Ciencia de **DATOS**

Estructuras de control de flujo
Ciclos e Iteradores

¿Qué es iteración?

La iteración significa ejecutar el mismo bloque de código una y otra vez, potencialmente muchas veces. Una estructura de programación que implementa la iteración se denomina loop, bucle o ciclo.

Tipos de iteración

En programación, hay dos tipos de iteraciones, indefinidas y definidas:

Con la iteración indefinida, la cantidad de veces que se ejecuta el ciclo no se especifica explícitamente de antemano. Más bien, el bloque designado se ejecuta repetidamente siempre que se cumpla alguna condición.

Iteración definida

Con una iteración definida, el número de veces que se ejecutará el bloque designado se especifica explícitamente en el momento en que comienza el ciclo.

El ciclo while

Veamos cómo se usa la instrucción while de Python para construir ciclos. Comenzaremos simple y embelleceremos a medida que avanzamos.

El formato de un ciclo while básico se muestra a continuación:

```
while <expresión>:
    <sentencia(s)>
```

El ciclo while

<sentencia(s)> representa el bloque que se ejecutará repetidamente, normalmente se conoce como cuerpo del ciclo o bucle. Esto se denota con indentación o sangría, al igual que en una sentencia if.

El ciclo while

La expresión de control, <expresión>, generalmente involucra una o más variables que se inicializan antes de iniciar el ciclo y luego se modifican en algún lugar del cuerpo del ciclo.

Cuando se encuentra un ciclo o bucle **while**, <expresión> se evalúa primero en contexto booleano. Si es verdadero, se ejecuta el cuerpo del ciclo. Luego, <expresión> se verifica nuevamente y, si aún es cierto, el cuerpo se ejecuta nuevamente. Esto continúa hasta que <expresión> se vuelve falso, momento en el cual la ejecución del programa continúa con la primera declaración más allá del cuerpo del ciclo.

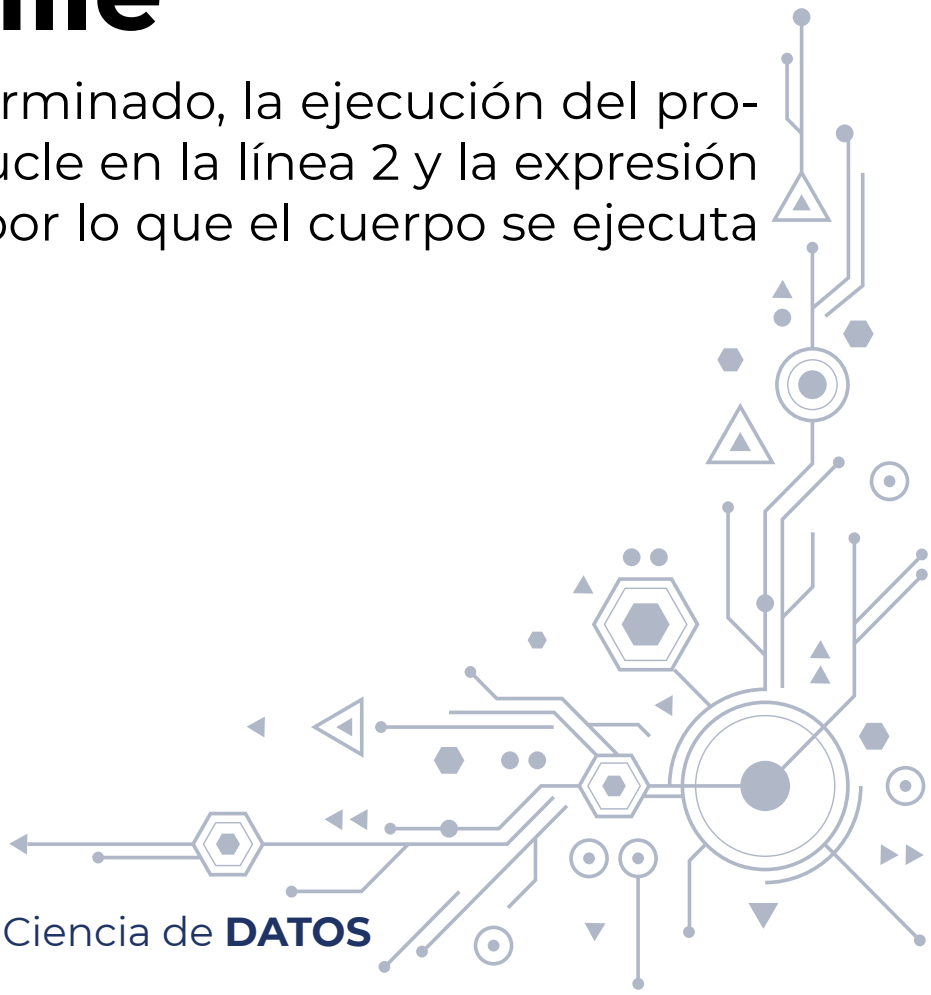
Ejemplo Ciclo While

```
1 n = 5
2 while n > 0:
3     n -= 1
4     print(n)
```

