Paso a Paso: Funcionamiento de la recursividad con factorial en Python

Vamos a analizar paso a paso lo que ocurre cuando llamamos a la función `factorial(5)` usando una función recursiva.

Definición de la función:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Llamada inicial:

```
factorial(5)
```

Proceso de ejecución:

1. `factorial(5)` no entra en el caso base, así que hace:

   ```
   5 * factorial(4)
   ```

2. Para resolver `factorial(4)`:

   ```
   4 * factorial(3)
   ```

3. Para resolver `factorial(3)`:

   ```
   3 * factorial(2)
   ```

4. Para resolver `factorial(2)`:

   ```
   2 * factorial(1)
   ```

5. Para resolver `factorial(1)`:

   ```
   1 * factorial(0)
   ```

6. Ahora llegamos al caso base:

   `factorial(0)` retorna `1`

Ahora se resuelven las llamadas en orden inverso:

- `factorial(1) = 1 * 1 = 1`
- `factorial(2) = 2 * 1 = 2`
- `factorial(3) = 3 * 2 = 6`
- `factorial(4) = 4 * 6 = 24`
- `factorial(5) = 5 * 24 = 120`

Representación en forma de pila de llamadas:

```
factorial(5)
 → 5 * factorial(4)
       → 4 * factorial(3)
             → 3 * factorial(2)
                   → 2 * factorial(1)
                         → 1 * factorial(0)
                               → 1 (caso base)
```

Luego, cada llamada vuelve y se multiplica con el valor retornado hasta llegar al resultado final: `120`.

Conceptos clave:

- Cada llamada queda "en pausa" hasta que se resuelve la siguiente.
- Se utiliza una estructura llamada **pila de llamadas** (call stack).
- Si no se llega a un caso base, la pila se llena hasta causar un error.

Ventajas del enfoque paso a paso:

Este tipo de análisis ayuda a comprender la lógica de la recursividad y a depurar posibles errores en la función.

Este ejemplo es ideal para introducir recursividad, ya que es sencillo, tiene un solo caso base, y el flujo de ejecución se puede seguir con facilidad.