



El futuro digital
es de todos

MinTIC

Universidad
Industrial de
Santander



Misión
TIC 2022

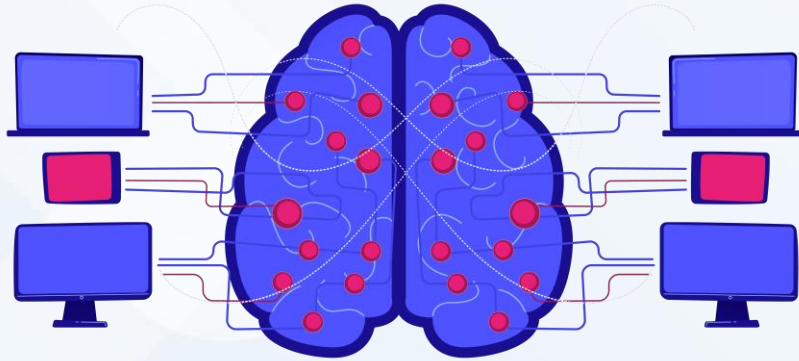
ESTRUCTURAS DE CONTROL ITERATIVAS



INTRODUCCIÓN

¡Hola! De seguro has estado practicando un montón con Python, nuestro lenguaje elegido para aprender a programar. En este punto tendrás muy claro los motivos y la manera de utilizar variables, condicionales y los distintos operadores aritméticos y lógicos. Incluso, ya podrías generar uno que otro algoritmo de inteligencia artificial haciendo uso de las estructuras de control condicionales. Por lo tanto, ya te encuentras preparado para aprender y entender las estructuras de control iterativas; estoy seguro de que lo disfrutarás.

INTELIGENCIA ARTIFICIAL



**Lo que las personas
creen que es**



**Lo que los programadores
creen que es**

```
1  
2 // 10,000 if-statements  
3  
4 if() {  
5   if() {  
6     if() {  
7       if() {  
8         if() {  
9           if() {  
10            if() {  
11              if() {  
12                if() {  
13                  if() {  
14                    if() {
```

Lo que realmente es

Comencemos por lo importante, ¿Por qué son necesarias las estructuras de control iterativas?

De seguro has visto en documentales y películas brazos robóticos o máquinas de llenado ejecutando una misma tarea una y otra vez, así como lo harías si tuvieras que ir todos los días una y otra vez al mercado. En la antigüedad, estas tareas eran realizadas por humanos, sin embargo, dadas las capacidades de las máquinas, las pueden hacer de manera rápida y **repetitiva**. Claro, déjame aclarar que la automatización es un área de la ciencia muy amplia, pero estás a punto de conocer un elemento básico para que la automatización en su nivel más simple sea posible.

En este punto notarás que hemos mencionado mucho la palabra “repetición”, “iteración” y ahora apareció un concepto nuevo que es la “automatización” puesto que es la base para entender las estructuras de control iterativas. Por lo tanto, vamos a dejar a un lado la palabra “repetición” y la vamos a reemplazar por “iteración” ¿De acuerdo?

Entonces, las estructuras de control iterativas son nuestra herramienta para ejecutar en los programas bloques de código de manera iterativa hasta que una o más condiciones se cumplan; ya vamos a ir entendiendo un poco más las condiciones y las distintas estructuras, pero por el momento ten claro el objetivo principal de las estructuras de control iterativas y su relación con la automatización.

2. Variables de control

Las variables son claves en las estructuras de control iterativas puesto que son el puente entre las iteraciones y la condición para que esta se ejecute. Normalmente y a lo largo de tu rol como programador, encontrarás tres tipos de variables de control principales: las banderas, los acumuladores y los contadores.

2.1. Banderas

Las banderas hacen referencia a variables que toman un valor preferiblemente binario, booleano e indican un estado; su valor y el cambio del mismo definen el estado en el que se encuentra el programa. Por ejemplo, requieres realizar una suma pero además deseas indicarle al usuario cuando la suma ya se ejecutó, podrías tener un código similar a:

```
● ● ●  
  
suma_realizada = False  
total = 0  
a = 5  
b = 10  
if (suma_realizada == False):  
    total = a + b  
    suma_realizada = True  
  
if(suma_realizada == True):  
    print("Se ha realizado una suma y su valor es " + str(total))
```

Como puedes observar, la variable *suma realizada* en este caso es de tipo bool y su función es indicar cuándo se ejecutó la suma, por lo tanto tiene un estado inicial False, pero luego de ejecutar la suma, toma valor *True*. Es común que escuches que la “bandera se levantó”, esto quiere decir que una u otra acción provocó que el estado de la bandera cambiase.

