«Talento Tech»

Iniciación a la Programación con Python

CLASE 9

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 9 | Bucles for

Temario:

- Control de flujo: bucles for
- Slices de listas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Bucle for.

Imaginemos que estás gestionando una pequeña tienda y creaste una lista con todos los productos que posee el inventario. Cada vez que quieras ver los productos o procesar los datos, sería tedioso hacerlo manualmente uno por uno. Aquí es donde el bucle for se convierte en una herramienta fundamental. Nos permite recorrer automáticamente esa lista y trabajar con cada elemento de manera rápida y eficiente.

A diferencia del bucle while, que sigue ejecutándose hasta que una condición se vuelva falsa, el bucle for es ideal cuando ya sabés exactamente cuántas veces necesitás repetir una acción. En el caso del inventario de productos, por ejemplo, podemos recorrer todos los elementos de la lista de productos con for, mostrándolos uno a uno sin importar cuántos productos haya.

¿Cómo funciona el bucle for?

El bucle for toma una secuencia (una lista, una cadena de texto, un rango numérico) y la recorre de principio a fin, ejecutando un bloque de código para cada elemento en la secuencia. Es como una cadena de montaje: cada elemento de la secuencia pasa por el mismo proceso, uno tras otro. Esta es la sintaxis básica del bucle:

```
for variable in secuencia:
    # Bloque de código
```

Supongamos que tenés una lista de productos en tu tienda, y querés mostrar cada uno de ellos.

```
productos = ["manzanas", "naranjas", "bananas", "peras"]
```

```
for producto in productos:
   print("Producto disponible:", producto)
```

Este ejemplo es súper simple, pero te permite ver el concepto en acción. En cada repetición del bucle, la variable producto toma uno de los elementos de la lista y lo muestra en pantalla.

```
Producto disponible: manzanas
Producto disponible: naranjas
Producto disponible: bananas
Producto disponible: peras
```

Por supuesto, podrías haber logrado lo mismo usando un bucle while, pero como se ve en el código siguiente, se requieren algunas líneas más, y el uso de un contador para poder hacer lo mismo:

```
productos = ["manzanas", "naranjas", "bananas", "peras"]

i = 0
while i < len(productos):
   print("Producto disponible:", productos[i])
   i = i + 1
```

Podés probarlo y comprobar que la salida por la terminal es la misma.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Bucle for: Iterando a través de cadenas.

Una cadena de texto en Python es una secuencia de caracteres. Esto significa que, usando for, podemos recorrerla carácter por carácter. Por ejemplo, si tenés un nombre o cualquier palabra, el bucle puede procesar cada letra individualmente. Imaginemos que querés imprimir cada letra del nombre "Python":

```
palabra = "Python"

for letra in palabra:
   print("Letra:", letra)
```

Este bucle for va a recorrer la cadena "Python" una letra a la vez. En cada iteración, la variable letra va a tomar un valor distinto, empezando por "P", luego "y", después "t", y así sucesivamente hasta recorrer toda la palabra. El resultado en pantalla será:

```
Letra: P
Letra: y
Letra: t
Letra: h
Letra: o
Letra: n
```

Pero ¿cómo funciona esto? Es muy simple: cada vez que el bucle for itera toma el siguiente elemento de la secuencia. En este caso, la secuencia es la cadena de texto "Python". El bucle continúa hasta que no quedan más elementos (letras) para procesar. Este concepto de "iteración" es central en el bucle for: siempre recorrerá un objeto secuencial elemento por elemento.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Bucle for: Iterando a través de tuplas.

