«Talento Tech»

Iniciación a la

Programación con Python

Clase 06

Buenos Aires Ciudad

Clase N° 6 | Bucles For y Range()

Temario:

- Control de flujo: bucles for.
- Función range().
- Recorrer cadenas y listas con bucles while y for.

Objetivos de la Clase

Incorporar el uso del **bucle for**, una herramienta clave para recorrer estructuras de datos como **cadenas** y **listas** de manera eficiente. Aprender cómo este tipo de bucle permite simplificar y agilizar tareas repetitivas.

Explorar la función **range()**, que facilita la generación de secuencias numéricas, abriendo nuevas posibilidades para iterar sobre rangos definidos de valores. Este concepto será fundamental para abordar problemas de programación con soluciones más claras y estructuradas.

Comparar el comportamiento y las aplicaciones de los bucles for y while, profundizando en sus diferencias y usos específicos.

¡Sexto día en TalentoLab! 🚀

Llegás a **TalentoLab** con entusiasmo y, mientras disfrutás de tu café matutino, te reunís con Mariana, la Gerente de Proyectos. Mariana, satisfecha con tu progreso, te motiva a enfrentar un nuevo desafío que te ayudará a manejar mejor la validación y procesamiento de datos.

“Necesitamos crear una lista con los nombres de los clientes que vamos a procesar. Además, es necesario detectar si alguno de los nombres está en blanco y mostrar una alerta en esos casos. Luego, para los nombres válidos, nos aseguraremos que uno comience con una letra mayúscula y el resto en minúsculas.”

Más tarde, te reunís con Luis, el desarrollador senior. Junto a él, revisan los conceptos básicos que necesitarás para completar esta misión. Luis te muestra ejemplos prácticos de cómo detectar cadenas vacías y aplicar métodos para formatear los nombres adecuadamente.

Control de flujo: bucles 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 += 1
```

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

Bucle for: Iterando a través de cadenas.

Sabemos que 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.

Break

El uso de break en un bucle for, tal como ocurre cuando lo usamos con while, 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 **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.
```

¿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 su valor 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 versátil.

Ahora vamos a ver varios ejemplos para que te quede claro cómo funciona.

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.

range() es muy versátil, pero es posible que te dé un poco de trabajo recordar todas sus opciones. Para ayudarte, Luis te proporciona el siguiente resumen:

Uso de range()	Ejemplo	Descripción

range(fin)	`for i in range(5):`
	`  print(i)`
	`# Salida: 0, 1, 2, 3, 4`
		Genera una secuencia que empieza en 0 y termina en fin - 1.
		El paso predeterminado es 1.

range(inicio, fin)	`for i in range(2, 6):`
	`  print(i)`
	`# Salida: 2, 3, 4, 5`
		Genera una secuencia desde inicio hasta fin - 1.
		El paso predeterminado es 1.

range(inicio, fin, paso)	`for i in range(1, 10, 2):`
	`  print(i)`
	`# Salida: 1, 3, 5, 7, 9`
		Genera una secuencia desde inicio hasta fin - 1, avanzando de acuerdo con el valor del paso.

Comparación entre while y for.

Hemos visto cómo los bucles nos permiten manejar tareas repetitivas de manera eficiente. Ahora es importante que reflexionemos sobre una de las grandes ventajas del bucle for: su capacidad para simplificar tareas que, con un bucle while, pueden volverse más laboriosas.

Cuando usás un bucle while, generalmente tenés que controlar variables como contadores o índices de forma manual. Esto significa que necesitás inicializarlos, actualizarlos en cada iteración y asegurarte de que la condición del bucle sea correcta. Sin embargo, con un bucle for, podés recorrer directamente los elementos de una lista o una secuencia sin preocuparte por gestionar esas variables, ya que Python lo hace automáticamente por vos.

Esta simplicidad no solo reduce posibles errores, sino que también hace que tu código sea más fácil de leer y mantener. Aprender a identificar cuándo usar for en lugar de while es clave para optimizar tus programas y trabajar de manera más eficiente.

La sentencia **continue** en los bucles for funciona de la misma manera que en los bucles while. Cuando Python encuentra un continue dentro de un bucle for, salta inmediatamente al siguiente elemento de la lista o secuencia, ignorando cualquier código que quede por ejecutar en la iteración actual. Esto es útil cuando querés omitir ciertos elementos que no cumplen con una condición específica, pero querés que el bucle siga procesando el resto de los elementos normalmente.

El bucle for en la práctica.

El bucle for y la función range() son herramientas fundamentales para procesar y manejar múltiples datos, como los ingresos mensuales de un cliente. Con lo has aprendido, ya estás preparado para crear soluciones más claras y organizadas, lo que te permitirá trabajar con listas y realizar cálculos sin necesidad de gestionar manualmente contadores o índices como en el bucle while.

Por ejemplo, imaginá que necesitás calcular el promedio de una serie de valores. Usando un bucle for, podés recorrer una lista de números y sumar los valores de forma automática, mientras simplificás el código y reducís posibles errores.