n es inicialmente 5. La expresión en el encabezado del ciclo **while** en la línea 2 es n > 0, lo cual es cierto, por lo que se ejecuta el cuerpo del ciclo. Dentro del cuerpo del bucle en la línea 3, n se reduce de 1 a 4 y luego se imprime.

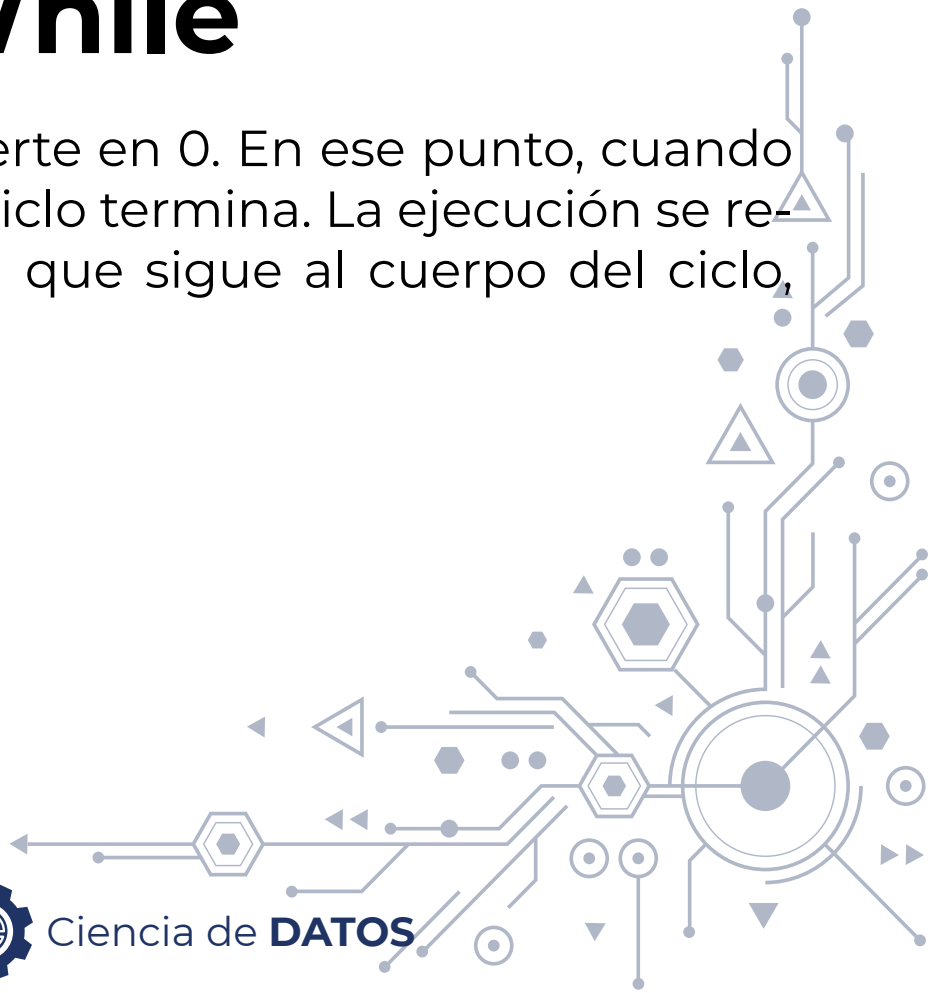
Ejemplo Ciclo While

Cuando el cuerpo del ciclo **while** ha terminado, la ejecución del programa vuelve a la parte superior del bucle en la línea 2 y la expresión se evalúa de nuevo. Todavía es cierto, por lo que el cuerpo se ejecuta de nuevo y se imprime 3.



