«Talento Tech»

Iniciación a la Programación con Python

CLASE 7

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 7 | Listas y tuplas

Temario:

- Listas y tuplas: creación y manipulación.
- Uso de subíndices.
- Métodos de listas y tuplas.
- Recorrer una lista con while.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción a listas y tuplas: cómo organizar la información en Python

Imaginate que tenés que hacer las compras del mes. No alcanza con memorizar todo lo que tenés que comprar y, si lo intentás, probablemente olvides algo importante. Entonces, ¿qué podés hacer? Armar una lista de compras. Ahora, pensá en lo mismo aplicado a la programación: en lugar de recordar cada dato por separado, Python te ofrece una herramienta llamada listas, donde podés almacenar toda esa información junta y organizada. Ya no es necesario que manejes cada dato individualmente, sino que podrás organizar la información y agruparla en una lista para trabajar de manera ordenada.

¿Qué es una lista en Python?

En Python, una lista es una estructura que permite almacenar varios valores dentro de una sola variable. A diferencia de una simple variable que almacena un único valor, una lista puede contener varios elementos, como si fuera una caja que guarda distintas cosas, una al lado de la otra. Podés usar una lista para llevar el registro de todos los productos que querés comprar, los precios de esos productos o cualquier otra información que necesites almacenar y manipular de manera organizada.

Una lista se crea encerrando los elementos entre corchetes “[ ]” y separándolos por comas:

```
compras = ["manzanas", "pan", "leche", "queso"]
```

Acá tenemos una lista llamada compras que contiene cuatro productos. Cada uno de estos productos tiene una posición específica dentro de la lista, lo que nos permite acceder a ellos cuando lo necesitemos.

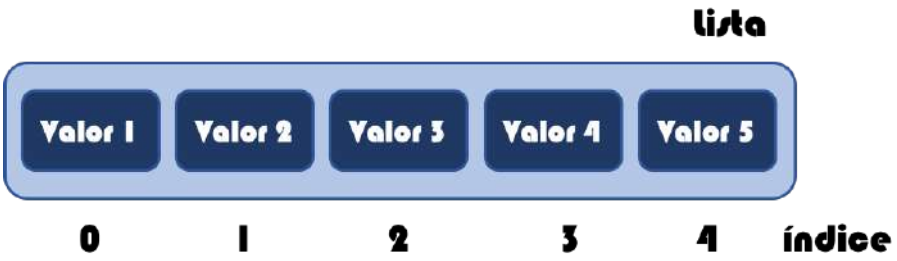
¿Por qué necesitamos listas?

Imaginá que en tu Proyecto Final Integrador estás desarrollando el inventario de productos de una pequeña tienda. A medida que se ingresan nuevos productos, los tenés que almacenar de alguna manera para poder acceder a ellos más tarde. Si usáramos una variable separada para cada producto, manejar esa información sería un caos. Con una lista, podés guardar todos esos productos en un solo lugar y luego acceder a ellos, modificarlos o incluso eliminarlos cuando sea necesario.

Creación y acceso a listas.

Crear una lista en Python es muy sencillo. Como vimos, sólo necesitás usar corchetes y separar los elementos con comas. Pero, ¿cómo hacés para acceder a un elemento específico dentro de la lista? Acá es donde entra el concepto de índices.

Cada elemento dentro de una lista tiene una posición, y esa posición se llama **“índice”**. El primer elemento de la lista tiene el índice 0, el segundo tiene el índice 1, y así sucesivamente. Podés acceder a un elemento de la lista simplemente usando su índice.



Estructura de una lista (Fuente: https://lineadecodigo.com/)

```
compras = ["manzanas", "pan", "leche", "queso"]
print(compras[0])  # Imprime "manzanas"
```

```
print(compras[2])  # Imprime "leche"
```

¿Entendés cómo funciona? El índice 0 nos da el primer producto, que es "manzanas", y el índice 2 nos da el tercero, que es "leche". Esto es muy útil cuando querés manipular información específica dentro de una lista.

Modificar y agregar elementos a una lista.

Como podés imaginar, las listas no están grabadas en piedra. Podés modificar los elementos existentes, agregar nuevos o eliminar algunos, dependiendo de lo que necesites.