Una tupla es similar a una lista, pero con la diferencia de que es inmutable (no se puede modificar una vez creada). Al igual que con las listas y cadenas, podemos recorrer una tupla con for.

```
numeros = (1, 2, 3, 4, 5)

for numero in numeros:
   print("Número:", numero)
```

Cada número de la tupla se va a mostrar en pantalla uno por uno, ya que el bucle for recorre la tupla en orden.

```
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Integrando los conceptos.

Lo importante aquí es entender que for puede iterar sobre cualquier tipo de secuencia en Python: cadenas de texto, listas, tuplas y otros tipos de secuencias que veremos más adelante. Cada iteración toma un elemento y permite procesarlo de manera individual, lo que nos da una gran flexibilidad a la hora de manipular colecciones de datos. Podemos combinar el uso del bucle for para recorrer diferentes tipos de secuencias:

```
frutas = ["manzana", "banana", "pera"]

for fruta in frutas:
   print("Fruta:", fruta)
   for letra in fruta:
       print("Letra:", letra)
```

En este ejemplo, estamos recorriendo una lista de frutas y, luego, dentro del mismo bucle, recorremos cada fruta letra por letra. El resultado será:

```
Fruta: manzana
Letra: m
Letra: a
Letra: n
Letra: z
Letra: a
Letra: n
Letra: a
Fruta: banana
Letra: b
Letra: a
Letra: n
Letra: a
```

```
Letra: n
Letra: a
Fruta: pera
Letra: p
Letra: e
Letra: r
Letra: a
```

Break.

El uso de break en un bucle for en Python permite interrumpir la ejecución del bucle antes de que haya terminado su recorrido completo. Esto es útil cuando querés detener el bucle en el momento en que una condición específica se cumple, evitando que siga iterando innecesariamente.

Por ejemplo, imaginate que estás buscando un producto específico en una lista de productos. Una vez que encontrás ese producto, no tiene sentido seguir buscando, por lo que podrías utilizar break para detener el bucle.

Ejemplo:

```
# Lista de productos
productos = ["P001", "P002", "P003", "P004", "P005"]

# Producto que queremos encontrar
producto_a_buscar = "P003"

# Recorremos la lista buscando el producto
for producto in productos:
   if producto == producto_a_buscar:
       print("Producto encontrado:", producto)
       break  # Detenemos el bucle al encontrar el producto
```

```
    print("Buscando...")

print("Fin de la búsqueda.")
```

En este ejemplo, primero creamos una lista llamada ‘productos’, que contiene varios códigos de productos. Luego, utilizamos un bucle for para recorrer cada uno de estos códigos dentro de la lista. Mientras recorremos los productos, se verifica con una condición ‘if’ si el producto actual es igual al que estamos buscando, almacenado en la variable ‘producto_a_buscar’. Si esta condición se cumple, el programa imprime que el producto ha sido encontrado y usa la instrucción ‘break’ para detener la ejecución del bucle inmediatamente, evitando que siga buscando entre los demás productos.

Gracias a la herramienta ‘break’ el bucle no continúa una vez que se ha encontrado el producto deseado. Si el producto no se hubiera encontrado, el bucle habría seguido recorriendo el resto de la lista. El objetivo de ‘break’ es optimizar el proceso, evitando iteraciones innecesarias. Esta es la salida por pantalla:

```
Buscando...
Buscando...
Producto encontrado: P003
Fin de la búsqueda.
```

De esta manera el bucle no continúa una vez que se ha encontrado el producto deseado. Si el producto no se hubiera encontrado, el bucle habría seguido recorriendo el resto de la lista. El objetivo de ‘break’ es optimizar el proceso, evitando repeticiones innecesarias del bucle for.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Qué es range()?

La **función range()** es muy útil para generar secuencias de números, lo que nos permite controlar cuántas veces queremos que un bucle se repita tanto como recorrer listas utilizando índices numéricos. Se utiliza principalmente con bucles for para ejecutar un bloque de código un número determinado de veces.

La sintaxis básica de range() es la siguiente:

