> CODIGO LIMPIO

Se considera que un código es limpio (en inglés, CLEAN CODE) cuando es fácil de leer y entender; si resuelve problemas evitando agregar complejidad innecesaria, permitiendo que el mantenimiento o adaptaciones, por algún cambio de requerimiento, sean tareas más sencillas.

Para crear código limpio hay que conocer y poner en práctica un conjunto de principios o técnicas de desarrollo que nos ayudarán a evitar los code smells, es decir, esos síntomas de un programa que te dan el indicio de que existe un problema más profundo.

Nombres con Sentido

Los nombres están en todas partes, clases, variables, ficheros, etc. Elegir nombres relevantes es de suma importancia, y aunque pareciera algo de sentido común, ni es tan común ni es tan sencillo.

```
Cada vez que encuentre un nombre de alguna variable que se puede mejorar, CAMBIALA!, la gente que lea su código sé lo agradecerá. Los nombres deben revelar nuestras intenciones.
```

Funciones

Una función es una sección de un programa que calcula un valor de manera independiente al resto del programa.

```
El tamaño de nuestra función debe ser reducido para facilitar su comprensión.
```

El objetivo es que el código se pueda leer de arriba hacia abajo, es lo que se denomina la regla descendente. Que en cada función que vamos bajando se baje en 1 el nivel de abstracción, así quedaría un solo nivel de abstracción en cada función, es algo complicado de entender, pero es la clave para escribir pequeñas funciones.
El número de argumentos ideal para una función es cero, luego uno y después dos. Siempre que sea posible, hay que evitar la presencia de tres argumentos.
No es recomendable que las funciones tengan efectos secundarios. Las funciones deben hacer algo o devolver algo, pero no ambas cosas, sería confuso.

Comentarios

Un comentario bueno puede resultar muy útil, pero un comentario antiguo y que diga mentiras es un error, los comentarios se suelen usar cuando somos incapaces de expresarnos en el código, pero lo ideal es dedicar tiempo a podernos expresar correctamente en el código, porque un comentario antiguo que no haya sido actualizado puede crear confusión, debido a que el código probablemente se haya modificado con el tiempo.

```
Aunque los comentarios  sean necesarios en ocasiones, debemos dedicar nuestra energía a minimizarlos.
```

Una de las razones por las que se escriben comentarios es el código incorrecto, creamos una función o una clase, vemos que es un poco confuso y lo comentamos, mejor dedica ese tiempo a crear una función o clase con un buen código, con código limpio.

Formatos

Cuando los usuarios ven nuestro código queremos que se asombren de lo bien hecho que está, que parezca un trabajo hecho por profesionales, y no solo una masa amorfa y desordenada de código

```
Quizá, como desarrollador, lo más importante es que el código funcione, pero lo es más que se lea bien
```

Esto debido a que la funcionalidad puede cambiar en la siguiente versión, pero la buena o mala legibilidad del código seguirá ahí.

Procesar Errores

Somos humanos, por lo tanto, cometemos errores y claramente la programación no es la excepción. Los errores en la programación responden a diferentes tipos, como lo puede ser un error en la argumentación o un error de controladores.

```
El control de errores es una parte importante al programar, puesto que las entradas pueden ser incorrectas y los dispositivos pueden fallar.
```

El control de errores es importante, pero si oscurece la lógica, es incorrecto.
Es recomendable usar excepciones en lugar de códigos devueltos, en el pasado los lenguajes carecían de excepciones y los errores se comprobaban mediante códigos de error, el problema de esto es que había que comprobar el código mediante condicionales y oscurecía la lógica, con las excepciones todo es más sencillo y no hace falta hacer comprobaciones.

```
Es importante ofrecer contexto en las excepciones, que nos indique en que caso y a que se debe dicha excepción, para ello es aconsejable redactar mensajes de error informativos y pasarlos junto a las excepciones
```

Pruebas de Unidad

Las tres leyes del desarrollo guiado por pruebas son las siguientes:

```
1. No debe crear código de producción hasta que haya creado una prueba para ello.

2. No debe crear más de una prueba que falle, el no compilar se considera un fallo.

3. No debe crear más código de producción del necesario para superar dicha prueba
```

Hay que tener en cuenta también que, aunque existan cosas que jamás usaríamos en un entorno de producción, si las podríamos usar en un entorno de pruebas, cada entorno tiene sus propias necesidades.

