«Talento Tech»

Iniciación a la

Programación con Python

Clase 05

Buenos Aires Ciudad

Clase N° 5 | Bucles While

Temario:

- Control de flujo: bucles while.
- Concepto y uso de contadores y acumuladores.
- Sentencias break y continue.
- Introducción a listas como estructuras para almacenar múltiples valores.

Objetivos de la Clase

En esta clase, aprenderás a utilizar los **bucles while**, una herramienta fundamental para repetir bloques de código bajo ciertas condiciones. Verás cómo relacionar las estructuras condicionales que ya manejás con el control de flujo mediante bucles, lo que te permitirá crear programas más dinámicos y eficientes.

También vas a incorporar los conceptos de **contadores** y **acumuladores**, entendiendo cómo usarlos para realizar cálculos iterativos, procesar datos y resolver problemas que requieran seguimiento de valores acumulativos. Además, explorarás las sentencias **break** y **continue**, y descubrirás cómo estas pueden modificar el comportamiento de un bucle según las necesidades de tu programa.

Finalmente, comenzarás a trabajar con listas, una estructura poderosa para almacenar múltiples valores en un solo lugar. Aprenderás a crearlas, asignarles valores y recuperarlos, preparando el camino para gestionar datos más complejos en tus futuros proyectos.

Otra mañana en TechLab 🚀

La primera reunión del día en **TalentoLab** te deja claro que todavía tenés un largo camino por delante. Hasta ahora, hiciste un gran trabajo con los datos de una persona. Pero en el mundo real, se necesitan procesar datos de muchas personas o incluso durante varios períodos. Por eso, en la reunión de equipo Mariana te explica que necesitamos un software que ayude a registrar y calcular información financiera básica para nuestras y nuestros clientes.

“Necesitamos registrar los ingresos mensuales de una persona durante 6 meses. Además, hay que validar que éstos sean números positivos. Y, por supuesto, calcular el importe total de los ingresos acumulados durante los 6 meses.“

Como siempre, vas a necesitar aprender algunas nuevas características y estructuras de control para poder realizar esta tarea. En ese momento, se abre la puerta de la sala de reuniones y entra Luis, dispuesto como siempre a ayudarte. ¡Allá vamos!

Control de flujo: bucles while

Hasta ahora, aprendiste a manejar el flujo de un programa utilizando **estructuras condicionales** como **if**, **elif** y **else**. Estas herramientas son esenciales para tomar decisiones dentro de un programa, pero ¿qué pasa si necesitás que una acción se repita varias veces? Acá es donde los **bucles while** entran en juego.

Un bucle while permite repetir un bloque de código mientras se cumpla una **condición** específica. Esto significa que el programa va a evaluar esa condición antes de ejecutar cada iteración del bucle. Si la condición es **True**, el código dentro del bucle se ejecuta; si es **False**, el bucle termina y el programa continúa con las instrucciones que siguen.

La sintaxis básica de un bucle while es la siguiente:

```
while condición:
   # Código que se ejecuta mientras la condición sea True
```

Es importante que dentro del bucle haya algo que eventualmente haga que la condición sea False, porque si no, el programa entraría en lo que llamamos un **"bucle infinito"**. Un bucle infinito sigue ejecutándose sin detenerse y puede hacer que tu programa deje de responder.

Veamos un caso sencillo para entender cómo funciona:

```
contador = 1

while contador <= 5:
   print(f"Este es el intento número {contador}.")
   contador += 1  # Incrementamos el valor de contador

print("Bucle terminado.")
```

En este ejemplo:

- El bucle comienza con la variable **contador** igual a 1.
- La condición **contador <= 5** se evalúa al inicio de cada iteración.
- Dentro del bucle, imprimimos un mensaje con el valor actual de contador y luego lo incrementamos en 1 usando contador = contador + 1.

- Cuando contador llega a 6, la condición contador <= 5 se vuelve **False**, y el bucle termina.

Salida del programa:

```
Este es el intento número 1.
Este es el intento número 2.
Este es el intento número 3.
Este es el intento número 4.
Este es el intento número 5.
Bucle terminado.
```

Luis: "***contador += 1*** *es una versión abreviada de* ***contador = contador + 1****, que hace lo mismo, pero es más concisa y se usa comúnmente en Python. Ambas son equivalentes, y podés usar cualquiera según tu preferencia o el nivel de detalle que quieras mostrar en tu código.*"

¿Cuándo usar un bucle while?

