

El Ciclo While

El Ciclo While	1
¿Qué aprenderás?	2
Introducción	2
¡Comencemos!	2
Introducción a Ciclos	3
Ciclo While	3
while paso a paso	4
Salida del ciclo	4
Ejercicio Guiado 1: Password	5
Iterar	8
Contando con while	9
Operadores de asignación	9
Ejercicio guiado 2: La bomba de tiempo	10
Contadores y Acumuladores	12



¡Comencemos!

¿Qué aprenderás?

- Reconocer las sentencias de ciclos para la construcción de programas.

Introducción

A medida que la complejidad de nuestro código aumenta, comenzaremos a notar que más y más líneas de código se necesitan; esto es un proceso normal, pero implica que cada vez será más difícil entender todo el código a cabalidad y, en el caso de algún error, se irá haciendo cada vez más difícil poder hallarlo de manera sencilla. Es por esto que, como programadores, tenemos la responsabilidad de hacer nuestro código lo más eficiente posible y en caso de hallar duplicaciones, evitarlas a toda costa.

Para ello, en este capítulo se introducirán los ciclos, una de las primeras herramientas que podemos utilizar para evitar repetir código y disminuir las líneas necesarias. Además, los ciclos nos darán velocidad tanto en el desarrollo como en la ejecución de nuestro programa y nos permitirán generar funcionalidades que de otra forma no sería posible.



¡Comencemos!

Introducción a Ciclos

Los ciclos son sentencias que nos permiten repetir la ejecución de una o más instrucciones.

```
Mientras se cumple una condición:
```

```
Instrucción 1  
Instrucción 2  
Instrucción 3
```

Repetir instrucciones es la clave para crear programas avanzados, ya que, como programadores, nos interesa poder tener la mayor cantidad de funcionalidades sin tener que tener un código extremadamente largo.

La diferencia principal entre programadores principiantes y los más experimentados es que estos últimos, están constantemente buscando cómo optimizar su código. Optimizar no solo significa que utilice funciones más avanzadas, sino que también hacerlo más compacto, lo que ayudará a entenderlo de mejor manera.

Ciclo While

La instrucción `while` nos permite ejecutar una o más operaciones **mientras** se cumpla una condición, la cual es idéntica a las utilizadas en nuestras sentencias `if` y la sintaxis es la siguiente:

```
while condición:  
    # código a implementar
```

Una manera más gráfica de poder entender esto, puede ser utilizando los diagramas de flujo que vimos anteriormente:

while paso a paso

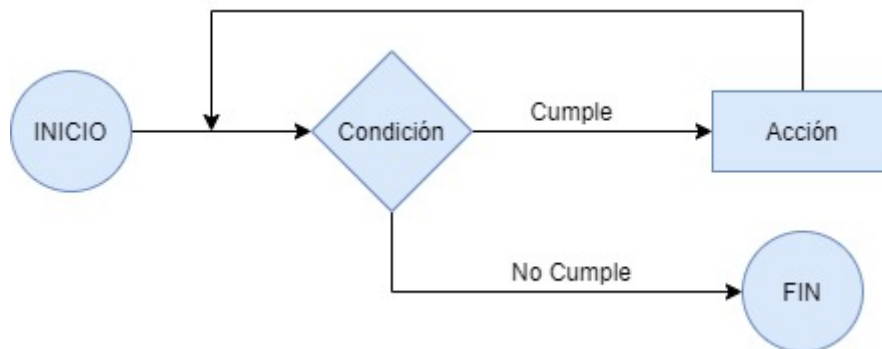


Imagen 1. Ciclo **while**.
Fuente: Desafío Latam

1. Se evalúa la condición; si es True, ingresa al ciclo.
2. Se ejecutan secuencialmente las instrucciones definidas dentro del ciclo.
3. Una vez ejecutadas todas las instrucciones, se vuelve a evaluar la condición:
 - Si se evalúa como **True**: vuelve a repetir.
 - Si se evalúa como **False**: sale del ciclo.