Si querés cambiar el valor de un elemento, simplemente lo asignás nuevamente:

```
compras[1] = "yogur"
print(compras)  # Ahora la lista es ["manzanas", "yogur", "leche",
"queso"]
```

También podés agregar nuevos elementos a la lista usando el método **append():**

```
compras.append("cereales")
print(compras)  # Ahora la lista es ["manzanas", "yogur", "leche",
"queso", "cereales"]
```

Esto es clave cuando trabajás en proyectos como el PFI, donde el inventario de productos cambia constantemente. Cada vez que se agrega un nuevo producto, lo podés añadir a la lista de forma sencilla y seguir adelante.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Introducción a las tuplas: lo inmutable.

Ahora que ya conocés las listas, vamos a introducir otro concepto similar pero con una diferencia clave: las tuplas. Una tupla es como una lista, pero con una característica importante: es inmutable. Esto significa que, una vez que creás una tupla, no podés cambiar sus elementos. Las tuplas son útiles cuando querés asegurarte de que ciertos valores no cambian a lo largo del programa.

```
mi_tupla = ("manzanas", "pan", "leche")
```

A diferencia de las listas, si intentás cambiar un elemento de una tupla, Python te va a decir que no se puede:

```
# Esto va a dar un error porque las tuplas no se pueden modificar
mi_tupla[1] = "yogur"
```

Las tuplas son útiles cuando querés proteger datos que no deberían cambiar, como precios fijos o información de configuración en el sistema.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Métodos básicos para manipular listas.

Ya vimos que podemos acceder y modificar elementos de una lista usando índices. Sin embargo, Python también nos ofrece métodos que facilitan mucho la manipulación de listas. Algunos de estos métodos son esenciales para trabajar de manera eficiente con listas en aplicaciones reales.

Agregar elementos con append().

Como vimos antes, el método **append()** te permite agregar un nuevo elemento al final de la lista. Esto es especialmente útil en el contexto de un inventario de productos. Cada vez que se ingresa un nuevo producto al sistema, lo podés agregar a la lista con este método.

```
productos = ["manzanas", "pan", "leche"]
productos.append("queso")
print(productos)  # Imprime ["manzanas", "pan", "leche", "queso"]
```

Cada vez que uses append(), el nuevo elemento se agrega al final. Esto puede ser muy útil en tu PFI, ya que te permitirá mantener actualizado el inventario sin problemas.

Eliminar elementos con remove().

Cuando un producto se vende o se da de baja en el inventario, necesitás eliminarlo de la lista. Para esto, podemos usar el método **remove()**. Este método busca el primer valor que coincida con el que indicás y lo elimina de la lista.

```
productos = ["manzanas", "pan", "leche"]
productos.remove("pan")
print(productos)  # Imprime ["manzanas", "leche"]
```

Esta herramienta es muy útil cuando querés eliminar un producto específico de la lista de inventario en tu PFI.

Encontrar el largo de una lista con len().

Len() te permite conocer la cantidad de elementos que tiene una lista. Esto es especialmente útil para controlar bucles, donde querés recorrer toda la lista sin ir más allá de su límite.

```
productos = ["manzanas", "pan", "leche"]
cantidad_productos = len(productos)
print("Cantidad de productos:", cantidad_productos)  # Imprime
"Cantidad de productos: 3"
```

La utilización de éste método es clave en programas que manejan listas de elementos como inventarios, donde necesitás saber cuántos productos tenés registrados.

Recorrer una lista usando un bucle while.

Hasta ahora vimos cómo agregar y eliminar elementos de una lista, pero ¿qué pasa si querés procesar todos los productos de una lista uno por uno? Acá es donde el bucle while es súper útil.

Supongamos que querés mostrar la lista de productos disponibles en tu inventario. Lo podés hacer fácilmente usando un bucle while que recorre la lista desde el principio hasta el final, utilizando un índice que comienza en 0 y se incrementa en cada iteración.

```
productos = ["manzanas", "pan", "leche"]
indice = 0

while indice < len(productos):
   print("Producto", indice + 1, ":", productos[indice])
   indice = indice + 1
```

En este ejemplo, el bucle while comienza con el índice en 0 y recorre la lista hasta que el índice sea igual a la longitud de la lista, obtenida con len(productos). Cada vez que el bucle se ejecuta, muestra el producto en la posición correspondiente.

```
Producto 1 : manzanas
Producto 2 : pan
Producto 3 : leche
```

Aplicación real: registrar ventas en un día.

