Capítulo 1

INTRODUCCIÓN A LAS LISTAS EN PYTHON

LAS LISTAS

En programación, las **colecciones** son estructuras de datos que nos permiten almacenar y organizar múltiples elementos bajo un solo nombre. Son fundamentales porque facilitan el manejo, la agrupación y el procesamiento de datos relacionados.

Python ofrece varias colecciones integradas que se adaptan a diferentes necesidades, cada una con características propias que las hacen útiles en distintos contextos.

¿Para qué sirven las colecciones?

- **Almacenar conjuntos de datos**: Desde listas de números, palabras, hasta objetos complejos.
- **Organizar datos**: Permiten agrupar información relacionada para acceder y manipularla fácilmente.
- **Optimizar operaciones**: Muchas colecciones están diseñadas para realizar ciertas operaciones (como búsquedas, ordenamientos o conteos) de manera eficiente.

Conceptos clave a entender

Mutabilidad: Una colección *mutable* puede cambiar después de ser creada (agregar, eliminar o modificar elementos). Una colección *inmutable* no puede cambiar; cualquier modificación implica crear una nueva.

Orden: Una colección *ordenada* mantiene el orden en que se agregan los elementos. Una colección *no ordenada* no garantiza ningún orden específico.

Duplicados: Algunas colecciones permiten almacenar elementos repetidos. Otras aseguran que cada elemento sea único.

Ejemplos de colecciones básicas en Python

- `list`: ordenadas, mutables, permiten duplicados.
- `tuple`: ordenadas, inmutables, permiten duplicados.
- `set`: no ordenados, mutables, no permiten duplicados.
- `dict`: ordenados (desde Python 3.7), mutables, almacenan pares clave-valor con claves únicas.

Colección	Ordenada	Mutable	Duplicados	Tipo de elementos	Uso principal
`list`	Sí	Sí	Sí	Cualquier tipo	Colección dinámica de elementos
`tuple`	Sí	No	Sí	Cualquier tipo	Colección fija, inmutable
`set`	No	Sí	No	Cualquier tipo	Colección única, operaciones de conjunto
`dict`	Sí (desde 3.7)	Sí	Claves únicas, inmutables	Pares clave-valor	Mapeo o asociación clave-valor

Cuadro 1.1: Resumen de colecciones en Python

Listas en Python

Introducción

Las **listas** son una de las colecciones más utilizadas en Python debido a su versatilidad y facilidad de uso. Una lista es una secuencia ordenada de elementos que puede contener diferentes tipos de datos, y es mutable, lo que significa que se puede modificar después de creada.

Ejemplos comunes de uso de listas:

- Almacenar una lista de nombres de estudiantes en una clase.
- Guardar una serie de temperaturas medidas durante una semana.
- Mantener una colección de tareas pendientes en una aplicación.
- Agrupar diferentes tipos de datos, como números, cadenas y objetos en un solo contenedor.

Las listas en Python se definen encerrando los elementos entre corchetes `[]` y separándolos con comas. Cada elemento de la lista tiene un índice, que indica su posición dentro de la secuencia, empezando desde 0 para el primer elemento.

Propiedades fundamentales de las listas

Orden

Las listas mantienen el orden de los elementos según fueron añadidos. Esto permite acceder a ellos mediante índices.

Mutabilidad

Las listas son *mutables*, lo que quiere decir que se pueden modificar en el lugar: agregar, eliminar o cambiar elementos sin necesidad de crear una nueva lista.

Duplicados

Las listas permiten tener elementos repetidos, es decir, no hay restricción para almacenar valores duplicados.

Tipos de elementos

Las listas pueden contener elementos de cualquier tipo: enteros, cadenas, booleanos, objetos, e incluso otras listas.

Declaración y manipulación básica de listas

Declaración de una lista

```
# Lista vacía
mi_lista = []

# Lista con elementos
frutas = ["manzana", "banana", "cereza"]
```

Agregar elementos

```
frutas.append("naranja")  # Añade "naranja" al final de la lista
frutas.insert(1, "kiwi")  # Inserta "kiwi" en el índice 1
```

Eliminar elementos

```
frutas.remove("banana")    # Elimina la primera ocurrencia de "banana"
frutas.pop()               # Elimina y retorna el último elemento
frutas.pop(2)              # Elimina y retorna el elemento en índice 2
```

Tabla de métodos comunes para creación, adición y eliminación

Método	Descripción
`append(x)`	Añade el elemento `x` al final de la lista.
`insert(i, x)`	Inserta el elemento `x` en la posición `i`.
`extend(iterable)`	Extiende la lista agregando todos los elementos de `iterable`.
`remove(x)`	Elimina la primera ocurrencia del elemento `x`. Lanza error si no se encuentra.
`pop([i])`	Elimina y retorna el elemento en la posición `i` (por defecto el último).
`clear()`	Elimina todos los elementos de la lista.

Recorrido de listas

Las listas se pueden recorrer utilizando bucles para acceder o modificar sus elementos.

Ejemplo con `for`

```
frutas = ["manzana", "banana", "cereza"]
for fruta in frutas:
    print(fruta)
```

Ejemplo con índice

```
for i in range(len(frutas)):
    print("Índice {}: {}".format(i, frutas[i]))
```

Métodos especiales de listas

Algunos métodos útiles para combinar o transformar listas:

- `extend()` — para concatenar listas.
- `sort()` — ordena la lista in-place.
- `reverse()` — invierte el orden de los elementos.
- `count(x)` — cuenta cuántas veces aparece `x`.
- `index(x)` — devuelve el índice de la primera aparición de `x`.

Ejemplo:

```
lista1 = [1, 2, 3]
lista2 = [4, 5]
lista1.extend(lista2)     # lista1 ahora es [1, 2, 3, 4, 5]

lista1.sort()             # lista1 ordenada (ya estaba
    ordenada)
lista1.reverse()          # lista1 ahora es [5, 4, 3, 2, 1]

print(lista1.count(3))    # Imprime 1

print(lista1.index(4))    # Imprime 1
```

Cheatsheet básico de listas

Operación	Ejemplo
Crear lista	`lst = [1, 2, 3]`
Agregar elemento	`lst.append(4)`
Insertar en posición	`lst.insert(1, 5)`
Eliminar por valor	`lst.remove(2)`
Eliminar por índice	`lst.pop(0)`
Recorrer lista	`for x in lst: ...`
Ordenar lista	`lst.sort()`
Invertir lista	`lst.reverse()`

Diez problemas prácticos con listas en Python

En esta sección encontrarás problemas que te ayudarán a practicar la creación, manipulación y recorrido de listas, así como algunos métodos especiales.

1. **Crear una lista vacía y agregarle elementos.** Crea una lista vacía llamada `numeros` y añade los números del 1 al 5 usando el método `append`.

2. **Insertar elementos en una posición específica.** En la lista `numeros`, inserta el número 10 en la posición 2 (índice 1).

3. **Eliminar un elemento por valor.** Elimina el número 3 de la lista `numeros` usando el método `remove`.

4. **Eliminar un elemento por índice.** Elimina el elemento que está en la posición 0 usando `pop`.

5. **Recorrer una lista con un ciclo `for`.** Imprime cada elemento de la lista `numeros` usando un ciclo `for`.

6. **Recorrer una lista con índice.** Utiliza un ciclo `for` y `range` para imprimir el índice y el valor de cada elemento en `numeros`.

7. **Obtener el largo de una lista.** Imprime la cantidad de elementos que tiene la lista `numeros` usando `len()`.

8. **Ordenar una lista.** Ordena la lista `numeros` de menor a mayor usando el método `sort`.

9. **Invertir una lista.** Invierte el orden de la lista `numeros` usando el método `reverse`.

10. **Contar cuántas veces aparece un elemento.** Cuenta cuántas veces aparece el número 10 en la lista `numeros` usando `count`.

Problemas resueltos con listas en Python

1. **Crear una lista vacía y agregarle elementos.**

```
numeros = []
for i in range(1, 6):
    numeros.append(i)
print(numeros)  # Salida: [1, 2, 3, 4, 5]
```

2. **Insertar elementos en una posición específica.**

```
numeros.insert(1, 10)  # Inserta 10 en índice 1
print(numeros)  # Salida: [1, 10, 2, 3, 4, 5]
```

3. **Eliminar un elemento por valor.**

```
numeros.remove(3)  # Elimina el primer 3 que encuentra
print(numeros)  # Salida: [1, 10, 2, 4, 5]
```

4. **Eliminar un elemento por índice.**

```
numeros.pop(0)  # Elimina elemento en índice 0
print(numeros)  # Salida: [10, 2, 4, 5]
```

5. **Recorrer una lista con un ciclo `for`.**

```
for n in numeros:
    print(n)
```

6. **Recorrer una lista con índice.**

```
for i in range(len(numeros)):
    print(f"Índice {i}: {numeros[i]}")
```

7. **Obtener el largo de una lista.**

```python
print(len(numeros))  # Salida: 4
```

8. **Ordenar una lista.**

```python
numeros.sort()
print(numeros)  # Salida: [2, 4, 5, 10]
```

9. **Invertir una lista.**

```python
numeros.reverse()
print(numeros)  # Salida: [10, 5, 4, 2]
```

10. **Contar cuántas veces aparece un elemento.**

```python
print(numeros.count(10))  # Salida: 1
```

Problemas resueltos con listas en Python

Problemas que resaltan el orden

1. **Obtener el primer y último elemento de una lista**

```python
frutas = ["manzana", "banana", "cereza", "durazno"]
print(frutas[0])    # Primer elemento
print(frutas[-1])   # Último elemento
```

2. **Invertir el orden de una lista**

```python
numeros = [1, 2, 3, 4, 5]
numeros.reverse()
print(numeros)  # [5,4,3,2,1]
```

3. **Obtener una sublista (slice) con los elementos centrales**

```python
letras = ["a", "b", "c", "d", "e", "f"]
sublista = letras[2:5]
print(sublista)  # ["c", "d", "e"]
```

4. **Recorrer una lista con índice**

```python
colores = ["rojo", "verde", "azul"]
for i in range(len(colores)):
    print(f"Índice {i}: {colores[i]}")
```

Problemas que resaltan la mutabilidad

1. **Agregar elementos a una lista**

```python
animales = ["perro", "gato"]
animales.append("conejo")
print(animales)  # ["perro", "gato", "conejo"]
```

2. **Eliminar un elemento específico**

```python
animales = ["perro", "gato", "conejo"]
animales.remove("gato")
print(animales)  # ["perro", "conejo"]
```

3. **Modificar un elemento por índice**

```python
colores = ["rojo", "verde", "azul"]
colores[1] = "amarillo"
print(colores)  # ["rojo", "amarillo", "azul"]
```

4. **Vaciar una lista**

```python
numeros = [1, 2, 3, 4]
numeros.clear()
print(numeros)  # []
```

Problemas que resaltan el tipo de datos

1. **Lista con diferentes tipos y acceso a cada uno**

```python
mi_lista = [10, "hola", 3.14, True]
print(mi_lista[0])  # int: 10
print(mi_lista[1])  # str: "hola"
print(mi_lista[2])  # float: 3.14
print(mi_lista[3])  # bool: True
```

2. **Sumar solo los números en una lista mixta**

```python
datos = [10, "20", 5.5, "hola", 3]
suma = 0
for item in datos:
    if isinstance(item, (int, float)):
        suma += item
print(suma)  # 18.5
```

3. **Convertir todos los elementos a string**

```python
elementos = [100, False, 3.5, "texto"]
str_lista = [str(e) for e in elementos]
print(str_lista)  # ["100", "False", "3.5", "texto"]
```

4. **Contar elementos de cierto tipo**

```python
mezcla = [1, "a", 2.5, 3, "b", False, 7]
cont_enteros = sum(isinstance(x, int) and not isinstance
    (x, bool) for x in mezcla)
print(cont_enteros)  # 3 (1,3,7)
```