Salida del ciclo

Como vimos al inicio de la unidad, un algoritmo es una secuencia de pasos **FINITA** para resolver un problema. En algún momento, algunas de las instrucciones dentro del bloque deben lograr que la condición no se cumpla, es decir, debe existir una **condición de salida o término** del ciclo.

Ejercicio Guiado 1: Password

Todo buen programador sabe que existen algunos elementos que son sensibles, ya que no todo puede ser público, por lo que algunas veces va a ser necesario la inclusión de contraseñas o passwords para proteger la información. Entonces ¿Cómo podemos utilizar while para implementar un password?

1. Creemos el archivo `password.py`
2. Solicitamos la clave. Para ello, utilizaremos la librería `getpass`.

```
import getpass
password = getpass.getpass("Ingrese la clave secreta: ")
```

3. Ahora viene la parte interesante, ya que debemos identificar cómo queremos que nuestro programa funcione. En este caso, si la clave es correcta queremos que nuestro programa inicie, de lo contrario, queremos que vuelva a solicitar la clave. Dado que queremos que vuelva a realizar una acción de solicitar la clave hasta que se ingrese la clave correcta, es necesario utilizar el ciclo `while`:

```
# En este caso definimos nuestro password como "hola mundo"
# En este caso, mientras la contraseña no sea hola mundo,
# seguirá solicitando la contraseña, pero esta vez con otro mensaje.

while password != "hola mundo":
    password = getpass.getpass("La clave secreta no es correcta. Intenta
otra vez.")
```

4. Finalmente, podemos incluir el código final de nuestro programa. En este caso, solo agregaremos un código genérico que da inicio a nuestro programa.

```
print("Clave Correcta. Puedes utilizar tu programa")
# Posterior a esto podríamos agregar el código de nuestro programa.
```

Al realizar los pasos anteriores, el script completo se verá así:

```
1 import getpass
2 password = getpass.getpass("Ingrese la clave secreta: ")
3
4 # En este caso definimos nuestro password como "hola mundo"
5 # En este caso, mientras la contraseña no sea hola mundo,
6 # seguirá solicitando la contraseña, pero esta vez con otro mensaje.
7
8 while password != "hola mundo":
9     password = getpass.getpass("La clave secreta no es correcta. Intenta otra vez: ")
10
11 print("Clave Correcta. Puedes utilizar tu programa")
12 # Posterior a esto podríamos agregar el código de nuestro programa.
13
```

Imagen 2: Vista al Script completo
Fuente: Desafío Latam

¿Por qué es necesario declarar dos veces el ingreso de datos por parte del usuario?

Hay varias razones por las cuales es necesario solicitar el ingreso de la clave dos veces:

- La línea 2 solicita la clave por primera vez. Si la clave es correcta, el ciclo `while` nunca comenzará, y el programa iniciará su funcionamiento de la manera esperada.
- Adicionalmente, si la línea 2 no existe, aparecerá un error, ya que la línea 8 realiza una prueba lógica con la variable `password`, que no estaría definida.
- Finalmente, queremos solicitar la clave de dos maneras distintas, una de manera inicial (línea 1) y una en caso de equivocarse línea 9.

Ahora bien, revisemos el funcionamiento de nuestro programa:

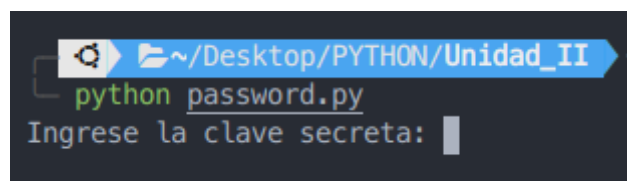
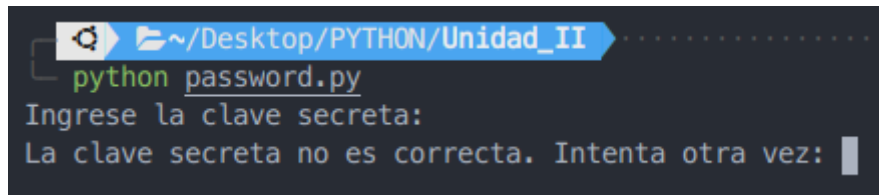


