«Talento Tech»

Iniciación a la Programación con Python

CLASE 6

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 6 | Bucles while

Temario:

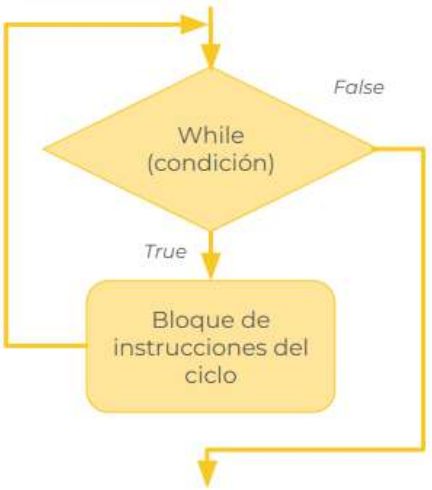
- Control de flujo: bucles while.
- Concepto y usos de los contadores.
- Concepto y usos de los acumuladores

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Control de flujo: bucles while.

Imaginá que estás desarrollando una aplicación en Python para tu Trabajo Final Integrador (PFI), donde querés asegurarte que las personas que usen tu programa ingresen datos válidos, como la cantidad de productos o el precio de un artículo. ¿Qué pasaría si ingresan un número negativo o un dato incorrecto? Ahí es donde entra en juego el **bucle while**, una herramienta súper útil que te permite que tu programa repita una acción hasta que se cumpla una condición.

El **bucle while** funciona como una conversación en la que el programa se pregunta a sí mismo algo una y otra vez hasta obtener una respuesta que cumpla (o no) con lo que espera. Mientras la condición que definas siga siendo verdadera (True), el bucle seguirá ejecutándose. Una vez que esa condición ya no se cumpla, el bucle se detiene y el programa continúa con el resto de las instrucciones.



Vamos a ver un caso concreto donde podríamos necesitar un bucle while. Imaginate que querés pedirle al usuario que ingrese la cantidad de productos que va a agregar al inventario de la tienda o comercio que tenés en mente. La cantidad debe ser un número positivo pero, como sabemos, las personas pueden cometer errores al ingresar datos por lo que necesitamos asegurarnos de que el número sea válido. Con un bucle while, podés

hacer que el programa siga pidiendo la cantidad hasta que la persona ingrese un valor mayor que 0. En este caso, el bucle se preguntará: “¿Es el valor ingresado mayor a cero?”, y en función de la respuesta ejecutará una u otra acción.

Ejemplo 1: Solicitar y validar un dato.

En este ejemplo, el programa le pide a nuestra persona usuaria que ingrese una cantidad. Si el número ingresado es menor o igual a 0 el programa le muestra un mensaje explicando que el valor no es válido y vuelve a pedir la cantidad. Este proceso se repite hasta que la persona ingresa un número mayor a 0. Una vez que el dato ingresado es correcto, el bucle se detiene y el programa continúa.

```
cantidad = int(input("Ingresá la cantidad de productos: "))

while cantidad <= 0:
   print("La cantidad debe ser mayor que 0.")
   cantidad = int(input("Por favor, ingresá nuevamente la cantidad de
productos: "))

print("Has ingresado una cantidad válida:", cantidad)
```

Este tipo de validación es súper importante cuando querés asegurarte de que los datos que está ingresando el usuario sean correctos y útiles. En el contexto del Proyecto Final Integrador, esto podría aplicarse cuando el sistema de inventario necesita validar que las cantidades de los productos sean siempre positivas.

Ejemplo 2: Implementar un menú interactivo

Otra aplicación muy útil de los **bucles while** es cuando queremos crear un menú interactivo que permita al usuario seleccionar diferentes opciones. En este caso, podemos usar un bucle while para mantener el menú activo hasta que el usuario decida salir. Esto va a ser fundamental en tu Trabajo Final Integrador cuando implementes la interfaz para gestionar el inventario.