Vamos a ver un ejemplo práctico que podrías usar en tu PFI. Te proponemos registrar las ventas del día y luego mostrar el listado de productos vendidos. Para esto, usaremos una lista donde almacenamos los productos vendidos, un bucle while para ingresar cada venta, y otro bucle while para mostrar el resultado al final del día.

```
ventas = []
seguir = "S"

# Registramos las ventas hasta que el usuario decida no ingresar más
productos
while seguir == "S":
    producto = input("Ingresá el nombre del producto vendido: ")
    ventas.append(producto)

    seguir = input("¿Querés registrar otra venta? (S/N): ").upper()

# Mostramos los productos vendidos al final del día
print("\nProductos vendidos hoy:")
indice = 0
while indice < len(ventas):
    print("Producto", indice + 1, ":", ventas[indice])
    indice = indice + 1
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Diferencia entre listas y tuplas.

Una vez que te familiarizás con las listas, es hora de hablar de las tuplas. Como mencionamos antes, una tupla es una estructura similar a una lista, pero con una diferencia importante: **es inmutable.** Esto significa que una vez que creás una tupla, no podés cambiar sus elementos.

Las tuplas se crean usando paréntesis en lugar de corchetes, y su inmutabilidad las hace útiles cuando querés asegurarte de que los valores no cambien durante la ejecución del programa.

```
configuracion_tienda = ("nombre_tienda", "calle falsa 123", "Buenos Aires")
```

Acá, configuracion_tienda es una tupla con tres valores que no cambiarán. Si intentás modificarla, Python te va a lanzar un error:

```
 # Esto da error porque las tuplas son inmutables
configuracion_tienda[0] = "otro_nombre"
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Recorriendo una tupla con un bucle while.

Al igual que las listas, podés recorrer una tupla con un bucle while. Si bien no podés modificar los elementos de una tupla, sí podés acceder a ellos de la misma forma que con una lista.

```
configuracion_tienda = ("nombre_tienda", "calle falsa 123", "Buenos
Aires")
indice = 0

while indice < len(configuracion_tienda):
   print("Dato", indice + 1, ":", configuracion_tienda[indice])
   indice = indice + 1
```

En este ejemplo, estamos recorriendo la tupla de la misma manera que recorrimos una lista antes. El bucle se encarga de moverse por cada posición de la tupla, desde el índice 0 hasta el final.

Diferencias clave entre listas y tuplas.

Las listas y las tuplas comparten muchas similitudes, pero la diferencia más importante es que las listas son mutables (podés cambiar sus elementos), mientras que las tuplas son inmutables (no podés cambiar sus elementos una vez creados).

Usá listas cuando necesitás que los valores cambien o que puedas agregar/eliminar elementos, como en un inventario de productos. En cambio, usá tuplas cuando querés asegurarte de que ciertos valores no cambien, como la configuración de la tienda o la lista de provincias y ciudades fijas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Listas de Listas.

Las listas que contienen otras listas son muy útiles cuando querés organizar datos más complejos. Este tipo de listas te permite manejar múltiples elementos agrupados en una sola estructura, lo cual es ideal para situaciones donde necesitás asociar información entre sí, como en el caso de productos y sus precios o en sistemas que requieren manejar diferentes características de un mismo elemento.

¿Qué es una lista de listas?

Una lista de listas es simplemente una lista donde cada uno de sus elementos es, a su vez, otra lista. Podés pensar en esto como una tabla, donde cada fila representa una lista con múltiples datos relacionados.

Por ejemplo, en el contexto del Proyecto Final Integrador, podríamos tener una lista que almacene productos junto con su precio y cantidad disponible en el inventario. Cada "producto" sería una lista con esos tres elementos: nombre, precio y cantidad.

Ejemplo de lista de listas: inventario de productos.

