



Hechos

QUE

CONECTAN ✓



El futuro digital
es de todos

MinTIC

VARIABLES DE CONTROL Y CICLOS ITERATIVAS

Universidad
Industrial de
Santander



**Misión
TIC 2022**

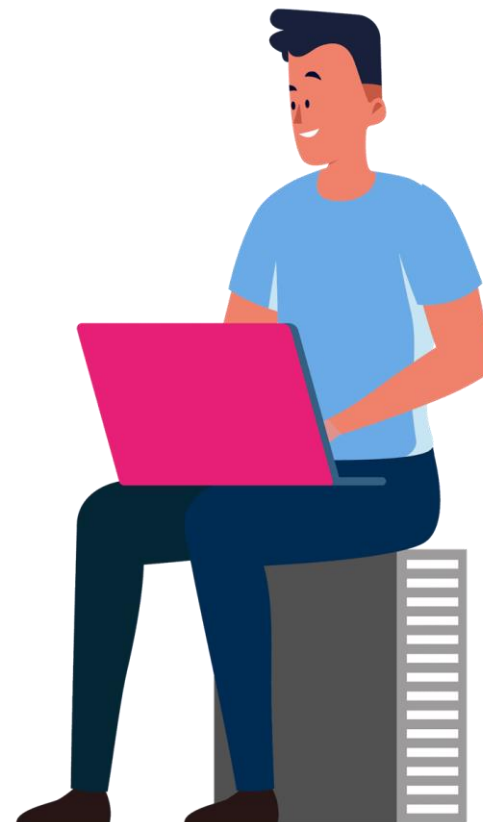
3.1. INTRODUCCIÓN

3.2. VARIABLES DE CONTROL

3.2.1. Banderas

Un ejemplo de banderas es:

Se requiere 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`, pueden ser de cualquier tipo, siempre y cuando tenga 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")
```

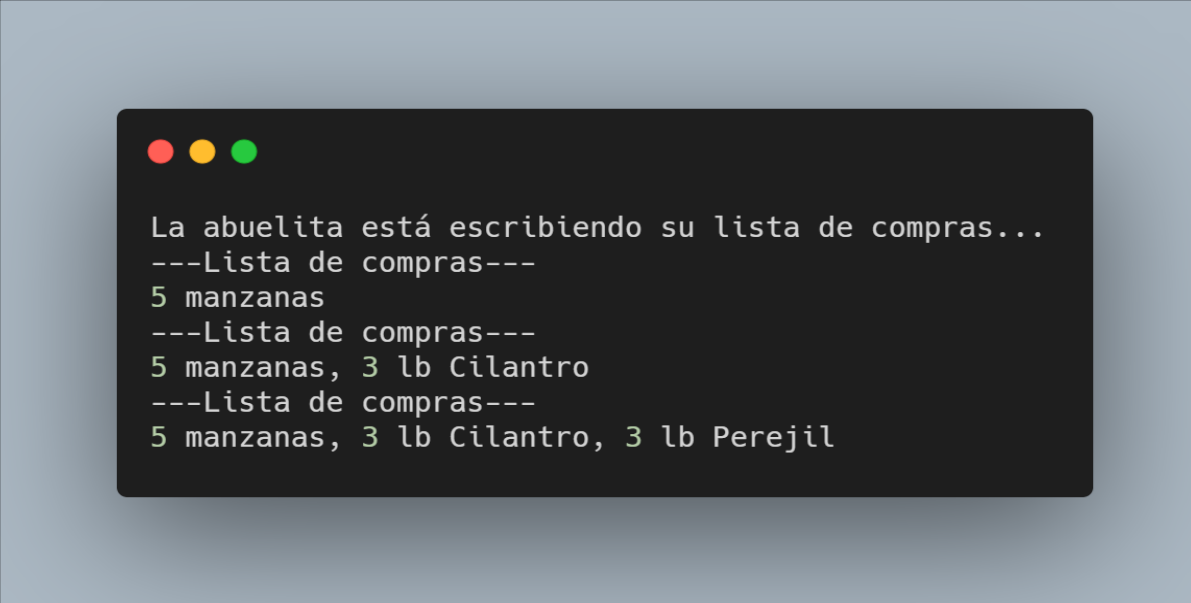
En este caso, la bandera `contagio_validado` de tipo `string` toma diferentes valores, puesto que cambia de estado "No" a "Pendiente", y por último, a "Si". Por esto, aunque es común que las banderas sean de tipo `bool`, también se pueden encontrar algunas de tipo `string`, en caso de que necesites validar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

3.2.2 Acumuladores

Pensemos en una tarea. Una abuelita desea comprar unos elementos en el supermercado: 5 manzanas, 3 libras de cilantro y 3 libras de perejil. Un amigo cercano decide hacer el favor de hacerle la compra. Con base a esto, se desea crear la lista de compras a partir del uso de las cadenas de caracteres. El código en Python que permite simular la creación de esta lista de ítems es:

```
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 que 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:



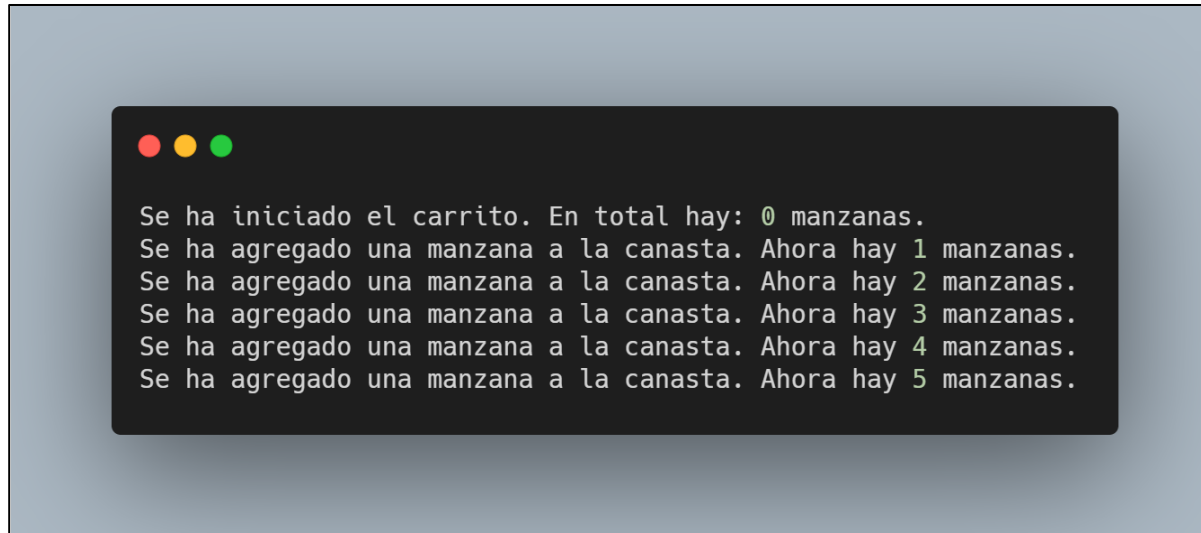
```
Calculando el total de tu mercado...  
El valor total de las manzanas es: $6000  
...El subtotal sería: $6000  
El valor total del cilantro es: $600  
...El subtotal sería: $6600  
El valor total del perejil es: $900  
---El total del mercado es: $7500
```


3.2.3 Contadores

¿Recuerdas a nuestro amigo del mercado al que le pedimos una cantidad de algo?, por ejemplo, las manzanas, siempre le solicitamos 5 manzanas, si desarrollaremos 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.")
```

Y si observamos la salida, obtendremos:

A terminal window with a dark background and light gray text. It has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside shows a sequence of messages: 'Se ha iniciado el carrito. En total hay: 0 manzanas.' followed by five lines of 'Se ha agregado una manzana a la canasta. Ahora hay X manzanas.' where X increases from 1 to 5.

```
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.

3.3. CICLOS ITERATIVOS

3.3.1. Ciclos controlados por condiciones

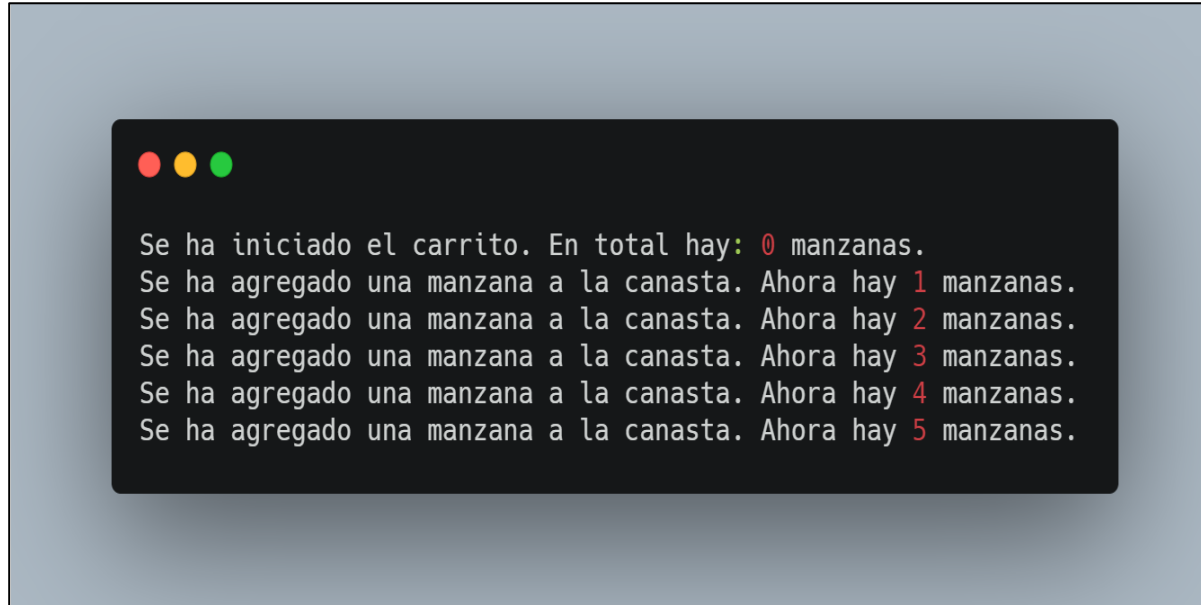
En el siguiente ejemplo, vamos a implementar el contador de manzanas teniendo en cuenta que queremos tener en total 5 manzanas. El código implementado sería:

```
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:

A screenshot of a terminal window with a dark background and light gray text. The window has three colored window control buttons (red, yellow, green) in the top-left corner. The text inside the terminal shows the execution of a program that adds apples to a basket. The output is as follows:

```
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 sino 2.000.000, no tendremos que copiar y pegar las mismas líneas de código 2.000.000 de veces, sino 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.



3.3.2 Ciclos controlados por cantidad

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>
- BUSTAMANTE, S. (2021) FreeCodeCamp, "Indentación en Python con ejemplos". Recuperado: <https://www.freecodecamp.org/espanol/news/indentacion-en-python-con-ejemplos/>



El futuro digital
es de todos

MinTIC

Hechos

QUE

CONECTAN



CICLO 1

EJE TEMÁTICO 3

**VARIABLES DE CONTROL
Y CICLOS ITERATIVOS**

Universidad
Industrial de
Santander



Mision
TIC 2022