Un bucle while es ideal cuando no sabés de antemano cuántas veces se debe repetir una acción. Por ejemplo:

- Pedir datos al usuario hasta que ingrese algo válido.
- Realizar cálculos hasta que se alcance un determinado resultado.
- Procesar datos en una lista mientras queden elementos por analizar.

En este ejemplo, vamos a pedirle a la persona que ingrese su nombre. Sin embargo, queremos asegurarnos de que no deje el campo vacío. Usaremos un bucle while para repetir la solicitud hasta que el usuario ingrese un nombre válido:

```
# Inicializamos una variable vacía para almacenar el nombre
nombre = ""

# Usamos un bucle while para seguir pidiendo un nombre hasta que
# el usuario ingrese algo válido

while nombre == "":
```

```
    # Solicitamos al usuario que ingrese su nombre.
    # .strip() elimina espacios en blanco al principio y al final

    nombre = input("Ingresá tu nombre: ").strip()

    if nombre == "":  # Verificamos si el nombre está vacío
        print("El nombre no puede estar vacío. Intentá de nuevo.")

# Si el bucle termina, mostramos un mensaje con el nombre ingresado
print(f"¡Hola, {nombre}! Gracias por ingresar tu nombre.")
```

Este programa comienza inicializando la variable **nombre** con una cadena vacía. Luego, establece un bucle while que seguirá ejecutándose mientras nombre esté vacío. Dentro del bucle, se solicita a la persona que ingrese su nombre. La función **.strip()** se usa para eliminar cualquier espacio en blanco al principio o al final de la entrada, asegurando que incluso si se ingresaron ślo espacios, la validación detecte que no es un dato válido.

Si la entrada está vacía, el programa muestra un mensaje indicando que vuelva a intentarlo. Una vez que se ingresa un nombre válido, el bucle se detiene y el programa muestra un saludo personalizado. Esto garantiza que el flujo del programa continúe solo cuando los datos ingresados sean adecuados.

Contadores en bucles while.

Un **contador** es una variable que se utiliza dentro de un bucle para realizar un seguimiento de cuántas veces se repite. En los bucles while, los contadores son fundamentales porque nos permiten controlar cuántas iteraciones se realizan, detener el bucle en el momento correcto o realizar acciones específicas en cada repetición.

Por lo general, un contador se inicializa antes del bucle, se actualiza en cada iteración (normalmente sumándole 1) y se utiliza como parte de la condición del bucle o para tomar decisiones dentro de él. Vamos a analizar un par de ejemplos prácticos para entender cómo funcionan.

Ejemplo 1: Mostrar los números del 1 al 5
Este ejemplo utiliza un contador para generar y mostrar una secuencia de números desde 1 hasta 5.

```
# Inicializamos el contador en 1
contador = 1

# Creamos un bucle que se ejecuta mientras el contador
```

```
# sea menor o igual a 5
while contador <= 5:
    # Mostramos el valor actual del contador
    print(f"Número: {contador}")

    # Incrementamos el contador en 1
    contador += 1

# Mensaje final una vez que el bucle termina
print("Bucle terminado.")
```

En este caso, el contador comienza en 1 y se incrementa en cada iteración con **contador += 1**. El bucle se ejecuta mientras la condición **contador <= 5** sea verdadera. Cuando el contador llega a 6, la condición se vuelve falsa y el bucle termina.

```
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Bucle terminado.
```

Este tipo de bucle es ideal para ejecutar una acción un número fijo de veces.

Ejemplo 2: Solicitar un nombre de usuario con un máximo de 3 intentos
En este ejemplo, utilizamos un contador para limitar la cantidad de intentos que tiene la persona para ingresar su nombre de usuario.

```
# Inicializamos el contador en 0
intentos = 0

# Establecemos el máximo de intentos permitidos
max_intentos = 3

# Usamos un bucle que se detendrá si el usuario
```

```
# ingresa un nombre válido o si se agotan los intentos
while intentos < max_intentos:
   # Solicitamos al usuario que ingrese su nombre
   nombre = input("Ingresá tu nombre de usuario: ").strip()

   # Verificamos si el nombre ingresado no está vacío
   if nombre != "":
       print(f"Bienvenido/a, {nombre}!")  # Mensaje de éxito
       break  # Salimos del bucle si se ingresa un nombre válido
   else:
       print("El nombre no puede estar vacío. Intentá de nuevo.")

   # Incrementamos el contador de intentos
   intentos += 1

# Verificamos si se agotaron los intentos
if intentos == max_intentos:
   print("Se agotaron los intentos. Intente más tarde.")
```

