Conferencia 06

En función de mejorar

Objetivos

- **Abstracción y Encapsulamiento**: Las **funciones** como "cajas negras".
- **Principio DRY**: *Don't Repeat Yourself* (No te repitas).
- **Anatomía**: Signatura, parámetros y retorno.
- **Ámbito (Scope)**: Variables locales vs. globales.
- **Flexibilidad**: Argumentos por defecto y `*args`.
- **Aplicación Práctica**: Descomposición de un algoritmo.

El Problema: Código Repetido

- **Violación del principio DRY**: La misma lógica aparece en múltiples lugares.
- **Mantenimiento Difícil**: Corregir un error implica buscar y cambiar todas las copias.
- **Baja Legibilidad**: El “qué” se pierde en los detalles del “cómo”.

Ejemplo de Código

La Solución: Abstracción y Encapsulamiento

- **Abstracción**: Ocultar la complejidad.
- *Analogía*: Manejar un carro sin conocer el motor.
- **Encapsulamiento**: Agrupar el código en una unidad autónoma.
- *Resultado*: Una “caja negra” reutilizable.

Anatomía de una definición de Función

```
1  def func_name(parameter1, parameter2):  # <-- Signatura
2  # Cuerpo de la función                  # <-- Cuerpo y Ámbito Local
3  # ...lógica...
4      return final_value                  # <-- Valor de Retorno
```

- La **signatura** define la interfaz.
- El **cuerpo** contiene la implementación (el “cómo”).
- El **retorno** es la “puerta de salida”.

Anatomía de un llamado a Función

```
1  var_name = func_name(arg1, arg2)
```

- Al definir la función se definen **parámetros**.
- Al llamarla se pasan **argumentos**.
- Ya lo han visto: `min`, `max`, `len`, ...

Ejemplo de Código

Ámbito o Scope SCOPE

- Las variables creadas dentro de una función son **locales**.
- Sólo existen mientras la función se ejecuta.
- **Ventajas**:
- Evita efectos secundarios.
- Previene colisiones de nombres.

Flexibilidad: Parámetros Avanzados

1. Valores por Defecto

Hacen que un argumento sea **opcional**.

```
def format_name(nombre, apellidos, inicial=None):
```

- Los parámetros opcionales siempre van al final.

Flexibilidad: Parámetros Avanzados

2. Argumentos Variables (`*args`)

Permite pasar un número **indeterminado** de argumentos.

```
def calcular_producto(*factores):
```

Python los agrupa automáticamente en una **tupla**.

Ejemplo de Código

Aplicación: Ordenación por Selección

Un algoritmo clásico para ordenar una lista.

Lógica principal:

1. Encontrar el elemento más pequeño del resto de la lista.
2. Intercambiarlo con la posición actual.
3. Repetir hasta ordenar toda la lista.

`[64, 25, 12, 22, 11]` –> `[11, 25, 12, 22, 64]` –> `[11, 12, 25, 22, 64]` ...

Descomposición Funcional

En lugar de un solo bloque de código, lo separamos en responsabilidades:

1. **`find_min_index()`**: Una función que sólo busca el índice del mínimo en un tramo de la lista.
2. **`selection_sort()`**: La función principal que orquesta el proceso, usando a `find_min_index()` como una herramienta.

Ejemplo de Código

Beneficios de la Descomposición

- **Responsabilidad Única**: Cada función hace una sola cosa.
- **Abstracción y Legibilidad**: El código de alto nivel (`selection_sort`) es mucho más fácil de entender.
- **Reusabilidad**: `find_min_index` podría usarse en cualquier otro problema.

Resumen Final

- **Abstracción**: Es tu herramienta principal contra la complejidad.
- **DRY**: Un código sin repeticiones es un código mantenible.
- **Scope**: Protege tu código de efectos inesperados.
- **Descomposición**: Piensa en problemas grandes como una colección de problemas pequeños y resueltos.

Conferencia 06

En función de mejorar