Clases

```
Una clase debe comenzar con el listado de variables y luego con las funciones públicas, pero las funciones que se estén llamando deben colocarse antes.
```

Es importante que nuestras variables y funciones sean privadas, pero no imprescindible, podemos hacerlas protected para que sean accesibles desde una prueba.

```
Las clases deben ser de tamaño justo y reducido
```

En las funciones nos fijamos en el número de líneas, pero en las clases la medida serán las responsabilidades; las clases también deben tener un número reducido de variables de instancia.
Una clase o un módulo solo debe tener un único motivo para cambiar, si una clase tiene varios motivos para cambiar, es que tiene demasiadas dependencias, una manera de solucionarlo es extraer esos métodos en clases pequeñas que se encarguen únicamente de eso.

Sistemas

No es lo mismo la construcción que el uso que se le va a dar al sistema. Son procesos totalmente diferentes, cuando construyen un hotel tenemos grúas y obreros, pero cuando este acabado no habrá grúas y los trabajadores serán diferentes.

```
Los sistemas de software  deben separar el proceso de inicio, de la lógica de ejecución que toma el testigo tras él inicio.
```

El proceso de inicio es un aspecto que toda aplicación debe abordar. La separación de aspectos es una de las técnicas de diseño más antiguas e importantes de nuestra profesión. Una forma de separar la construcción del uso consiste en trasladar todos los aspectos de la construcción a main o módulos involucrados por main. La función principal crea los objetos y se los pasa a la aplicación para que los utilice.

Emergencia

La calidad es lo más importante en nuestro trabajo, de nada nos sirve que el código se ve complicado e impresionante si no cumple con todas las funciones requeridas, por eso es importante que:

Ejecuta todas las pruebas.
Se puede tener un buen diseño, pero si no existe una forma sencilla de probar el código, el esfuerzo sobre el papel es cuestionable, y un sistema que no se puede verificar no debe implementarse.

```
Crear códigos testables hace que diseñemos clases de tamaño reducido y un solo cometido, lo que nos conduce a obtener mejores diseños.
```

No contiene duplicados.
Dentro del código podrían existir varias funciones que realizan el mismo proceso, pero trabajan con diferentes variables. Esto puede ser corregido para darle una mejor presentación y un mayor profesionalismo a nuestro código.

Expresa la intención del programador.
El código funciona de maravilla, pero otro colega programador necesita realizar algunos cambios para cumplir funciones más especificas de un proyecto, pero, al momento de entrar se topa con muchas variables y funciones con nombres a los cuales no les encuentra sentido ni propósito, por lo que no entiende la intención de cada apartado.

```
Por esto mismo es muy importante manejar estructuras y nombres de variables claros para así poder transmitir fácilmente la intención de cada función
```

Minimiza el número de clases y métodos.
El dividir funciones de nuestro código en una gran práctica, sin embargo, es importante llegar al punto donde sobredividirlo puesto que se perdería la optimización inicial.

Concurrencia

La concurrencia es la capacidad de diferentes partes o unidades de un programa para ejecutarse fuera de orden o en orden parcial, sin afectar el resultado.

```
Es importante limitar el ámbito de concurrencia en datos, ya que dos procesos que modifican el mismo campo pueden interferir entre ellos.
```

Una forma de evitar datos compartidos es no compartirlos. En algunos casos se pueden copiar objetos y procesarlos como solo lectura, o copiar, recopilar los resultados y combinarlos en un resultado en mismo proceso. Si existe una forma sencilla de evitar los objetos compartidos, el código resultante tendrá menos problemas