📝 Apuntes de Programación: Las Listas (Lists) en Python

Las Listas son colecciones de datos **ordenadas** y **mutables** (modificables). Pueden contener elementos de diferentes tipos (heterogéneas) y permiten elementos duplicados, siendo la estructura más flexible de Python.

1. Creación e Inicialización

A. Formas de Declarar una Lista

Código de Ejemplo	Descripción
`lista = []`	Lista vacía usando corchetes `[]`. Es la forma más común.
`lista2 = list()`	Lista vacía usando el constructor `list()`.
`lista3 = [2, 4, 589, 33, 1, 44, 6, 7, 2]`	Lista inicializada con elementos. **Permite duplicados** y mantiene el orden.
`lista5 = [34, "Jose Maria", False, 45.6, [1, 5, "DAM"]]`	Lista **heterogénea** (diferentes tipos de datos, incluyendo otras listas).

B. Listas Anidadas (Matrices/Vectores)

Una lista puede contener otras listas, lo que permite crear estructuras multidimensionales (matrices).

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Vector de 2 dimensiones
print(matriz[2][0]) # Accede a la fila 2 (índice 2) y al elemento 0: 7
```

2. Recorrido y Acceso a Elementos

A. Acceso Posicional

Los elementos se acceden por su índice, que comienza en `0`.

```
print(lista5[6][2]) # Accede al elemento 6 ([1, 5, "DAM"]) y luego a su elemento 2 ("DA
```

B. Recorrido de Listas

Existen tres formas principales de iterar, dependiendo de si necesitas el índice o solo el valor.

Método	Propósito	Sintaxis
Simple (Solo valor)	Cuando solo importa el contenido.	`for i in lista5:`

Por Rango (Índice y valor)	Permite acceder y modificar usando el índice.	`for i in range(0, len(lista5)): print(i, "-", lista5[i])`
Con `enumerate()` (¡Mejor práctica!)	La forma más limpia y pitónica de obtener el índice y el valor a la vez.	`for i, nombre in enumerate(listaNombre): print(i, "-", nombre)`

3. Modificación y Manipulación

A. Adición de Elementos

Método	Propósito	Comportamiento Clave
`list.append(elemento)`	Añade el elemento **al final** de la lista.	
`list.insert(indice, elemento)`	Añade el elemento en la posición especificada. Si el índice es muy grande, lo inserta al final.	`lista7.insert(1, 333)`
`list1 + list2`	**Suma de listas** (Crea una lista nueva).	`lista6 = lista3 + lista4`
`list1.extend(list2)`	Añade los elementos de `list2` al final de `list1` (modifica `list1`).	`lista3.extend(lista4)`

B. Eliminación de Elementos

Método	Propósito	Comportamiento Clave
`list.pop(indice)`	Elimina el elemento en el `indice` y **lo devuelve**. Si no se especifica el índice, elimina el último. El índice puede ser negativo (`-1` es el último).	`elemento = lista7.pop(2)`
`list.remove(valor)`	Elimina la **primera aparición** del `valor` especificado.	Lanza un error (`ValueError`) si el valor no existe.

C. Ordenación

- `list.sort()` **:** Ordena la lista original **en su lugar** (modifica la lista) de forma ascendente (numérica o alfabética).
- `list.sort(reverse=True)` **:** Ordena de forma descendente.
- **Limitación:** No se puede ordenar una lista con mezcla de tipos incompatibles (ej. números y cadenas).

4. Búsqueda, Conteo y Slicing

A. Búsqueda y Conteo

- **Verificación de Pertenencia:**

```
if 333 not in lista7: # Forma rápida de saber si un elemento existe
    print("No esta en la lista😔")
```

- **Contar Ocurrencias:**

```
print(f"aparece {lista7.count(333)} veces")
```

- **Encontrar Índice:**

```
print(lista7.index(0)) # Devuelve la primera posición. Cuidado: lanza excepción si
```

B. Slicing Avanzado (Rebanado)

El *slicing* permite extraer sub-listas o modificar el orden de forma concisa.

- **Sintaxis:** `lista[inicio : fin : paso]`
- **Invertir la lista:**

```
print(lista7[::-1]) # El paso de -1 invierte la lista.
```

- **Saltos (Paso):**

```
print(lista7[::2]) # Si el paso es 2, coge cada segundo elemento (impares si el ini
```

5. Interacción con el Módulo `random`

El módulo `random` es esencial para simular sorteos o desordenar datos.

Función	Propósito	Comportamiento
`random.choice(lista)`	Extrae y devuelve un elemento **aleatorio** de la lista.	
`random.sample(lista, k)`	Extrae una sub-lista de `k` elementos **sin repetición**.	`random.sample(alumnos, 4)`
`random.shuffle(lista)`	Desordena la lista **original** en su lugar (mutación).	`random.shuffle(alumnos)`

6. Consejos para Mejorar la Programación con Listas (Examen Pro-Tip)

🎯 Principio 1: Las Listas son Dinámicas

- Una lista es la mejor opción cuando esperas que el tamaño de la colección **cambie** (añadir, quitar elementos). Para colecciones fijas y rápidas, usa Tuplas.

🎯 Principio 2: Evitar la Conversión Manual a `str`

- Viste en el código una forma manual de limpiar la representación de una lista:

```
texto2= texto2.replace("[","  ") # ¡Evita esto!
```

- **Mejor Práctica (Método `join`):** Para convertir una lista de cadenas en una sola cadena, usa `join()` . Esto es más eficiente, limpio y evita tener que limpiar caracteres feos.

```
# Ejemplo de uso profesional:
nombres_lista = ["Ana", "Luis", "Eva"]
texto_limpio = " | ".join(nombres_lista) # Resultado: "Ana | Luis | Eva"
```

🎯 Principio 3: El Uso Correcto de `copy()`

- Cuando asignas una lista a otra (`num2 = num1`), ambas variables apuntan a la **misma lista** (referencia). Si modificas `num2` , modificas `num1` .
- **Para crear una copia independiente** (una lista nueva en memoria), usa `list.copy()` o el *slicing* completo:

```
num2 = num1.copy() # Crea una copia superficial (lista nueva)
# O también: num2 = num1[:]
```

- **Consejo de Examen:** El concepto de copia superficial (`copy()`) frente a referencia es un tema recurrente en exámenes sobre estructuras de datos.