Cabe resaltar que las banderas no siempre son de tipo `bool`, puede ser de cualquier tipo siempre y cuando tengan un significado su estado, por ejemplo, podría ser una variable de tipo *string* como se muestra en el siguiente código de ejemplo:

```
contagio_validado = "No"
paciente = "Lisa"

if(contagio == "No"):
    print("La paciente " + paciente +
          " aún no se ha realizado su prueba para validar si se encuentra contagiada, se recomienda aplicar la prueba PCR")
    print("Aplicando prueba...")
    contagio_validado = "Pendiente"

if(contagio_validado == "Pendiente"):
    print(paciente + ", por favor valide en su correo el resultado de la prueba")
    contagio_validado = "Si"

if(contagio_validado == "Si"):
    print(paciente + ", de acuerdo a su resultado, por favor manténgase alejado de las personas")
```

En este caso, la bandera *contagio_validado* de tipo *string* toma diferentes valores, puesto que cambia de estado "No" a "Pendiente" y por último a "Si", por esto, aunque es común que las banderas sean de tipo *bool*, también se pueden encontrar algunas de tipo *string* en caso de que necesites validar más de dos estados.

2.2. Acumuladores

El objetivo de este tipo de variables es “almacenar” algún tipo de información en una sola variable a medida que se va ejecutando el programa. ¿Recuerdas el ejemplo de la abuelita pidiéndote realizar algunas compras? Ahora te voy a compartir el código con el que podríamos simular la creación de esa lista de ítems:



```
lista_compras = ""
print("La abuelita está escribiendo su lista de compras...")
lista_compras = lista_compras+"5 manzanas"
print("---Lista de compras---")
print(lista_compras)
lista_compras = lista_compras+", 3 lb Cilantro"
print("---Lista de compras---")
print(lista_compras)
lista_compras = lista_compras+", 3 lb Perejil"
print("---Lista de compras---")
print(lista_compras)
```


Al ejecutar el programa obtendrías como salida:



```
La abuelita está escribiendo su lista de compras...  
---Lista de compras---  
5 manzanas  
---Lista de compras---  
5 manzanas, 3 lb Cilantro  
---Lista de compras---  
5 manzanas, 3 lb Cilantro, 3 lb Perejil
```

Como puedes observar, la variable *lista compras* está **acumulando** la información de la lista de compras, es decir, no estamos creando una variable para cada ítem de compra, sino en una sola variable definimos y acumulamos el proceso.

Otro ejemplo de uso de acumuladores podría darse cuando ya estás en el mercado pero ahora es el señor que está haciendo la cuenta del total de tu compra. Por ejemplo, supongamos que las manzanas son a \$1200, la libra de cilantro a \$200 y la libra de perejil a \$300. Una opción sería en una sola línea de código realizar el cálculo de la cantidad de ítems por su precio unitario, sumarlos y determinar el total. Sin embargo, podemos tener otro acercamiento realizando el proceso paso a paso haciendo uso de acumuladores de la siguiente forma:

```
precio_manzana = 1200
cant_manzanas = 5
precio_cilantro = 200
cant_cilantro = 3
precio_perejil = 300
cant_perejil = 3
subtotal = 0
print("Calculando el total de tu mercado...")
total_manzana = precio_manzana * cant_manzanas
print("El valor total de las manzanas es: $" + str(total_manzana))
subtotal = subtotal + total_manzana
print("...El subtotal sería: $" + str(subtotal))
total_cilantro = precio_cilantro * cant_cilantro
print("El valor total del cilantro es: $" + str(total_cilantro))
subtotal = subtotal + total_cilantro
print("...El subtotal sería: $" + str(subtotal))
total_perejil = precio_perejil * cant_perejil
print("El valor total del perejil es: $" + str(total_perejil))
subtotal = subtotal + total_perejil
print("---El total del mercado es: $" + str(subtotal))
```

Con este bloque de código podemos tener mayor control puesto que con el uso del acumulador *subtotal* podemos ir dando más información a nuestro usuario de cómo va aumentando el valor total de su compra. Por lo tanto, lo que nuestro usuario vería como salida sería:



```
Calculando el total de tu mercado...  
El valor total de las manzanas es: $6000  
...El subtotal sería: $6000  
El valor total del cilantro es: $600  
...El subtotal sería: $6600  
El valor total del perejil es: $900  
---El total del mercado es: $7500
```

2.3. Contadores

En esta sección aprenderás a hacer uso de los contadores, de hecho, están al final de las variables de control dado que tienen mucha relación con los acumuladores de la sección anterior.

Los contadores son variables de control que como su nombre lo indica, controlan la **cantidad** de veces que se ejecuta determinada acción, estado, etc. Vamos a ver un ejemplo para tratar de entenderlo un poco mejor.

¿Recuerdas a nuestro amigo del mercado al que le pedimos una **cantidad** de algo?, por ejemplo, las manzanas, siempre le solicitamos 5 manzanas, si desarrolláramos un algoritmo para contar estas manzanas se vería algo así:

```
cont_manzanas = 0
print("Se ha iniciado el carrito. En total hay: " +
      str(cont_manzanas) + " manzanas.")
cont_manzanas = cont_manzanas + 1
print("Se ha agregado una manzana a la canasta. Ahora hay " +
      str(cont_manzanas) + " manzanas.")
cont_manzanas = cont_manzanas + 1
print("Se ha agregado una manzana a la canasta. Ahora hay " +
      str(cont_manzanas) + " manzanas.")
cont_manzanas = cont_manzanas + 1
print("Se ha agregado una manzana a la canasta. Ahora hay " +
      str(cont_manzanas) + " manzanas.")
cont_manzanas = cont_manzanas + 1
print("Se ha agregado una manzana a la canasta. Ahora hay " +
      str(cont_manzanas) + " manzanas.")
cont_manzanas = cont_manzanas + 1
print("Se ha agregado una manzana a la canasta. Ahora hay " +
      str(cont_manzanas) + " manzanas.")
```


Y si observamos la salida, obtendremos:

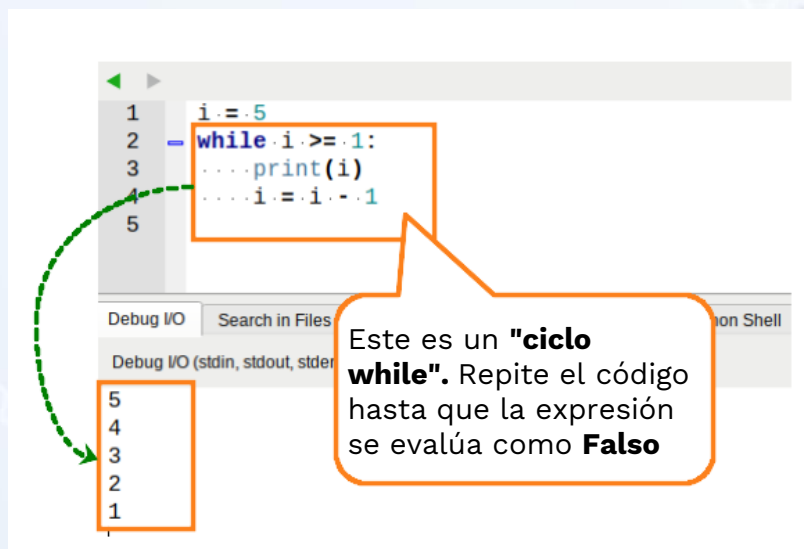
```
Se ha iniciado el carrito. En total hay: 0 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 1 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 2 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 3 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 4 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 5 manzanas.
```

Si te das cuenta, solo tenemos la variable *cont_manzanas* y sobre ella estamos **iterando**, en este caso, 5 veces. Su función es llevar la cuenta de las manzanas que se van agregando al carrito. Este tipo de variable de control será vital para el funcionamiento del ciclo for pero no te preocupes, cuando llegue el momento lo abordaremos con mayor profundidad.

3. Ciclos controlados por condiciones (WHILE)

Ya terminaste de entender las variables de control, son la base fundamental de los ciclos de control, pues ya notarás que son las encargadas de determinar el flujo de tu algoritmo. Ahora te encuentras preparado para empezar por el ciclo controlado por condiciones, específicamente el ciclo while.

Acá es muy importante su nombre pues su traducción literal del inglés al español es "mientras" y verás que su nombre nos indica muy específicamente de su funcionamiento. El ciclo while se compone de una condición y su bloque de código; este bloque de código se ejecutará **mientras** que la condición da como resultado Verdadero o True.



The image shows a Python IDE with a code editor and a console. The code editor contains the following Python code:

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

The console shows the output of the code, which is the numbers 5, 4, 3, 2, and 1, each on a new line. A green dashed arrow points from the code to the output. An orange callout box points to the code and contains the text: "Este es un 'ciclo while'. Repite el código hasta que la expresión se evalúa como Falso".

La condición que define la **iteración** del bloque de código por lo general tendrá relación con un acumulador, bandera o contador y acá es donde se empiezan a unir las cosas. Como habrás notado en la explicación de variables de control, muchas veces repetimos el código una y otra vez cambiando el valor de la bandera, contador o acumulador, siendo evidente que no es algo óptimo, y como siempre, las cosas que no son óptimas son reemplazadas por la automatización.

Por lo tanto, revisemos uno de los ejemplos implementados previamente pero ahora haciendo uso del ciclo while, específicamente vamos a implementar el contador de manzanas teniendo en cuenta que queremos tener en total 5 manzanas. El código implementado sería:

```
manzanas = 5
cont_manzanas = 0

print("Se ha iniciado el carrito. En total hay: " +
      str(cont_manzanas) + " manzanas.")

while(cont_manzanas < manzanas):
    cont_manzanas = cont_manzanas+1
    print("Se ha agregado una manzana a la canasta. Ahora hay " +
          str(cont_manzanas) + " manzanas.")
```

y la salida obtenida sería:

```
Se ha iniciado el carrito. En total hay: 0 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 1 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 2 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 3 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 4 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 5 manzanas.
```

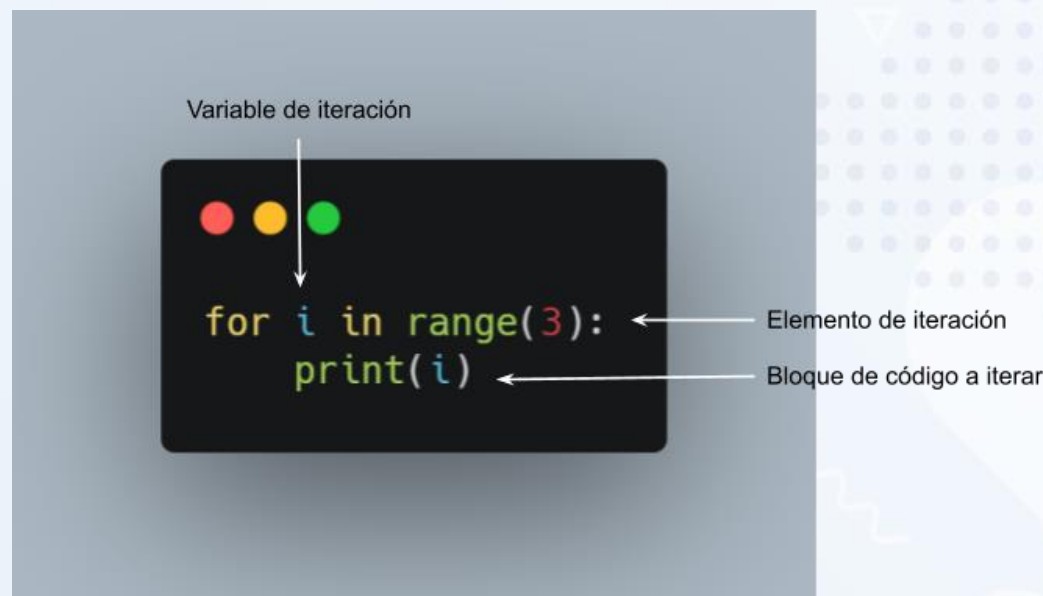
Como se puede observar en el código implementado, la cantidad de líneas disminuye y la complejidad en la lectura de la implementación es mucho más sencilla. Por otro lado, si ahora no queremos agregar 5 manzanas al carrito si no 2.000.000, no tendremos que copiar y pegar las mismas líneas de código 2.000.000 de veces si no que cambiaríamos el valor de la variable *manzanas* por 2.000.000.