Ejemplo Ciclo While

Esto continúa hasta que **n** se convierte en 0. En ese punto, cuando se prueba la expresión, es falsa y el ciclo termina. La ejecución se reanudaría en la primera declaración que sigue al cuerpo del ciclo, pero no hay ninguna en este caso.



Ejemplo Ciclo While

Debes tener en cuenta que la <expresión> de control del ciclo while se prueba primero, antes de ejecutar cualquier sentencia. Si la <expresión> es falsa desde el inicio, el cuerpo del ciclo nunca se ejecutará.

```
n = 0
while n > 0:
 n -= 1
print(n)
```

Las sentencias break y continue en Python

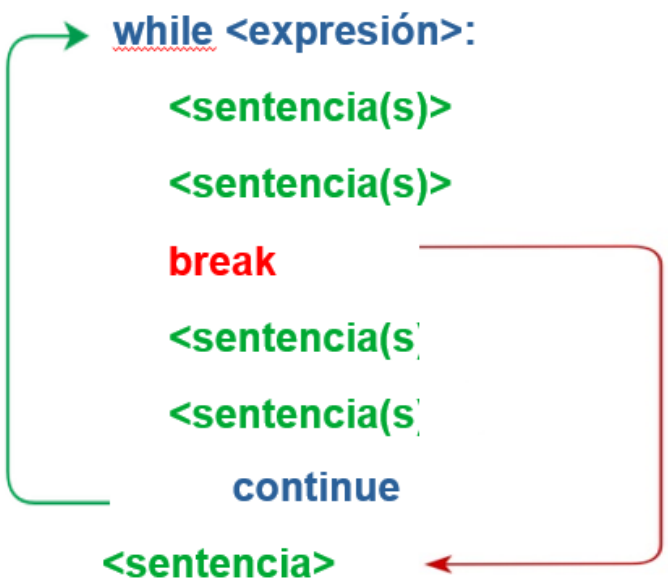
En los ejemplos que has visto hasta ahora, todo el cuerpo del ciclo **while** se ejecuta en cada iteración. Python proporciona dos palabras clave que terminan de forma anticipada una iteración de ciclo:

La declaración break de Python termina de forma inmediata el ciclo **while** por completo. La ejecución del programa saltará hasta la siguiente instrucción que sigue al cuerpo del ciclo.

La declaración de continue de Python finaliza inmediatamente la iteración del ciclo actual. La ejecución salta a la parte superior del ciclo y la expresión de control se vuelve a evaluar para determinar si el ciclo se ejecutará de nuevo o terminará.

Las sentencias break y continue

La diferencia entre break y continue se demuestra en el siguiente diagrama:



Ejemplo

```
1 n = 5
2 while n > 0:
3   n -= 1
4   if n == 2:
5       break
6   print(n)
7  print('Ciclo terminado.')
```

Cuando n se convierte en 2, se ejecuta la instrucción break. El ciclo termina por completo y la ejecución del programa salta a la instrucción print() en la línea 7.

```
4
3
Ciclo terminado.
```

La clausula else

Python permite una cláusula else opcional al final de un ciclo **while**. Esta es una característica única de **Python**, que no se encuentra en la mayoría de los otros lenguajes de programación. La sintaxis se muestra a continuación:

```
while <expresión>:
    <sentencia(s)>
else:
    <sentencia(s)_adicional(es)>
```

La <sentencia(s)_adicional(es)> especificada en la cláusula else se ejecutará cuando finalice el ciclo while.

La clausula else

Cuando se colocan <sentencias_adicionales> en una cláusula **else**, se ejecutarán solo si el ciclo termina "por agotamiento", es decir, si el ciclo itera hasta que la condición de control se vuelve falsa. Si se sale del ciclo con una instrucción break, la cláusula else no se ejecutará.

Ejemplo clausula else

```
n = 5
while n > 0:
    n -= 1
    print(n)
else:
    print('Ciclo terminado.')
```

Resultado:

```
4
3
2
1
0
Ciclo terminado.
```

Ejemplo clausula else

En este caso, el ciclo se repitió hasta que se agotó la condición: n se convirtió en 0, por lo que n > 0 se volvió falso. Debido a que el ciclo vivió su vida natural, por así decirlo, se ejecutó la cláusula else.

Ejemplo clausula else

```
n = 5
while n > 0:
    n -= 1
    print(n)
    if n == 2:
        break
else:
    print('Ciclo terminado.')
```

Resultado:

4
3
2

Este ciclo termina prematuramente con break al momento que la condición de n==2 es verdadera, por lo que la cláusula else no se ejecuta.

Ciclo for

Los ciclos de iteración definidos se denominan con frecuencia ciclos **for** porque **for** es la palabra clave que se utiliza para presentarlos en casi todos los lenguajes de programación, incluido **Python**.

Históricamente, los lenguajes de programación han ofrecido una variedad de tipos de ciclo **for**. Estos se describen brevemente en las siguientes secciones.

Ciclo de rango numérico

El ciclo **for** más básico es una declaración de rango numérico simple con valores iniciales y finales. El formato exacto varía según el lenguaje, pero normalmente la sintaxis es la siguiente:

```
for i = 1 to 10
    <cuerpo del ciclo>
```

Ciclo de rango numérico

En el ejemplo, el cuerpo del ciclo se ejecuta diez veces. La variable i toma el valor 1 en la primera iteración, 2 en la segunda, y así sucesivamente. Este tipo de bucle **for** se usa en los lenguajes BASIC, Algol y Pascal.

Cilco de tres expresiones

Otra forma de ciclo **for** muy popular traída por ell lenguaje de programación C contiene tres partes:

Una inicialización

Una expresión que especifica una condición final.

Una acción que se realizará al final de cada iteración.

Forma del ciclo for de 3 expresiones

Este tipo de ciclo tiene la siguiente forma:

```
for (i = 1; i <= 10; i++)
  <cuerpo del ciclo>
```

Este ciclo se interpreta de la siguiente manera:

Inicializar i a 1.
Continúa recorriendo mientras i <= 10.
Incrementa i en 1 después de cada iteración de ciclo.

El ciclo for de Python

El ciclo **for** de Python, facilita la iteración sobre los elementos de una lista. El ciclo **for** siempre se utilizará sobre una lista de elementos, de forma tal que en cada iteración, se puedan ejecutar las misma acciones sobre cada uno de los elementos de la lista.

Sintaxis ciclo for

```
for <variable> in <iterable>:
    <sentencia(s)>
```

<iterable> es una colección de objetos, por ejemplo, una lista de elementos. Las <sentencia(s)> en el cuerpo del ciclo se indican mediante indentación o sangría, como todas las estructuras de control de Python, y se ejecutan una vez para cada elemento en <iterable>. La variable de ciclo <variable> toma el valor del siguiente elemento en <iterable> cada vez que pasa por el ciclo.

Ejemplo ciclo for

```
mi_lista = ['Juan', 'Antonio', 'Pedro', 'Ana']
for elemento in mi_lista:
    print(elemento)
```

Resultado:

Juan
Antonio
Pedro
Ana

Explicación ejemplo

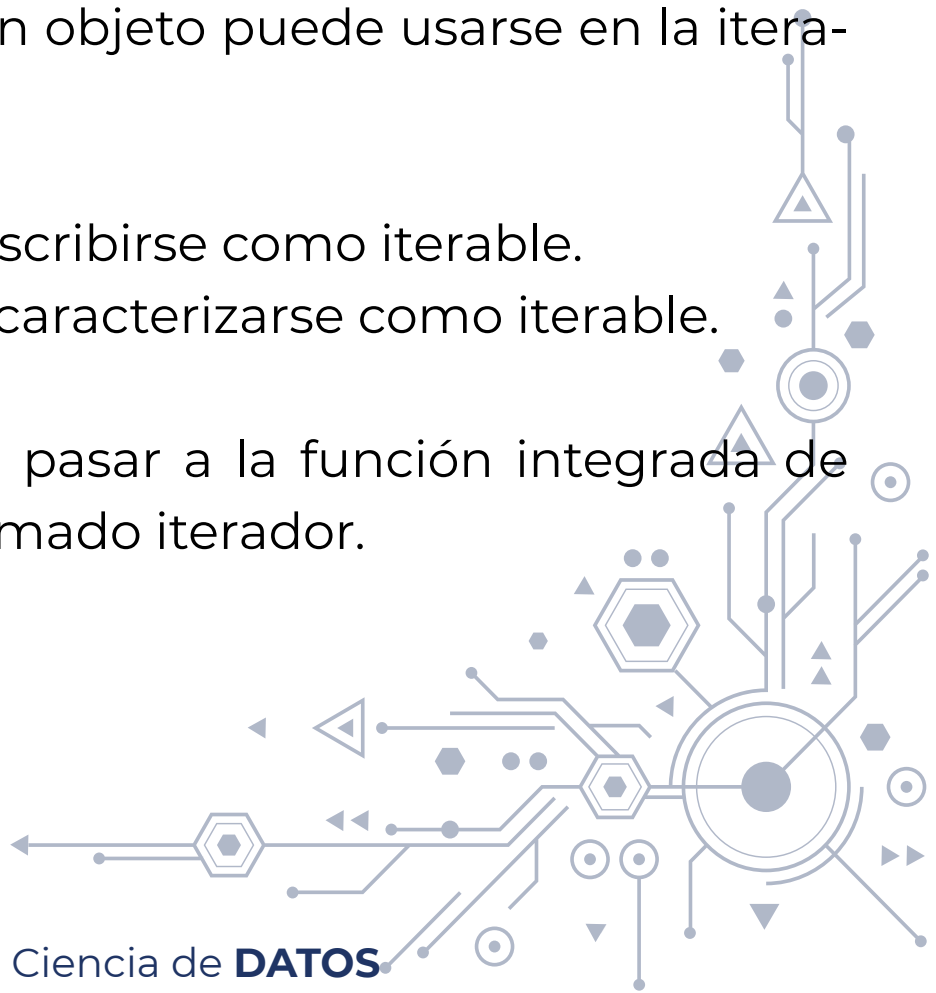
En este ejemplo, <iterable> es la lista mi_lista, y <var> es la variable elemento. Cada vez que pasa por el ciclo, elemento toma un elemento sucesivo en mi_lista, por lo que print() muestra los valores 'Juan', 'Antonio', 'Pedro', 'Ana', respectivamente. Un ciclo **for** como este es la forma Pythonic de procesar los elementos en un iterable.

Iterables

En Python, iterable significa que un objeto puede usarse en la iteración. El término se utiliza como:

Un adjetivo: un objeto puede describirse como iterable.
Un sustantivo: un objeto puede caracterizarse como iterable.

Si un objeto es iterable, se puede pasar a la función integrada de Python iter(), que devuelve algo llamado iterador.



Ejemplo iterable

Cada uno de los objetos del siguiente ejemplo es iterable y devuelve algún tipo de iterador cuando se pasa a iter():

```
iter('foobar')                    # String

iter(['foo', 'bar', 'baz'])       # Lista

iter(('foo', 'bar', 'baz'))       # Tupla

iter({'foo', 'bar', 'baz'})       # Set

iter({'foo': 1, 'bar': 2, 'baz': 3})    # Dict
```

Ejemplo iterable

Un iterador es esencialmente un productor de valor que produce valores sucesivos de su objeto iterable asociado. La función incorporada next() se usa para obtener el siguiente valor del iterador.

Ejemplo iterador

Aquí hay un ejemplo usando la misma lista que la anterior:

```
a = ['foo', 'bar', 'baz']
 itr = iter(a)
itr
<list_iterator object at 0x031EFD10>

next(itr)
'foo'
next(itr)
'bar'
next(itr)
'baz'
```

Explicación ejemplo

En este ejemplo, a es una lista iterable e itr es el iterador asociado, obtenido con iter(). Cada llamada next(itr) obtiene el siguiente valor de itr.

Observa cómo un iterador conserva su estado internamente. Sabe qué valores ya se han obtenido, por lo que cuando llama a next(), sabe qué valor devolver a continuación.