En este programa, la variable intentos actúa como contador y se incrementa cada vez que se ingresa un nombre no válido. Si se ingresa un nombre válido antes de alcanzar el máximo de intentos (max_intentos), el programa muestra un mensaje de bienvenida y sale del bucle con **break**. Si se alcanzan los 3 intentos y no se ingresó un nombre válido, el programa finaliza mostrando un mensaje de error. Esta es una salida posible del programa:

```
Ingresá tu nombre de usuario:
El nombre no puede estar vacío. Intentá de nuevo.
Ingresá tu nombre de usuario:
El nombre no puede estar vacío. Intentá de nuevo.
Ingresá tu nombre de usuario: Pedro
Bienvenido/a, Pedro!
```

Este tipo de uso de contadores es especialmente útil en programas que requieren limitar la cantidad de intentos para realizar una acción, como autenticarse o completar un formulario.

Break.

La sentencia **break** en Python se utiliza para terminar un bucle de forma inmediata, sin esperar a que se cumpla la condición del bucle. Esto significa que, cuando se ejecuta break, el programa salta automáticamente a la siguiente línea de código después del bucle, ignorando cualquier iteración restante.

Es especialmente útil cuando querés salir de un bucle antes de que su condición se vuelva False, por ejemplo, al encontrar un resultado deseado o al detectar una situación específica que hace innecesario continuar.

Luis: "*En muchos casos es posible evitar el uso de break modificando la condición del bucle. Aunque break es una herramienta válida, muchas veces se puede lograr el mismo resultado con un diseño cuidadoso del flujo lógico del programa*."

Por ejemplo, podemos reescribir el programa de solicitud de nombre de usuario con un máximo de tres intentos sin usar break, ajustando la condición del bucle para que se detenga automáticamente cuando se ingrese un nombre válido o se alcancen los intentos permitidos.

```
# Inicializamos el contador en 0
intentos = 0

# Establecemos el máximo de intentos permitidos
max_intentos = 3

nombre = ""  # Variable para almacenar el nombre
while intentos < max_intentos and nombre == "":
    # Solicitamos al usuario que ingrese su nombre
    nombre = input("Ingresá tu nombre de usuario: ").strip()

    # Verificamos si el nombre ingresado está vacío
    if nombre == "":
        print("El nombre no puede estar vacío. Intentá de nuevo.")
        intentos += 1  # Incrementamos el contador de intentos

# Verificamos si se ingresó un nombre válido o si se agotaron los
intentos
if nombre != "":
    # Mensaje de éxito
    print(f"Bienvenido/a, {nombre}!")
```

```
else:
   # Mensaje de error
   print("Se agotaron los intentos. Intente más tarde.")
```

Esta es una salida posible del programa anterior:

```
Ingresá tu nombre de usuario:
El nombre no puede estar vacío. Intentá de nuevo.
Ingresá tu nombre de usuario: Ana
Bienvenido/a, Ana!
```

La sentencia continue.

La sentencia **continue** se utiliza dentro de bucles para omitir el resto del código en la iteración actual y pasar directamente a la siguiente iteración. A diferencia de **break**, que termina el bucle por completo, **continue** sólo salta la ejecución restante en la iteración actual, permitiendo que el bucle continúe funcionando como está diseñado.

Esto es útil cuando querés evitar ciertas condiciones específicas dentro del bucle, pero sin detener todo el proceso.

En este ejemplo, vamos a pedirle al usuario que ingrese un número positivo. Si el número ingresado es negativo, utilizaremos la sentencia **continue** para saltar el resto del código de esa iteración y pedir otro número.

```
# Inicializamos una variable para controlar el bucle
numero = 0

print("Ingresá números positivos para sumarlos. Ingresá 0 para
terminar.")

# Usamos un bucle while que se ejecuta hasta que el usuario ingrese 0
suma = 0
while True:
   # Solicitamos al usuario un número
   numero = int(input("Ingresá un número: "))

   # Verificamos si el número es negativo
   if numero < 0:
```

```
        print("El número es negativo, se ignora. Intentá de nuevo.")
        continue  # Saltamos el resto del código en esta iteración

    # Si el número es 0, salimos del bucle
    if numero == 0:
        break

    # Sumamos el número positivo al total
    suma += numero

# Mostramos el resultado final
print(f"La suma de los números positivos es: {suma}")
```

El programa solicita que ingrese números positivos para sumarlos. Si se ingresa un número negativo, el programa utiliza la sentencia continue para saltar el resto del código en esa iteración, evitando que el número negativo se sume al total. Si se ingresa 0, el programa utiliza break para salir del bucle, ya que ese es el indicador de que se terminó de ingresar datos.

Mientras se sigan ingresando números positivos, estos se suman en la variable suma. Finalmente, el programa muestra el total de los números positivos ingresados.

```
Ingresá números positivos para sumarlos. Ingresá 0 para terminar.
Ingresá un número: 10
Ingresá un número: -1
El número es negativo, se ignora. Intentá de nuevo.
Ingresá un número: 20
Ingresá un número: 5
Ingresá un número: 0
La suma de los números positivos es: 35
```

Acumulador.

En el ejemplo anterior, la variable **suma** cumple el rol de un acumulador. Un acumulador es una variable que se utiliza para almacenar un valor que va aumentando (o disminuyendo) de manera progresiva a lo largo de las iteraciones de un bucle. Generalmente, se inicializa con un valor base (por ejemplo, 0 para una suma o 1 para una multiplicación), y luego se le van agregando o combinando valores durante cada iteración del bucle.

El propósito de un acumulador es mantener un registro de un cálculo o un resultado parcial que depende de los datos procesados en el bucle.

Introducción a las listas como estructuras para almacenar múltiples valores.

Una lista es una **estructura de datos** en Python que te permite almacenar múltiples valores en un solo lugar. A diferencia de las variables simples, que contienen un solo dato, las listas pueden contener varios elementos, como números, cadenas o incluso otras listas. Las listas son extremadamente útiles cuando querés trabajar con colecciones de datos, ya que te permiten agrupar información relacionada y acceder a ella de manera sencilla.

Las listas en Python se definen utilizando corchetes [], y los elementos se separan por comas. Podés agregar, modificar o eliminar elementos de una lista, lo que las convierte en una herramienta muy flexible para manejar datos en tus programas, y que pronto aprenderemos a utilizar a fondo. Por ahora, veremos cómo se crean y cómo podemos utilizarlas con while.

Creación de listas y acceso a sus elementos.

Antes de usar listas dentro de un bucle while, es fundamental entender cómo se crean y cómo acceder a los valores que contienen utilizando índices. Una lista es simplemente una colección ordenada de elementos que podés manipular fácilmente. Los elementos de una lista están numerados por su posición, empezando desde el índice 0.

Creación de listas
Para crear una lista en Python, usás corchetes [] y separás los elementos con comas. Una lista puede contener cualquier tipo de dato, como números, cadenas o incluso otras listas.

```
# Ejemplo de lista de números
numeros = [10, 20, 30, 40, 50]

# Ejemplo de lista de cadenas
nombres = ["Ana", "Luis", "Carlos"]

# Lista vacía
lista_vacia = []
```

Acceso a los elementos de una lista
Podés acceder a los elementos de una lista utilizando su índice. El primer elemento tiene el índice 0, el segundo el índice 1, y así sucesivamente.

```
# Lista de ejemplo
frutas = ["manzana", "banana", "cereza"]

# Accedemos a los elementos por índice
print(frutas[0])  # Salida: manzana
print(frutas[1])  # Salida: banana
print(frutas[2])  # Salida: cereza
```

Si intentás acceder a un índice que no existe, Python genera un error. Por ejemplo, frutas[3] daría error porque la lista tiene solo tres elementos (0, 1, y 2).

El uso de índices en las listas seguramente te resulta familiar, ya que comparte muchas de sus características con las cadenas de caracteres.

Índices negativos
Python también permite usar índices negativos para acceder a los elementos desde el final hacia el principio. Por ejemplo, el índice -1 representa el último elemento de la lista.

```
# Lista de ejemplo
frutas = ["manzana", "banana", "cereza"]

# Acceso usando índices negativos
print(frutas[-1])  # Salida: cereza (último elemento)
print(frutas[-2])  # Salida: banana
```

Con este conocimiento, podés entender cómo manipular y trabajar con los elementos de una lista. Ahora, veamos cómo integrar las listas con los bucles. Por ejemplo, podemos usarlos para procesar valores de una lista en un bucle while:

```python
# Lista de números
numeros = [10, 20, 30, 40, 50]

# Inicializamos un índice para recorrer la lista
indice = 0

# Usamos un bucle while para iterar sobre la lista
# len(numeros) = cantidad de elementos en la lista
while indice < len(numeros):
    # Mostramos el valor actual
    print(f"El valor en el índice {indice} es: {numeros[indice]}")

    # Incrementamos el índice para pasar al siguiente elemento
    indice += 1

print("Fin del bucle.")
```

En el código proporcionado, primero creamos una lista llamada **números** que contiene los datos que queremos procesar. A continuación, inicializamos un **índice** que usaremos para llevar el control de la posición actual dentro de la lista que estamos procesando.

El bucle **while** comienza con la condición **indice < len(numeros)**, lo que significa que seguirá ejecutándose mientras el valor de **indice** sea menor que la longitud de la lista numeros.

Dentro del bucle, en cada iteración, accedemos al elemento de la lista que corresponde al índice actual utilizando **numeros[indice]**. Luego, mostramos en pantalla el valor de ese elemento junto con su índice correspondiente mediante la función **print()**.

Después de imprimir el valor, incrementamos el valor de indice en 1 utilizando **indice += 1** para avanzar al siguiente elemento de la lista en la próxima iteración del bucle. El bucle continúa este proceso hasta que indice ya no sea menor que la longitud de la lista. Finalmente, se imprime el mensaje "Fin del bucle." para indicar que el procesamiento ha concluido. ¡Probalo!

Ejercicio Práctico.

El día ha sido largo, pero Luis ha conseguido que entiendas un montón de conceptos nuevos. Conceptos poderosos, que seguramente vas a tener que profundizar pronto. Pero lo importante es que con todo lo aprendido hoy, ya puedes resolver el pedido que Mariana te hizo a la mañana. Como es usual, te lo ha formalizado mediante un correo electrónico:

¡Buenas tardes!

Necesitamos un software que ayude a registrar y calcular información financiera básica para nuestros y nuestras clientes. Tu tarea para esta semana es la siguiente:

- Registrar los ingresos mensuales de un cliente durante 6 meses. Usá un bucle while para solicitar el ingreso de cada mes.
- Validar que los ingresos sean números positivos. Si se ingresa un valor negativo, mostrá un mensaje indicando que el valor no es válido y volvé a pedir el dato.
- Calcular el ingreso total acumulado durante los 6 meses. Mostrá este resultado al final del programa.

¡Estoy segura de que harás un excelente trabajo!

Materiales y Recursos Adicionales:

Artículos:

IONOS: Bucle while en Python

FreeCodecamp.org: Explicación del búcle While de Python

El libro de Python: Bucle while

Videos:

Tutoriales sobre Ciencia y Tecnología: Bucles while en Python y Acumulador en bucles while

---

Preguntas para reflexionar:

1. ¿Cómo podés asegurarte de que un bucle while no se convierta en un bucle infinito? Pensá en la importancia de actualizar correctamente las variables involucradas en la condición del bucle.

2. Reflexioná sobre las diferencias entre usar las sentencias break y continue. ¿En qué situaciones podrían ser más útiles cada una de estas herramientas?
3. Al recorrer una lista con un bucle while, ¿qué pasos considerás esenciales para garantizar que cada elemento sea procesado correctamente y que el bucle finalice en el momento adecuado?

Próximos pasos:

En la siguiente clase, ampliarás tu conocimiento sobre cómo manejar algunas tareas repetitivas de manera más directa y eficiente con el **bucle for**. Esta herramienta te permitirá recorrer estructuras como cadenas o listas sin necesidad de controlar manualmente cada iteración, simplificando el código y haciéndolo más legible.

Además, exploraraemos la función **range()** y profundizaremos en las **diferencias entre los bucles while y for**. Por último, introduciremos las **listas**, una estructura poderosa para almacenar y manipular múltiples valores.

¡Seguimos avanzando hacia programas más eficientes y completos!


Buenos Aires
aprende
Buenos
Aires
Ciudad