Aunque no podés aplicar directamente este código al Trabajo Final Integrador, te da una idea de cómo trabajar con listas y condiciones para extraer información útil de los datos.

```
# Lista de ingresos mensuales
ingresos = [40000, 55000, 60000, 45000, 70000, 50000]

# Definimos un umbral
umbral = 50000

# Inicializamos un contador para los ingresos superiores al umbral.

contador_superiores = 0

# Usamos un bucle for para recorrer la lista
for ingreso in ingresos:
   if ingreso > umbral:
       # Incrementamos el contador si el ingreso es mayor al umbral
       contador_superiores += 1

# Mostramos el resultado
print(f"Cantidad de ingresos superiores a ${umbral}:
{contador_superiores}")
```

Este programa, con esos datos, genera la siguiente salida:

```
Cantidad de ingresos superiores a $50000: 3
```

Otro caso interesante donde el bucle for puede simplificar tareas es cuando necesitás realizar un seguimiento de ciertos elementos que cumplen con una condición específica. Por ejemplo, imaginá que estás trabajando con un sistema que procesa nombres de usuarios y usuarias registrados, y querés identificar cuáles comienzan con una letra en particular. Supongamos que querés encontrar todos los nombres que empiezan con la letra "A" y veamos cómo aplicar esta lógica:

```
# Lista de nombres de usuarios y usuarias
usuarios = ["Ana", "Luis", "Andrés", "María", "Alejandro", "Lucía"]

# Letra que queremos buscar
letra_buscada = "A"
```

```
print(f"Nombres que comienzan con la letra '{letra_buscada}':")

# Usamos un bucle for para recorrer la lista
for usuario in usuarios:
   if usuario.startswith(letra_buscada):  # Verificamos si el nombre
comienza con la letra buscada
       print(usuario)
```

Este código utiliza el método **.startswith()** para comprobar si cada nombre en la lista comienza con la letra especificada. La variable **letra_buscada** define qué letra estás buscando, y el bucle for se encarga de recorrer la **lista usuarios**. Si el nombre cumple la condición, se imprime en pantalla.

```
Nombres que comienzan con la letra 'A':
Ana
Andrés
Alejandro
```

Este enfoque no solo simplifica el manejo de datos, sino que también demuestra cómo el bucle for puede combinarse con otros métodos de cadenas para realizar tareas específicas, junto a listas y condiciones,para resolver problemas más complejos en tus proyectos.

Ejercicio Práctico.

Luis está haciendo un excelente trabajo. Al finalizar la jornada, lograste entender la sintaxis y utilidad de los bucles for, y lo súper útil que son para recorrer estructuras de datos. Con estas herramientas estás listo para resolver el pedido de Mariana:

¡Buenas tardes!
Espero que estés bien. Tu tarea es la siguiente:

- Crear una lista con los nombres de los clientes y clientas que vamos a procesar. Algunos nombres pueden estar en blanco por lo que debemos poder detectarlo.

- Recorrer la lista y mostrar el nombre de cada cliente, junto con su posición en la lista (por ejemplo, Cliente 1, Cliente 2, etc.).
- Si encuentras un o una cliente cuyo nombre sea una cadena en blanco, mostrar un mensaje de alerta indicando que ese dato no es válido.
- Para los nombres válidos, convertir cada uno a formato adecuado usando .capitalize(), de modo que siempre tengan la primera letra en mayúscula y el resto en minúscula.

¡Estoy segura de que harás un excelente trabajo!

Materiales y Recursos Adicionales:

Artículos:
IONOS: Bucle for en Python
FreeCodecamp.org: Explicación del búcle for de Python
El libro de Python: Bucle for

Videos:

Tutoriales sobre Ciencia y Tecnología: Bucles for en Python y una Tabla de multiplicar con bucle for

---

Preguntas para reflexionar:

1. ¿Qué diferencias encontrás entre el bucle while y el bucle for? Pensá en situaciones específicas sean de programación o de la vida cotidiana en las que elegirías uno sobre el otro y por qué.
2. Al recorrer listas con un bucle for, ¿qué ventajas encontrás frente a la gestión manual de índices en un bucle while? ¿Cómo influye esto en la legibilidad y mantenibilidad del código?
3. ¿Cómo podés aplicar lo aprendido en esta clase al desarrollo de sistemas más complejos, como el Trabajo Final Integrador? Pensá en validaciones, normalización de datos y la eficiencia que aportan los bucles.

Próximos pasos:

En la próxima clase, profundizaremos en el **manejo de listas** y exploraremos una nueva estructura de datos: **las tuplas**. Aunque ya conocés cómo crear y recorrer listas, vamos a enfocarnos en las operaciones más comunes que podés realizar con ellas, como agregar, eliminar y modificar elementos utilizando métodos específicos.

Además, aprenderás qué son las tuplas y cómo se diferencian de las listas. Las tuplas tienen usos específicos que las hacen ideales para ciertos escenarios donde los datos no deben cambiar.

Asegurate de repasar lo aprendido sobre bucles y listas, ya que estos conceptos serán indispensables para sacar el máximo provecho de las nuevas herramientas que veremos.

¡Seguimos avanzando!

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad