Módulo 2.3: Listas

Prof. Carlos Cedeño

Listas 📝

En los módulos pasados vimos las operaciones que podemos hacer con las variables y los distintos tipos de datos. Ahora introducimos una de las estructuras de datos más versátiles y utilizadas en Python: las **listas**. Las listas nos permitirán agrupar y gestionar colecciones de estos elementos.

¿Qué es una Lista? 🧐

Imagina que tienes varios datos que quieres guardar juntos, como una lista de compras, los nombres de tus amigos o las calificaciones de un examen. En Python, una **lista** es una colección ordenada y mutable de elementos.

- **Ordenada**: Los elementos mantienen el orden en el que los agregaste.
- **Mutable**: Puedes cambiar, agregar o eliminar elementos después de crear la lista.
- **Elementos**: Una lista puede contener elementos de diferentes tipos de datos (números, strings, booleanos, ¡e incluso otras listas!).

Creando Listas

Para crear una lista se usan corchetes `[]` y separamos los elementos con comas `,`.

```
# Lista vacía
lista_vacia = []
print(lista_vacia)

# Lista de números
numeros = [1, 2, 3, 4, 5]
print(numeros)

# Lista de strings
nombres = ["Ana", "Luis", "Carlos"]
print(nombres)

# Lista mixta (diferentes tipos de datos)
mixta = [10, "Hola", True, 3.14]
print(mixta)
```

Accediendo a los Elementos de una Lista

Podemos acceder a los elementos individuales de una lista usando su **índice**. El índice es la posición del elemento en la lista.

¡Importante! En Python, el primer elemento tiene el índice `0`.

```
frutas = ["manzana", "banana", "cereza", "durazno"]

# Acceder al primer elemento (índice 0)
print(frutas[0])  # Salida: manzana

# Acceder al tercer elemento (índice 2)
print(frutas[2])  # Salida: cereza
```

Índices Negativos

También podemos usar índices negativos. El índice `-1` se refiere al último elemento, `-2` al penúltimo, y así sucesivamente.

```
frutas = ["manzana", "banana", "cereza", "durazno"]

# Acceder al último elemento
print(frutas[-1]) # Salida: durazno

# Acceder al penúltimo elemento
print(frutas[-2]) # Salida: cereza
```

Slicing (Rebanadas)

Podemos obtener una porción (una sublista) de una lista usando la técnica de *slicing*. Especificamos un rango de índices `[inicio:fin]`. **El elemento en el índice `fin` no se incluye.**

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Obtener elementos desde el índice 2 hasta el índice 4 (sin incluir el 5)
sublista1 = numeros[2:5]
print(sublista1)  # Salida: [2, 3, 4]
```

```python
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Obtener los primeros 3 elementos
sublista2 = numeros[:3] # Si se omite el inicio, es desde el principio (índice 0)
print(sublista2)  # Salida: [0, 1, 2]

# Obtener elementos desde el índice 6 hasta el final
sublista3 = numeros[6:] # Si se omite el fin, es hasta el final de la lista
print(sublista3)  # Salida: [6, 7, 8, 9]

# Obtener una copia de toda la lista
copia_lista = numeros[:]
print(copia_lista) # Salida: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


# Slicing con paso (cada "n" elementos)
# [inicio:fin:paso]
numeros_pares = numeros[0:10:2] # De 0 a 9, tomando de 2 en 2
print(numeros_pares) # Salida: [0, 2, 4, 6, 8]

numeros_impares_reversa = numeros[9:0:-2] # De 9 a 1 (sin incluir 0), en reversa de 2 en 2
print(numeros_impares_reversa) # Salida: [9, 7, 5, 3, 1]
```

Operaciones Comunes con Listas

Las listas vienen con un conjunto de operaciones y métodos muy útiles.

Longitud de una Lista: `len()`

La función `len()` nos devuelve el número de elementos en una lista.

```
mi_lista = [10, 20, 30, 40]
print(len(mi_lista))  # Salida: 4
```

Concatenación: `+`

Podemos unir dos listas usando el operador `+`.

```
lista1 = [1, 2, 3]
lista2 = ["a", "b", "c"]
lista_concatenada = lista1 + lista2
print(lista_concatenada)  # Salida: [1, 2, 3, 'a', 'b', 'c']
```

Repetición: `*`

Podemos repetir los elementos de una lista usando el operador `*`.

```
lista_original = [0, 1]
lista_repetida = lista_original * 3
print(lista_repetida)  # Salida: [0, 1, 0, 1, 0, 1]
```

Pertenencia: `in` y `not in`

Podemos verificar si un elemento existe o no en una lista.