Vamos a ver un ejemplo donde el menú sigue mostrando opciones hasta que el usuario elige salir:

```
opcion = 0

while opcion != 3:
   print("\nMenú de Opciones")
   print("1. Ver productos")
   print("2. Agregar un producto")
   print("3. Salir")

   opcion = int(input("Seleccioná una opción: "))

   if opcion == 1:
       print("Mostrando productos...")
   elif opcion == 2:
       print("Agregando un producto...")
   elif opcion == 3:
       print("Saliendo del menú.")
   else:
       print("Opción no válida. Por favor, ingresá una opción entre 1 y
3.")
```

En este ejemplo, el menú sigue mostrando las opciones hasta que la persona selecciona la opción 3, que corresponde a "Salir". El bucle while se asegura de que el menú siga apareciendo una y otra vez, a menos que se elija específicamente salir. La condición opcion != 3 es clave aquí, porque != significa *"distinto de".* Mientras la variable opcion sea distinta de 3, el bucle seguirá ejecutándose. En otras palabras, el menú solo se detiene cuando la persona selecciona el número 3, que corresponde a salir.

Otro detalle importante es el uso de \n en el primer print(). Este carácter especial indica un salto de línea, lo que significa que agrega una línea en blanco antes del título "Menú de opciones". Esto se hace para que el menú se vea más limpio y separado de las líneas anteriores, mejorando la legibilidad del texto cuando se muestra repetidamente en pantalla.

Si la persona ingresa una opción incorrecta (algo que no esté entre 1 y 3), el programa le muestra un mensaje de error y el menú vuelve a aparecer. Esto es posible gracias al bucle, que "espera" que se ingrese una opción válida.

Este tipo de bucle es súper útil para crear una interfaz interactiva, donde el programa permanece en espera de que quien lo use decida qué hacer. Además, es importante para mantener el control del flujo dentro de una aplicación como la que vas a desarrollar en tu Trabajo Final Integrador. El menú no solo te permite navegar por las diferentes opciones, sino que también se asegura de que el programa solo termine cuando la persona realmente quiera salir, gracias a la condición != 3 que controla el bucle.

¿Por qué usar bucles while?

El bucle while te permite repetir acciones en tu programa de manera controlada. En lugar de escribir el mismo código una y otra vez, el bucle hace el trabajo por vos hasta que se cumpla una condición que definiste. Es como tener una puerta giratoria: el programa sigue girando hasta que le digas que pare. En los ejemplos que vimos, usamos while para asegurarnos de que los datos ingresados sean correctos y para implementar un menú interactivo, pero hay un montón de aplicaciones más.

El bucle while te va a acompañar en muchos aspectos de tu proyecto, ya que te permite construir programas más dinámicos e interactivos. Con while, el control del flujo de tu código está en tus manos y eso abre la puerta a un montón de posibilidades.

Contadores.

El contador es uno de los conceptos más básicos y útiles en programación. Básicamente, un contador es una variable que suma (o resta) de uno en uno cada vez que ocurre algo, como cuando se repite una acción dentro de un bucle. La idea es llevar un registro de cuántas veces ha sucedido algo.

Los contadores son perfectos cuando querés saber, por ejemplo, cuántos productos ingresaron al sistema, cuántas veces una persona intentó hacer algo, o cuántos pasos se dieron en un proceso. El contador cambia su valor dentro de un bucle, incrementando su valor por una cantidad fija, generalmente uno.

¿Cómo funciona un contador?

Lo primero que hacés es inicializar el contador en cero (o en otro valor si es necesario). Luego, dentro del bucle, cada vez que sucede la acción que estás contando le sumás uno al contador. Cuando el bucle termina, el contador va a tener guardada la cantidad total de veces que se realizó esa acción.

¿Dónde se usa un contador?

Los contadores son súper útiles para:

- **Contar iteraciones en un bucle:** Saber cuántas veces se repitió una acción.
- **Contar errores o intentos:** Como en el ejemplo anterior donde contás cuántos intentos fallidos hubo antes de un valor correcto.

- **Controlar la cantidad de veces que se ejecuta algo:** Por ejemplo, para limitar un bucle a un número determinado de repeticiones.

Vamos a verlo en acción con un ejemplo práctico relacionado con el Trabajo Final Integrador (TFI).

Ejemplo 1: Contar intentos de ingreso de datos.

Imaginá que necesitás un sistema que permita al usuario ingresar cuántos productos va a agregar al inventario. Si ingresa un valor incorrecto (por ejemplo, una cantidad negativa), queremos contar cuántas veces cometió un error antes de ingresar un número válido. Para esto, usamos un contador:

```
intentos = 0  # Inicializamos el contador

cantidad = int(input("Ingresá la cantidad de productos: "))

while cantidad <= 0:
   # Aumentamos el contador por cada error
   intentos = intentos + 1

   print("Error: La cantidad debe ser mayor que 0.")
   cantidad = int(input("Por favor, ingresá nuevamente la cantidad de
productos: "))

print(f"Ingresaste un valor válido después de {intentos} intentos.")
```

En este caso, el contador es la variable **intentos.** Cada vez que la persona comete un error (ingresa un valor menor o igual a 0), sumamos uno al contador con intentos += 1. De esta

forma, cuando el bucle termina, el programa puede mostrar cuántas veces se ingresó un valor incorrecto antes de finalmente dar con uno correcto.

Ejemplo 2: Contar la cantidad de productos en un inventario.

Otro caso muy útil de los contadores es cuando querés llevar un registro de cuántos productos hay en total en el inventario de una tienda. Cada vez que agregás un producto, el contador aumenta.

```
total_productos = 0  # Inicializamos el contador

for dia in range(3):  # Supongamos que ingresamos productos durante 3 días
   print("Día",dia + 1, ":")
   productos_agregados = int(input("Productos ingresados en el día: "))
   total_productos += 1  # Cada vez que agregamos productos, sumamos 1 al contador

print("Se ingresaron",total_productos,"productos en total al inventario.")
```

Este ejemplo si bien es simple pero resulta muy efectivo para entender el funcionamiento de un contador. Cada vez que agregamos productos el contador aumenta de a uno, lo que nos permite llevar la cuenta de cuántas veces se realizó esa acción.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Por qué es útil un contador?

El contador es una herramienta simple pero poderosa. Te permite llevar un registro de cuántas veces ocurre algo, y esta información es importante en la mayoría de las aplicaciones. Desde contar intentos, productos agregados, hasta controlar el número de repeticiones de un bucle, los contadores son indispensables para que tu código sea dinámico y efectivo.

En el contexto del Trabajo Final Integrador vas a encontrar muchas situaciones donde necesitás contar cosas: **desde cuántos productos se ingresan al inventario hasta cuántos movimientos se hicieron en un día.** Un contador te va a permitir manejar estos escenarios correctamente.

Acumuladores

Si el contador te sirve para contar cuántas veces ocurre algo, el acumulador opera de una manera similar pero, en lugar de sumar de a uno, lo que hace es acumular un valor que puede ir variando. Mientras que el contador te dice cuántas veces pasó algo, el acumulador te dice cuánto se acumuló a lo largo del tiempo o a lo largo de una serie de acciones. El acumulador es la herramienta ideal cuando necesitás llevar un registro de algo que no solo ocurre una vez, sino que también involucra sumar, restar o **manipular un valor de forma más compleja.**

Imaginemos el contexto del Trabajo Final Integrador (TFI), donde estás desarrollando una aplicación para gestionar un inventario. Si cada día ingresan diferentes cantidades de productos, un acumulador te permitirá llevar el total exacto de productos que ingresaste al inventario durante varios días. Así como el contador suma de a uno, el acumulador suma el valor que vos le asignes en cada iteración o “vuelta” del bucle.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Cómo funciona un acumulador?

Al igual que con un contador, lo primero que hacés es inicializar el acumulador, normalmente en 0. Luego, dentro de un bucle, le vas sumando un valor específico que puede ser variable. El acumulador guarda ese valor actualizado en cada iteración del bucle, permitiéndote llevar un registro totalizado al final.

Ejemplo 1: Sumar la cantidad total de productos agregados al inventario

En este caso, total_productos es nuestro acumulador. Cada día, sumamos la cantidad de productos que ingresaron en el inventario al valor total que ya teníamos guardado. Con el operador +=, vamos acumulando el valor que la persona ingresa permitiendo que, al final del ciclo, tengamos la cantidad total de productos acumulada.

```
total_productos = 0  # Inicializamos el acumulador

# Simulamos el ingreso de productos durante 3 días
for dia in range(3):
   print("Día", dia + 1, ":")
   productos_ingresados = int(input("Productos ingresados en el día:
"))
   # Vamos sumando la cantidad de productos al acumulador
   total_productos = total_productos + productos_ingresados

print("El total de productos agregados al inventario es:",
total_productos)
```

¿Dónde se usa un acumulador?

El acumulador es súper útil en situaciones donde necesitás sumar valores de manera progresiva. Mientras que el contador solo suma de a uno, el acumulador suma siguiendo el criterio que establezcas, lo que lo hace mucho más flexible para manejar cantidades que varían.

Por ejemplo:

- **Suma de productos:** Llevar el total de productos agregados a un inventario, como en el ejemplo anterior.
- **Cálculo de precios:** Sumar el precio de los productos en un carrito de compras.
- **Sumar cantidades de ventas:** Acumular el total de ventas realizadas en un día o una semana.

Ejemplo 2: Acumular caracteres de una cadena

El acumulador también puede usarse con cadenas de texto. Supongamos que querés construir una cadena letra por letra, acumulando las letras en un texto final. Esto es útil si querés, por ejemplo, mostrar el nombre de un producto de manera dinámica o armar un mensaje personalizado.

```
producto = "Notebook"  # Cadena original
nombre_acumulado = ""  # Inicializamos el acumulador

# Vamos a recorrer cada letra de la cadena y acumularla
for letra in producto:
    # Acumulamos letra por letra
    nombre_acumulado = nombre_acumulado  + letra
    print("Nombre parcial:", nombre_acumulado)

print("Nombre completo:", nombre_acumulado)
```

En este caso, usamos el acumulador nombre_acumulado para ir sumando letra por letra de la palabra "Notebook". A medida que el bucle recorre las letras de la cadena original, las va añadiendo al acumulador, mostrando cómo se construye el nombre del producto letra por letra.

¿Por qué es útil un acumulador?

El acumulador es fundamental cuando querés llevar un registro progresivo de un valor que puede cambiar en cada paso. Ya sea que estés sumando productos al inventario, calculando el total de ventas o construyendo un mensaje a partir de texto, esta herramienta te permite manejar esos valores de manera simple.

Ejercicios prácticos:

1) Control de stock de productos:

Desarrollá un programa que permita a quienes interactúen con él ingresar el nombre de varios productos y la cantidad en stock que hay de cada uno. El programa debe seguir pidiendo que ingrese productos hasta que el usuario decida salir, ingresando "salir" como nombre de producto. Después de que el bucle termine el programa debe mostrar la cantidad total de productos ingresados.

Tips:

- Vas a necesitar un contador.
- Tené presente las estructuras condicionales.

2) Validación de precios de productos:

Escribí un programa que permita ingresar el precio de un producto, pero que sólo acepte valores mayores a 0. Si la persona ingresa un valor inválido (negativo o cero), el programa debe mostrar un mensaje de error y solicitar nuevamente el valor hasta que se ingrese uno válido. Al final, el programa debe mostrar el precio aceptado.

Tips:

- Antes de empezar, pensá si es necesario usar contadores o acumuladores.
- Recordá que input() siempre devuelve cadenas de caracteres.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad