Bucle for en Python: cómo se utiliza, ventajas y ejemplos

El **ciclo for** es una herramienta muy útil en Python que permite iterar sobre objetos iterables y realizar operaciones repetitivas de manera sencilla y eficiente. Entre sus principales ventajas se encuentra su facilidad de uso y su capacidad para recorrer cualquier objeto iterable.

Además, el **ciclo for** en Python tiene una gran cantidad de usos, desde procesamiento de datos y manipulación de listas y diccionarios, hasta la realización de tareas repetitivas, y sobre eso hablaremos en este artículo.

# Bucle for

A continuación explicaremos el bucle for y sus particularidades en Python, que comparado con otros lenguajes de comparación, tiene ciertas diferencias.

El for es un tipo de bucle, parecido al [while](https://ellibrodepython.com/while-python) pero con ciertas diferencias. La principal es que el número de iteraciones de un for **esta definido** de antemano, mientras que en un while no. La diferencia principal con respecto al while es en la condición. Mientras que en el while la condición era evaluada en cada iteración para decidir si volver a ejecutar o no el código, en el for no existe tal condición, sino un iterable que define las veces que se ejecutará el código. En el siguiente ejemplo vemos un bucle for que se ejecuta 5 veces, y donde la i incrementa su valor “automáticamente” en 1 en cada iteración.

for i in range(0, 5):

print(i)

# Salida:

# 0

# 1

# 2

# 3

# 4

Si has leído el capítulo del while, tal vez ya empieces a ver ventajas en el uso del for. Si por ejemplo, queremos tener un número que va creciendo de 0 a n, hacerlo con for nos ahorra alguna línea de código, porque no tenemos que escribir código para incrementar el número.

En Python se puede iterar prácticamente todo, como por ejemplo una cadena. En el siguiente ejemplo vemos como la i va tomando los valores de cada letra. Mas adelante explicaremos que es esto de los **iterables** e **iteradores**.

for i in "Python":

print(i)

# Salida:

# P

# y

# t

# h

# o

# n

## Iterables e iteradores

## Sintaxis del Ciclo For en Python:

La sintaxis de un ciclo for es simple en Python. En realidad, en la mayoría de los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3 componentes del ciclo for (inicio, final y tamaño de paso) tenemos prácticamente todo hecho

for variable\_contadora in range(valor\_inicial, valor\_final, tamaño\_paso:

...

...

Bloque de Instrucciones...

...

...

Bloque fuera del ciclo...

Como puedes ver, la magia está en la función range() de Python, pues con esta definimos desde dónde y hasta dónde irá el ciclo, además del tamaño del paso. De hecho, la función range(), solo necesita como mínimo el límite superior, generando así un rango de números desde cero hasta ese límite de uno en uno. Finalmente, esta función omite el último valor, de ese modo, el ciclo va de valor\_inicial hasta valor\_final - 1 (hay que quitarle uno al valor\_final).

Por supuesto, como ya se te hará familiar, la instrucción se llama for, i termina con los dos puntos ":", y los que se ejecute en su interior lleva la indentación. Adicionalmente, variable\_contadora nos indicará en qué valor exacto del recorrido de ese rango de números nos encontramos.

No te preocupes si no comprendiste muy bien lo que acabo de escribir, estoy seguro que con un par de ejemplos que veremos a continuación te va a quedar todo claro.

## Ejemplos de Ciclo For en Python

A continuación, vamos a ver unos cuantos ejemplos para comprender de manera adecuada el uso de los ciclos for en Python. Recuerda que, si no comprendes alguno de estos ejemplos o tienes alguna pregunta o sugerencia sobre estos o cualquier contenido de la sección, puedes dejarlas en la [sección de comentarios](https://www.programarya.com/Cursos/Python/Ciclos/Ciclo-for#comments)

### Ejemplo 1: Mostrar en pantalla los números pares

Vamos a retomar el ejemplo mencionado anteriormente, donde deseábamos sacar los números pares entre el número 500 y el 1000 y mostrarlos. Es un ejemplo sencillo con el que nos aseguraremos de haber comprendido bien lo anterior:

### Solución inicial del ejemplo 1:

for i in range(500, 1000, 2):

print(i)

El código anterior funciona. Sin embargo, seguro notaste al ejecutarlo que solo mostró hasta el 998, incluso aunque 1000 es un número par. Esto es debido a que, como indiqué, range() va hasta el límite superior - 1, así que debes sumarle 1, para que sea incluido, así:

### Solución mejorada del Ejemplo 1:

for i in range(500, 1000 + 1, 2):

print(i)

Nota: Si no comprendes alguna de las líneas de estos códigos, te recomiendo visitar los contenidos anteriores ([Salida de Datos](https://www.programarya.com/Cursos/Python/salida-y-mostrar-datos) o también puedes [dejar un comentario](https://www.programarya.com/Cursos/Python/Ciclos/Ciclo-for#async) al final de la sección.

### Ejemplo 2: Cuenta regresiva en un ciclo for

Ahora veremos otro ejemplo sencillo. Con un ciclo for que haga sus iteraciones en sentido inverso, es decir, disminuyendo el valor del paso, desde el 100 hasta el 0, veamos:

### Solución Ejemplo 2:

Para este caso, debido a que queremos ir de un número mayor a uno más pequeño, por lo tanto, para este ejemplo el valor inicial será 100 y el valor final será 0. Adicional, el tamaño de paso será de 1 negativo, es decir, -1, así:

for i in range(100, -1, -1):

print(i)

Es importante notar que el ciclo va hasta -1, para que se incluya el 0 en el resultado.

### Ejemplo 3: Contador con un ciclo for

Para este ejemplo haremos algo un poco más complejo. El ejemplo consiste en contar al interior de un ciclo for, cuántos números entre el 0 y el 10.000 son múltiplos del 33. Para ello haremos uso del operador % (módulo) que obtiene el residuo de una división y también usaremos un pequeño [condicional](https://www.programarya.com/Cursos/Python/Condicionales) para verificar que el módulo sea cero al dividir por 33.

**Nota:**El operador de módulo (%) obtiene el residuo de una división, por tanto, cuando el residuo es cero implica que la división es exacta y el dividendo es un múltiplo del divisor. Por ejemplo 10 % 3 nos dará el residuo de dividir 10 entre 3, el cual es 1, si calculamos 99 % 33 nos dará cero, pues 99 es múltiplo de 33 (33 \* 3 = 99).

### Solución Ejemplo 3:

Para este caso el valor inicial será 0 y el valor final será 10000. Adicional, el tamaño de paso será de 1 (este es el caso más común). Al interior del ciclo, en cada iteración verificaremos si el número en el que estamos es divisible por 33 o no y en caso afirmativo aumentaremos el contador en una unidad así:

**Nota:**Este problema se puede solucionar de maneras mucho más eficientes y adecuadas, pero para fines de comprensión y facilidad lo haremos usando un ciclo for de uno en uno.

contador = 0 # Iniciamos el contador en cero

for i in range(10000):

if (i % 33 == 0): # Preguntamos si el residuo es 0 (es múltiplo de 33)

contador += 1 # Si es múltiplo aumentamos el contador en 1

# Si no es múltiplo no hacemos nada

#Mostramos el valor del contador

print(contador)

Este ciclo for nos permitirá saber que existen 304 múltiplos del número 33 en los números del 0 al 10000 (incluido el 10000 mismo).