```
vocales = ['a', 'e', 'i', 'o', 'u']

print('a' in vocales)      # Salida: True
print('z' in vocales)      # Salida: False
print('b' not in vocales)  # Salida: True
```

Métodos de las Listas

Los métodos son funciones especiales que pertenecen a un objeto (en este caso, una lista) y nos permiten realizar acciones específicas sobre él. Se llaman usando la sintaxis `nombre_lista.metodo()`.

`append(elemento)`: Agregar al Final

Añade un elemento al final de la lista.

```
colores = ["rojo", "verde"]
colores.append("azul")
print(colores)  # Salida: ['rojo', 'verde', 'azul']
```

`insert(indice, elemento)`: Insertar en una Posición

Inserta un elemento en una posición específica.

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

`pop(indice)`: Eliminar y Devolver por Índice

Elimina y devuelve el elemento en el índice especificado. Si no se proporciona un índice, elimina y devuelve el último elemento.

```
letras = ['a', 'b', 'c', 'd']

# Eliminar el elemento en el índice 1
elemento_eliminado = letras.pop(1)
print(letras)             # Salida: ['a', 'c', 'd']
print(elemento_eliminado) # Salida: b

# Eliminar el último elemento
ultimo_elemento = letras.pop()
print(letras)             # Salida: ['a', 'c']
print(ultimo_elemento)    # Salida: d
```

`remove(elemento)`: Eliminar por Valor

Elimina la primera aparición del elemento especificado. Si el elemento no existe, genera un error.

```
mascotas = ["perro", "gato", "pez", "gato"]
mascotas.remove("gato") # Elimina la primera ocurrencia de "gato"
print(mascotas)       # Salida: ['perro', 'pez', 'gato']

# Si intentas eliminar un elemento que no existe:
# mascotas.remove("hamster") # Esto daría un ValueError
```

`index(elemento)`: Encontrar el Índice

Devuelve el índice de la primera aparición del elemento especificado. Si el elemento no existe, genera un error.

```
numeros = [10, 20, 30, 20, 40]
indice_de_20 = numeros.index(20)
print(indice_de_20) # Salida: 1 (el primer 20 está en el índice 1)

# Si intentas encontrar un elemento que no existe:
# print(numeros.index(50)) # Esto daría un ValueError
```

`count(elemento)`: Contar Ocurrencias

Devuelve el número de veces que un elemento aparece en la lista.

```
numeros = [1, 2, 3, 2, 2, 4, 2]
cantidad_de_dos = numeros.count(2)
print(cantidad_de_dos) # Salida: 4
```

`sort()`: Ordenar la Lista (Modifica la Original)

Ordena los elementos de la lista *in situ* (modifica la lista original). Por defecto, ordena de forma ascendente. Solo funciona si los elementos son comparables (ej. todos números o todos strings).

```
numeros_desordenados = [3, 1, 4, 1, 5, 9, 2, 6]
numeros_desordenados.sort()
print(numeros_desordenados) # Salida: [1, 1, 2, 3, 4, 5, 6, 9]

nombres_desordenados = ["Carlos", "Ana", "Beatriz"]
nombres_desordenados.sort()
print(nombres_desordenados) # Salida: ['Ana', 'Beatriz', 'Carlos']

# Ordenar de forma descendente
numeros_desordenados.sort(reverse=True)
print(numeros_desordenados) # Salida: [9, 6, 5, 4, 3, 2, 1, 1]
```

Nota: `sort()` no devuelve una nueva lista, modifica la original. Si quieres una nueva lista ordenada sin modificar la original, puedes usar la función `sorted()`.

```
mi_lista_original = [5, 2, 8, 1]
lista_ordenada_nueva = sorted(mi_lista_original)
print(mi_lista_original)     # Salida: [5, 2, 8, 1] (no cambia)
print(lista_ordenada_nueva)  # Salida: [1, 2, 5, 8]
```

`reverse()`: Invertir la Lista (Modifica la Original)

Invierte el orden de los elementos de la lista *in situ*.

```
mi_lista = [1, 2, 3, 4, 5]
mi_lista.reverse()
print(mi_lista) # Salida: [5, 4, 3, 2, 1]
```

`clear()`: Vaciar la Lista

Elimina todos los elementos de la lista.

```
mi_lista_llena = [10, 20, 30]
mi_lista_llena.clear()
print(mi_lista_llena) # Salida: []
```

`copy()`: Copiar una Lista

Devuelve una copia superficial de la lista. Esto es importante porque si simplemente asignas una lista a otra variable (`lista_b = lista_a`), ambas variables apuntarán a la *misma* lista en memoria. Modificar una afectará a la otra. `copy()` crea una lista independiente.

```
original = [1, 2, 3]
copia_superficial = original.copy()
asignacion_directa = original

copia_superficial.append(4)
asignacion_directa.append(5)

print(original)            # Salida: [1, 2, 3, 5] (afectada por asignacion_directa)
print(copia_superficial)   # Salida: [1, 2, 3, 4] (es independiente)
print(asignacion_directa)  # Salida: [1, 2, 3, 5]
```