Ciclos Anidados

Materia: Algoritmos y Programación

Prof. D.Sc. BARSEKH-ONJI Aboud
Facultad de Ingeniería
Universidad Anáhuac México

13 de octubre de 2025

Agenda

1. Parte Teórica: ¿Qué son los Ciclos Anidados?

2. Ejemplos Prácticos

¿Qué es un Ciclo Anidado?

Definición

Un ciclo anidado es, en su forma más simple, un ciclo que se encuentra **dentro de otro ciclo**. El ciclo exterior se conoce como **ciclo externo** y el ciclo interior como **ciclo interno**.

La Analogía del Reloj

La forma más intuitiva de entenderlo es pensar en las manecillas de un reloj:

- El **ciclo externo** es el **horario**: se mueve lentamente, una vez por hora.
- El **ciclo interno** es el **minutero**: se mueve rápidamente.

Por cada hora que avanza el horario, el minutero debe completar su **ciclo completo** de 60 minutos. Esta es la esencia de un ciclo anidado.

Estructura y Flujo de Ejecución

¿Qué va a mostrar el código?

La indentación (sangría) es clave para definir qué ciclo está dentro de cuál.

```
1
2 for i in range(5):
3     for j in range(3):
4         print(f"i={i}, j={j}")
5
```

Estructura y Flujo de Ejecución

Flujo de Ejecución

El orden de ejecución es el siguiente:

1. El ciclo **externo** comienza su primera iteración.
2. El ciclo **interno** se activa y ejecuta **todas** sus iteraciones, de principio a fin.
3. Una vez que el ciclo interno termina, el ciclo **externo** comienza su **segunda** iteración.
4. El ciclo **interno** se **reinicia** y vuelve a ejecutar **todas** sus iteraciones.
5. El proceso se repite hasta que el ciclo externo finaliza.

Estructura y Flujo de Ejecución

¿Qué va a mostrar el código?

La indentación (sangría) es clave para definir qué ciclo está dentro de cuál.

```
1
2 for i in range(5):
3     for j in range(3):
4         print(f"i={i}, j={j}")
5     print(f"i={i}, j={j}")
6 print(f"i={i}, j={j}")
7
```

Usos Comunes de los Ciclos Anidados

¿Para qué sirven?

Los ciclos anidados son fundamentales para resolver problemas que tienen una naturaleza bidimensional o jerárquica.

Aplicaciones Típicas

- **Procesar estructuras de datos 2D:** Recorrer matrices, tablas o "listas de listas". El ciclo externo itera sobre las filas y el interno sobre las columnas de cada fila.

Usos Comunes de los Ciclos Anidados

¿Para qué sirven?

Los ciclos anidados son fundamentales para resolver problemas que tienen una naturaleza bidimensional o jerárquica.

Aplicaciones Típicas

- **Procesar estructuras de datos 2D:** Recorrer matrices, tablas o "listas de listas". El ciclo externo itera sobre las filas y el interno sobre las columnas de cada fila.
- **Generar Patrones:** Dibujar figuras geométricas en la consola, como cuadrados, triángulos o pirámides de asteriscos.

Usos Comunes de los Ciclos Anidados

¿Para qué sirven?

Los ciclos anidados son fundamentales para resolver problemas que tienen una naturaleza bidimensional o jerárquica.

Aplicaciones Típicas

- **Procesar estructuras de datos 2D:** Recorrer matrices, tablas o "listas de listas". El ciclo externo itera sobre las filas y el interno sobre las columnas de cada fila.
- **Generar Patrones:** Dibujar figuras geométricas en la consola, como cuadrados, triángulos o pirámides de asteriscos.
- **Crear Combinaciones:** Emparejar cada elemento de un conjunto con cada elemento de otro. Por ejemplo, generar todas las combinaciones posibles de un mazo de cartas (palos y números).

Usos Comunes de los Ciclos Anidados

¿Para qué sirven?

Los ciclos anidados son fundamentales para resolver problemas que tienen una naturaleza bidimensional o jerárquica.

Aplicaciones Típicas

- **Procesar estructuras de datos 2D:** Recorrer matrices, tablas o "listas de listas". El ciclo externo itera sobre las filas y el interno sobre las columnas de cada fila.
- **Generar Patrones:** Dibujar figuras geométricas en la consola, como cuadrados, triángulos o pirámides de asteriscos.
- **Crear Combinaciones:** Emparejar cada elemento de un conjunto con cada elemento de otro. Por ejemplo, generar todas las combinaciones posibles de un mazo de cartas (palos y números).
- **Simulaciones Simples:** Iterar a través del tiempo (ciclo externo) y, para cada paso de tiempo, actualizar el estado de múltiples objetos o agentes (ciclo interno).

Agenda

1. Parte Teórica: ¿Qué son los Ciclos Anidados?

2. Ejemplos Prácticos

Ejemplo 1: Tablas de Multiplicar

Objetivo

Vamos a crear un programa que imprima en la consola las tablas de multiplicar del 1 al 10. Este es el ejemplo perfecto para ver cómo el ciclo interno se ejecuta por completo para cada paso del ciclo externo.

Nuestra Lógica

Usaremos dos ciclos `for` anidados:

- **Ciclo Externo:** Iterará a través del número de la tabla que queremos calcular (del 1 al 10).
- **Ciclo Interno:** Para cada tabla, iterará a través del número por el cual vamos a multiplicar (también del 1 al 10).

Ejemplo 1: Código y Salida

Código en Python

```python
# Ciclo externo para el numero de la tabla
for tabla in range(1, 11):
    print(f"\n--- Tabla del {tabla} ---")

    # Ciclo interno para el multiplicador
    for multiplicador in range(1, 11):
        resultado = tabla * multiplicador
        print(f"{tabla} x {multiplicador} = {resultado}")

```

Salida en Consola (Fragmento)

```
--- Tabla del 1 ---
1 x 1 = 1
1 x 2 = 2
...
1 x 10 = 10

--- Tabla del 2 ---
2 x 1 = 2
2 x 2 = 4
... etc.
```

Ejemplo 2: Creando Combinaciones

Objetivo

Imagina que tienes dos listas: una con tipos de plato principal y otra con tipos de bebida. Escribe un programa para generar e imprimir todas las combinaciones posibles de menús que se pueden crear. Utiliza ciclos anidados de for para lograr esto.

Estructura de Datos

Trabajaremos con dos listas simples de strings.

```
1 platos_principales = ["Pasta", "Pizza", "Ensalada"]
2 bebidas = ["Agua", "Refresco", "Jugo"]
3
```

Ejemplo 2: Creando Combinaciones

Nuestra Lógica

- **Ciclo Externo:** Recorrerá la lista de 'platos-principales', tomando un plato en cada iteración.
- **Ciclo Interno:** Para **cada** plato del ciclo externo, este ciclo recorrerá la lista **completa** de 'bebidas'.
- **Acción:** Dentro del ciclo interno, imprimiremos la combinación del plato actual con la bebida actual.

Ejemplo 3: Encontrando Elementos Comunes

Objetivo

Tenemos dos listas de invitados para dos eventos diferentes. Escribe un programa que compare ambas listas y genere una nueva lista que contenga únicamente los nombres de las personas que están invitadas a **ambos** eventos. Utiliza ciclos anidados de for y una condición if para lograr esto.

Estructura de Datos

Trabajaremos con dos listas simples de strings.

```
1 invitados_evento_A = ["Ana", "Carlos", "Sofia", "Luis"]
2 invitados_evento_B = ["Sofia", "Pedro", "Ana", "Laura"]
3
```

Ejemplo 3: Encontrando Elementos Comunes

Nuestra Lógica

- **Ciclo Externo:** Recorrerá la primera lista de invitados, tomando un nombre en cada iteración.
- **Ciclo Interno:** Para **cada** nombre del ciclo externo, este ciclo recorrerá la **segunda lista completa**.
- **Condición 'if':** Dentro del ciclo interno, una condición 'if' comparará el nombre del ciclo externo con el nombre del ciclo interno. Si son idénticos, significa que hemos encontrado un invitado en común.

Códigos de Solución

Códigos de Solución: Ejemplo1

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example23.py

Códigos de Solución: Ejemplo2

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example24.py

Códigos de Solución: Ejemplo3

https://github.com/AboudOnji/ExamplesAyP/blob/main/Example25.py

Tarea Final: Simulador de Torneo

Objetivo

Escribir un programa que tome una lista de jugadores (Ana, Bruno, Carlos, Diana) y que genere todos los enfrentamientos posibles para un torneo "todos contra todos", donde cada jugador se enfrenta a cada uno de los otros jugadores **una sola vez**.

Tarea Final: Simulador de Torneo

Estructura de Datos y Salida Esperada

Dado una lista de jugadores:

```
jugadores = ["Ana", "Bruno", "Carlos", "Diana"]

```

La salida en consola debe ser la siguiente (el orden puede variar):

```
--- Rondas del Torneo ---
Ana vs. Bruno
Ana vs. Carlos
Ana vs. Diana
Bruno vs. Carlos
Bruno vs. Diana
Carlos vs. Diana
```

Tarea Final: Simulador de Torneo

El reto y la lógica a seguir

El reto principal es evitar enfrentamientos repetidos (si ya generaste 'Ana vs. Bruno', no debes generar 'Bruno vs. Ana') y evitar que un jugador se enfrente a sí mismo ('Ana vs. Ana').

- Necesitarás dos ciclos `for` anidados para recorrer la lista de jugadores.
- **La Clave:** Para evitar repeticiones, el ciclo interno **no debe empezar desde el principio** de la lista en cada iteración. Debe comenzar desde la posición **siguiente** a la del jugador actual del ciclo externo.
- (Pista: necesitarás usar `range()` y `len()` para controlar los índices de las listas).