Esta es la magia del ciclo *while*, por último es importante resaltar que dentro del ciclo se va afectando una o más de las variables implicadas en la declaración de la condición que debe cumplir el ciclo, en nuestro ejemplo, es la variable *cont_manzanas* para que en **algún momento** la condición sea verdadera y se salga del ciclo, de lo contrario, tendríamos un ciclo que nunca se detiene pues su condición de verdad sería siempre *False* y provocaremos una ruptura en el espacio-tiempo con un ciclo interminable, o en su defecto, cerrar el programa.

4. Ciclos controlados por cantidades (FOR)

De seguro ahora que entendiste los ciclos *while* te habrás imaginado una serie de situaciones en las que este ciclo tiene una relación directa con un contador, y no necesariamente con una condición de verdad, es decir, estarás imaginando diferentes situaciones, como la del contador de manzanas, en el que nuestra condición está directamente relacionada con el conteo de elementos y por lo tanto, el uso de contadores.

Pues gracias a que los diseñadores de los lenguajes de programación notaron esta situación bastante común, decidieron crear una estructura especializada y optimizada para los ciclos controlados por cantidad.



Para nuestro lenguaje de programación Python, el ciclo *For* es una herramienta muy poderosa puesto que el elemento de iteración nos permite ingresar rangos, cadenas de caracteres, estructuras de datos complejas o cualquier otro elemento iterable, este elemento iterable es necesario que tenga una **cantidad** definida y esta cantidad es la que marca la diferencia con el ciclo *while*, puesto que mientras que el ciclo *while* parte de una **condición de verdad**, el ciclo *for* parte de una **cantidad** definida.

Vamos a darnos la oportunidad de recurrir de nuevo a nuestro contador de manzanas para que puedas tener más clara la diferencia entre los dos tipos de ciclos. A continuación encontrarás el código:

```
print("Se ha iniciado el carrito. En total hay: 0 manzanas.")  
  
for i in range(1, 6):  
    print("Se ha agregado una manzana a la canasta. Ahora hay " +  
          str(i) + " manzanas.")
```

Y obtendremos en la salida:

```
Se ha iniciado el carrito. En total hay: 0 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 1 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 2 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 3 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 4 manzanas.  
Se ha agregado una manzana a la canasta. Ahora hay 5 manzanas.
```

Un momento, nos acabamos de encontrar una instrucción un tanto extraña de la que nunca hemos hablado y es la instrucción *range()*, sin embargo, no te preocupes por ello, voy a darte una pequeña introducción pero más adelante lo entenderás a profundidad.

La instrucción *range()* como su nombre lo indica define un rango, puedes definirlo, por ejemplo, como *range(5)* y lo que significa es que se va a iterar desde **0** hasta **5** sin incluir el 5, es decir, aplicado en el *for*, la variable *i* tomará los valores de 0, 1, 2, 3, 4; en este caso, definimos *range(1,6)* lo que implica que el rango **iniciará** en 1 y terminará en **6**, sin incluirlo, es decir, 1, 2, 3, 4, 5.

Ahora, en la declaración del ciclo *for*, como se ha dicho previamente, luego de la palabra clave, se define la variable que se iterará, en este caso, es la variable *i*, el nombre "i" como variable iteradora es un estándar bastante común en los lenguajes de programación. Esta variable *i* tomará el valor correspondiente a como vaya ejecutando la iteración el ciclo.

Material de estudio complementario

[Python conditional statements and loops - Exercises, Practice, Solution - w3resource](#)

[Achieve mastery through challenge | Codewars](#)

[Python for loop and if else Exercise \[10 Exercise Questions\] \(pynative.com\)](#)

Referencias Bibliográficas

W3schools, (2021). "Python While Loops".

Recuperado: https://www.w3schools.com/python/python_while_loops.asp

Uniwebsidad, (2021). "Estructuras de Control de Flujo" Recuperado:

<https://uniwebsidad.com/libros/python/capitulo-2/estructuras-de-control-de-flujo>

Python Software Foundation (2021). Python.org, "Sentencias compuestas" Recuperado:

https://docs.python.org/es/3/reference/compound_stmts.html

Learnpython.org, "Loops" (s.f.) Recuperado: <https://www.learnpython.org/es/Loops>