Supongamos que tenés un inventario de productos y para cada producto querés guardar su nombre, su precio y la cantidad disponible. Podríamos usar una lista que contenga una sublista para cada producto, así:

```
# Lista de productos: cada producto tiene nombre, precio y cantidad
inventario = [
    ["manzanas", 100, 50],
    ["pan", 50, 20],
    ["leche", 60, 30]
]
```

```
# Recorrer el inventario y mostrar los datos de cada producto
indice = 0
while indice < len(inventario):
    producto = inventario[indice]
    print("Producto:", producto[0])
    print("Precio: $", producto[1])
    print("Cantidad disponible:", producto[2])
    print("-------------------------")
    indice = indice + 1
```

En el ejemplo anterior, cada producto es una lista que contiene tres elementos: el nombre del producto, su precio y la cantidad disponible. La lista inventario contiene tres listas, una por cada producto.

Para acceder a un producto en particular, usamos dos niveles de índice. El primer índice nos dice qué producto queremos (por ejemplo, inventario[0] para las "manzanas"). El segundo índice nos dice qué dato de ese producto queremos: inventario[0][0] es el nombre ("manzanas"), inventario[0][1] es el precio (100), y inventario[0][2] es la cantidad (50).

Recorremos cada producto en el inventario. Dentro del bucle, accedemos a los elementos individuales de cada producto con un segundo nivel de índices.

Ventaja de usar listas de listas.

Este tipo de estructura te permite manejar datos más complejos sin tener que crear muchas listas separadas. Por ejemplo, podés usar una lista de listas para gestionar un inventario completo con toda la información de cada producto agrupada en una sola estructura, lo que facilita la manipulación de los datos.

Si necesitás agregar más productos, simplemente añadís una nueva lista a inventario. Si necesitás modificar el precio o la cantidad de un producto, podés hacerlo accediendo directamente a los índices correctos dentro de la lista.

Modificar el precio de un producto
Retomando el ejemplo anterior, si quisiéramos modificar el precio de un producto específico en esta lista de listas, podríamos hacerlo así:

```
# Lista de productos: nombre, precio y cantidad
inventario = [
    ["manzanas", 100, 50],
    ["pan", 50, 20],
    ["leche", 60, 30]
]

# Pedir al usuario qué producto quiere modificar
producto_modificar = input("¿Qué producto querés modificar?: ")

# Recorrer el inventario para encontrar el producto y modificar su
precio
indice = 0
while indice < len(inventario):
    if inventario[indice][0] == producto_modificar:
        nuevo_precio = float(input("Ingresá el nuevo precio: "))
        inventario[indice][1] = nuevo_precio
        break
    indice = indice + 1

# Mostrar el inventario actualizado
print("\nInventario actualizado:")
indice = 0
while indice < len(inventario):
    producto = inventario[indice]
```

```
    print("Producto:", producto[0])
    print("Precio: $", producto[1])
    print("Cantidad disponible:", producto[2])
    print("-------------------------")
    indice = indice + 1
```

Primero pedimos a la persona que ingrese el nombre del producto cuyo precio quiere modificar. Luego recorremos la lista inventario usando un bucle while y comparamos el nombre de cada producto con el nombre ingresado por el usuario. Si encontramos una coincidencia, solicitamos el nuevo precio y actualizamos el valor en la lista correspondiente. Finalmente, mostramos el inventario actualizado.

Este tipo de solución es ideal cuando necesitamos trabajar con datos agrupados, y las listas de listas nos permiten hacerlo de forma clara y eficiente, manejando toda la información relevante de un solo vistazo.

Ejercicios prácticos:

Registro de productos en un inventario

Te proponemos implementar un sistema básico para registrar productos en el inventario de una tienda. El programa debe permitir que se agreguen productos a una lista hasta que se decida no agregar más. Luego, deberás mostrar todos los productos ingresados al inventario.

Tips:

- Usá una lista para almacenar los productos.
- Usá un bucle while para seguir ingresando productos mientras el usuario lo desee.
- Mostrá todos los productos registrados al final.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Consultar el stock de productos

Tu programa debe permitir consultar el inventario de una tienda para verificar si un producto está en stock. Si el producto está en la lista, el programa debe informarlo, si no, debe mostrar un mensaje indicando que no está disponible.

Tips:

- Usá una lista para almacenar los productos en stock.
- Permití que el usuario ingrese el nombre de un producto a consultar.
- Recorré la lista con un bucle while para verificar si el producto está en stock.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad