

## Post Destacado

## Configurando la Impresión Perfecta de un Libro de Excel

escrito por Federico Garay

Aunque vivimos en una era digital, a veces todavía es necesario imprimir datos, tablas, gráficos, o información de cualquier tipo que hayamos procesado en Excel. Especialmente si estamos hablando de informes financieros, gráficos estadísticos, ...

## Cohesión - Pilares de la Programación Orientada a Objetos en Python

escrito por Federico Garay



La **cohesión** se refiere al grado de **relación entre los elementos de un módulo**. Cuando diseñamos una función, debemos identificar de un modo bien específico qué tarea va a realizar, reduciendo su finalidad a un objetivo único y bien definido.

En resumen: para que una función sea cohesiva debe hacer **solo una cosa**, y si tiene que hacer más de una cosa, estas deben tener una alta relación entre sí.

Cuantas más cosas haga una función sin relación entre sí, más complicado será entender el código.

Existen dos tipos de cohesión:

## Escuela Directa | Blog

- **Cohesión fuerte:** indica que existe una alta relación entre los elementos existentes dentro del módulo. Este debe ser nuestro objetivo al diseñar programas.

Un ejemplo bien claro de cohesión débil o fuerte podría ser el siguiente:

Queremos tener una función llamada `suma()` cuya finalidad sea sumar dos argumentos numéricos. Una versión con **cohesión fuerte** de esta función sería la siguiente:

```
def suma(num1, num2):  
    resultado = num1 + num2  
    return resultado
```

El problema ocurriría si al programador le dan ganas de poner todo en un solo sitio, y además de sumar dos números, aprovecha esta función para:

- pedirle al usuario que ingrese esos números (en vez de pedirlo en otra función y pasarlos como argumentos),
- y como va a necesitar que esos números sean `float()` también va a hacer la conversión dentro de la misma función.

El resultado de añadir estas funcionalidades sería una función de **cohesión débil**:

```
def suma():  
    num1 = float(input("Elige un número"))  
    num2 = float(input("Elige otro número"))  
    resultado = num1 + num2  
    return resultado
```

Podrías estar pensando que estas otras dos funcionalidades extra no son para tanto, pero supongamos que una persona quiere usar nuestra función `suma()` pero ya tiene los números y no quiere pedirlos por pantalla, nuestra función no le serviría.

## Escuela Directa | Blog

```
def suma(lista_numeros):  
    resultado = 0  
    for n in lista_numeros:  
        resultado += n  
    return resultado
```

Por supuesto que este es un ejemplo muy simple, en el que las implicaciones no serían tan dramáticas, pero es importante buscar que las funciones realicen una única tarea, o al menos un conjunto de tareas pero relacionadas entre sí.

Las ventajas de diseñar código con cohesión fuerte son:

- **Reducir la complejidad** del módulo, ya que tendrá un menor número de operaciones.
- Se podrá **reutilizar los módulos** más fácilmente
- El sistema será más **fácil de mantener**.

La cohesión se vincula a otro de los pilares llamado acoplamiento (al que explico en [este otro artículo](#)). Normalmente **cohesión fuerte** se relaciona con **acoplamiento débil**.



**FEDERICO GARAY**



## Posts favoritos

### Encapsulamiento - Pilares de la Programación Orientada a Objetos en Python

*El encapsulamiento es el pilar de la programación orientada a objetos que se relaciona con ocultar al exterior determinados estados internos de un objeto , tal que sea el mismo objeto quien acceda o los modifique , pero que dicha acción no se pueda llevar a cabo desde el exterior , llamando a los ...*

### Acoplamiento - Pilares de la Programación Orientada a Objetos en Python

*El acoplamiento es un concepto que mide la dependencia entre dos módulos distintos (como por ejemplo, clases). Podemos hablar de dos tipos: Acoplamiento débil , que implica que no hay dependencia entre un módulo y otros. Esta es la situación ideal. Acoplamiento fuerte , que es la situación ...*

## Con la tecnología de Blogger

Todos los derechos reservados. Prohibida su reproducción total o parcial