Escuela Directa | Blog

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, …

Acoplamiento - Pilares de la Programación Orientada a Objetos en Python

escrito por Federico Garay

El **acoplamiento** es un concepto que mide la **dependencia entre dos módulos distinto**s (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 contraria, e indica que el módulo tiene dependencias internas con otros.

Se trata de un pilar vinculado con la **cohesión**, ya que por lo general un acoplamiento débil se asocia a una cohesión fuerte. Esta última es la situación buscada al escribir código. Es decir, buscamos que una clase o función no tenga dependencias con otras clases o funciones, y que las tareas que realizan estén

Para terminar de comprenderlo, imaginemos la situación contraria: un código fuertemente acoplado:

- Si quisiéramos reutilizar un módulo que depende de otros, deberíamos "traer" también todas las dependencias, de lo contrario resultaría en errores y pérdida de funcionalidad.
- Si efectuamos un cambio en un módulo dependiente de otros, los efectos de este cambio pueden afectar a los otros módulos que dependen del anterior.

Es muy importante prestarle atención a medida que nuestros programas crecen y se hacen más complejos, donde este tipo de situaciones pueden comenzar a ocurrir inadvertidamente. Si esto sucede, un cambio en una clase puede inutilizar otra, haciendo que deje de funcionar. **Una situación de acoplamiento fuerte puede originar errores que son difíciles de depurar.**

Veamos el siguiente ejemplo:

```
class Mascota:
    tiene_patas = True
    pass
class Perro:
    def correr(self, velocidad):
        if Mascota.tiene_patas:
            self.velocidad = velocidad
mi_mascota = Perro()
mi_mascota.correr("rápido")
print(mi_mascota.velocidad)
```

Tenemos una clase `Mascota` que define un atributo de clase `tiene_patas`. Por otra parte, la clase `Perro` basa el comportamiento del método `correr()` en el atributo `tiene_patas` de la clase `Mascota`. Tenemos acoplamiento fuerte, ya que hay una dependencia entre la función de una clase con el atributo de otra.

efectos adversos mencionados.

FEDERICO GARAY

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 …

Cohesión - Pilares de la Programación Orientada a Objetos en Python

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 del …