# AUT04\_06 Pruebas unitarias con código

# Objetivo.

Esta actividad consiste en generar un informe con el nombre **AUT04\_06\_Apellido1Apellido2Nombre\_Apellido1Apellido2Nombre.pdf**

con el plan de pruebas y el código de C# con las pruebas unitarias.

# Descripción.

Como no estás contento con tu desempeño educativo en la asignatura de programación (¡el profesor no te ha puesto ese diez que tanto anhelabas!) has decidido tomar cartas en el asunto y vas a proceder a hacer un ***test plan (plan de pruebas)*** sobre un proyecto:

1. Dos funciones que trabajan con ficheros.

***Con la diferencia de que no va a ser tu código, sino el de un compañero tuyo.***

Para hacer el test plan, podéis seguir la metodología usada en [este](https://www.linkedin.com/pulse/cómo-crear-un-test-plan-qué-hacer-y-juan-pablo-aguirre/?originalSubdomain=es) enlace y no tener que hacer un documento extenso.

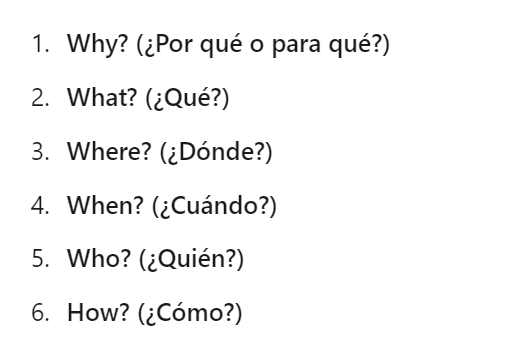
Además del plan de pruebas, tenéis que planificar las pruebas, generando casos de pruebas que después tenéis que implementar usando MSTest.

Para planificar las pruebas hay que usar los dos métodos vistos en clase, las técnicas de prueba negra y prueba blanca.

* Tened en cuenta que durante el testing se buscan todos los casos de prueba que garanticen el funcionamiento correcto, cubriendo todas las posibles ejecuciones y estados que tiene nuestro programa. Se valorará el detalle y la precisión, cuantos más casos de prueba y mejor detallados estén, más seguro estaremos tanto el desarrollo como el cliente. No probéis sólo los casos extremos de un rango de valores, tener casos de prueba que trabajen con valores en la parte intermedia de un rango de valores, representando un caso usual es igual de importante.
* A la hora de implementar los casos de prueba en las pruebas unitarias podéis seguir [esta](https://gist.github.com/pmcilreavy/c1a4913544716617763d97dbd9f7a5c5) plantilla de MSTest, que incluye algunas etiquetas que no hemos visto, pero ofrece una estructura general para vuestro proyecto.
* Aunque no es lo común, podéis usar más de un Assert por test. En ese caso, y para separar las diferentes ejecuciones, podéis usar un parámetro de entrada opcional reservado para mostrar mensajes cuando el resultado del Assert sea falso. Por ejemplo, podéis lanzar ***Assert.IsTrue(x > 34, “x tendría que ser mayor que 34 y ha devuelto” + x);***

El segundo parámetro, completamente opcional, nos servirá para poder diferenciar aquellas partes de nuestro test unitario.

* El plan de pruebas debe ser un documento pdf donde se establezcan las preguntas establecidas:



* Se adjunta junto al enunciado un fichero .csv con información de 721 Pokémon. En base a este fichero, hay que crear estas dos funciones:
  + ***Public static String strongestPokemon(string rutaFichero, int generation)***, que devolverá el nombre del pokémon con la generación igual a la pasada como parámetro que más ataque tenga. No tiene que ser legendario.
  + ***Public void filterPokemon(string rutaFichero)***, que creará un nuevo fichero con toda la información de los Pokémon que tengan exactamente dos tipos. El programa no devolverá nada, pero creará un fichero de nombre “pokemonDosTipos.csv” en la ruta por defecto establecida en el IDE.

# No olvides ¿Qué debemos hacer?

Hay que entregar el documento PDF con el plan de pruebas y el código de C# con las pruebas unitarias.