```
range(inicio, fin, paso)
```

Veamos qué significa cada uno de estos términos:

inicio- Es el número donde comienza la secuencia (es opcional, por defecto es 0).
fin- Es el número donde termina la secuencia (no incluye este valor).
paso- Es el intervalo entre cada número de la secuencia (opcional, por defecto es 1).

La función range() es útil para controlar cuántas veces se repite un bucle for. Nos permite:

- Definir un rango de números.
- Controlar desde dónde comienza y termina la secuencia.
- Ajustar el paso entre los números de la secuencia.

El uso de range() es fundamental cuando trabajás con iteraciones en Python, especialmente para recorrer listas, repetir tareas un número fijo de veces o manejar secuencias de números de manera flexible. Ahora vamos a ver varios ejemplos para que te quede claro cómo funciona.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejemplo 1: Uso básico de range().

En este primer ejemplo, generamos una secuencia de números desde 0 hasta 4. Notá que el número de "fin" (5) no está incluido.

```
for i in range(5):
   print(i)
```

Esto es lo que verás en la pantalla:

```
0
1
2
3
4
```

Acá range(5) genera los números del 0 al 4. Python ejecuta el bucle for una vez por cada número en esa secuencia.

Ejemplo 2: Valor de inicio específico.

Si queremos que el bucle comience desde un número distinto de 0, podemos indicarlo pasando dos argumentos a range(inicio, fin).

```
for i in range(3, 7):
   print(i)
```

Esto es lo que verás en la pantalla:

```
3
4
5
6
```

En este caso, range(3, 7) genera los números del 3 al 6, porque 7 es el límite superior, pero no está incluido.

Ejemplo 3: El parámetro paso.

Podemos controlar el intervalo entre cada número de la secuencia con el parámetro paso. Esto es útil si queremos saltar números.

```
for i in range(0, 10, 2):
   print(i)
```

Esto es lo que verás en la pantalla:

```
0
2
4
6
8
```

Acá range(0, 10, 2) genera números desde 0 hasta 8, saltando de 2 en 2.

Ejemplo 4: Secuencias decrecientes.

También podemos usar range() para generar secuencias decrecientes, indicando un valor de paso negativo.

```
for i in range(10, 0, -2):
   print(i)
```

Esto es lo que verás en la pantalla:

```
10
8
6
4
2
```

En este caso, range(10, 0, -2) genera números desde 10 hasta 2, restando de 2 en 2.

Ejemplo 5: El uso de range() para recorrer listas

Podemos usar range() para iterar sobre los índices de una lista y acceder a sus elementos.

```
frutas = ["manzana", "banana", "naranja"]
for i in range(len(frutas)):
   print(f"Fruta {i + 1}: {frutas[i]}")
```

Esta es la salida de ese código:

```
Fruta 1: manzana
Fruta 2: banana
Fruta 3: naranja
```

Acá estamos utilizando range(len(frutas)) para generar índices del 0 al 2, y con esos índices accedemos a los elementos de la lista frutas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Ejercicios Prácticos:

Suma de números naturales

Desarrolla una función que calcule la suma de los números naturales hasta un número dado utilizando un bucle for. La suma de los números naturales hasta el número n se define como la suma de todos los números enteros positivos desde 1 hasta n.
Por ejemplo, la suma de los números naturales hasta 6 es 1 + 2 + 3 + 4 + 5 + 6 = 21.

La función debe recibir un número entero n y devolver la suma de los números naturales hasta n.

Tips:

- ¡Usá range()!

Mostrar los códigos de productos ingresados

En un sistema de inventario, cada producto tiene un código que lo identifica. Escribí un programa que permita ingresar los códigos de 4 productos y luego indicá que se muestren uno por uno, junto con su posición en la lista. Ejemplo: si los códigos ingresados son "P001", "P002", "P003", "P004", el programa debe mostrar:

```
Producto 1: P001
Producto 2: P002
Producto 3: P003
Producto 4: P004
```

Tips:

- Utilizá un bucle for y range() para recorrer los códigos e imprimirlos.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad