo.
- ✓ Papel del encargado de la calidad
- ✓ Hacer primero el test obliga a escribir un mínimo de funcionalidad.
- ✓ El test es la mejor documentación
- ✓ Incrementa la productividad
- ✓ Nos hace afrontar más casos de uso en tiempo de diseño
- ✓ Sensación de trabajo bien hecho



# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

### EJEMPLO

1. Necesitamos una clase a la que se le pasen dos cadenas de caracteres y devuelva la longitud de la más larga
2. Si uno de los parámetros no es una cadena de texto, devuelve la longitud del que sí lo es
3. Si ninguno de los dos parámetros es un string, devuelve false

# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

Debemos trabajar un análisis inicial y determinar una lista de características que amplíen la lista inicial

La idea es obtener todos los casos de prueba necesarios

Podemos plantearnos algunas preguntas



# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

- a. ¿Necesitamos crear un objeto? ¿De que clase?
- b. ¿Tenemos que llamar a algún método de ese objeto? ¿Ese método tiene parámetros? ¿opcionales u obligatorios?
- c. ¿Qué pasa si no le pasamos ningún parámetro?
- d. ¿Qué sucede si le pasamos un único parámetro de tipo distinto a string? ¿y si es de tipo string?
- e. ¿Y si pasamos los dos parámetros con tipos diferentes a string? ¿y si sólo lo es uno de los dos?
- f. Si pasamos los dos parámetros de tipo string, ¿devolverá la longitud correcta?
- g. ¿y si tienen la misma longitud?

# PHPUnit: TDD

## Test-Driven Development (Desarrollo guiado por pruebas)

1. Necesitamos crear un objeto de una clase que llamaremos StringManager. (a)
2. Llamaremos a un método de este objeto que será el que me devuelva la longitud de la cadena de caracteres más larga de las dos que le pasaremos como parámetros. (b)
3. Si no le pasamos ningún parámetro, devuelve false (c)
4. Si le pasamos un único parámetro de tipo distinto a string, devuelve false (d)
5. Si le pasamos un único parámetro de tipo string, devuelve su longitud (d)
6. Si le pasamos los dos parámetros de tipo distinto a string, devuelve false (e)
7. Si le pasamos uno de los dos parámetros de tipo distinto a string y otro de tipo string, devuelve la longitud del tipo string, independientemente de su orden. (f)
8. Pasarle dos cadenas de igual longitud y comprobar que devuelve esa longitud (g)
9. Pasarle dos cadenas de diferente longitud y comprobar que devuelve la longitud de la más larga. (g)

# PHPUnit: TDD

**Test-Driven Development  
(Desarrollo guiado por pruebas)**

EJEMPLO 2

## ACTIVIDAD 2

Siguiendo el mismo modelo de desarrollo en TDD, pensar en especificar una pequeña función que a partir del dni nos saque la letra.

Ver ejercicio del módulo anterior.

### FASE 1

Preparar las preguntas adecuadas y responderlas

### FASE 2

Trabajar el test y el desarrollo de la funcionalidad

# Certificado de profesionalidad IFCD0210