Imagen 3: Ejecutando el programa en el terminal
Fuente: Desafío Latam

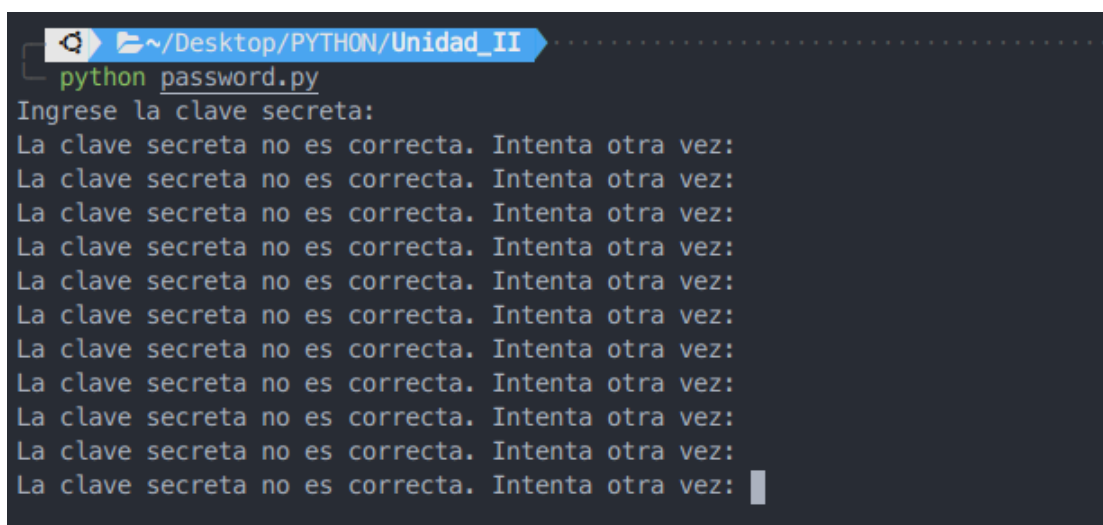
Al ejecutar el programa podemos ver que se solicita la clave secreta, y en caso de ingresar una clave incorrecta, ocurre lo siguiente:



```
~/Desktop/PYTHON/Unidad_II
python password.py
Ingrese la clave secreta:
La clave secreta no es correcta. Intenta otra vez: 
```

Imagen 4: Verificando con claves incorrectas
Fuente: Desafío Latam

Notamos que aparece el mensaje de error e inmediatamente nos solicita la clave nuevamente. La ventaja de usar una instrucción `while`, es que esto puede ocurrir cuantas veces sea necesario:



```
~/Desktop/PYTHON/Unidad_II
python password.py
Ingrese la clave secreta:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez: 
```

Imagen 5: Verificando en caso de muchas claves incorrectas
Fuente: Desafío Latam

Finalmente, si se ingresa la clave correcta, el programa funciona correctamente:

```
~/Desktop/PYTHON/Unidad_II
python password.py
Ingresa la clave secreta:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
Clave Correcta. Puedes utilizar tu programa
```

Imagen 6: Verificando con clave correcta
Fuente: Desafío Latam



NOTA: Este programa fue ejecutado en Linux, y las claves ocultas no aparecen. En otros sistemas, las claves ocultas pueden aparecer como *****.

Iterar

Iterar es dar una vuelta al ciclo, y por diseño, el ciclo **while** es un ciclo infinito, donde la mayoría de las veces no se sabe cuántas iteraciones tendrá. Por ejemplo, en el ejercicio de la contraseña, va a depender de cuantas veces el usuario ingrese el password de manera incorrecta; aún así, existen otros problemas en el que a priori **sí se conoce** cuántas veces es necesario iterar.

Por ejemplo el siguiente:

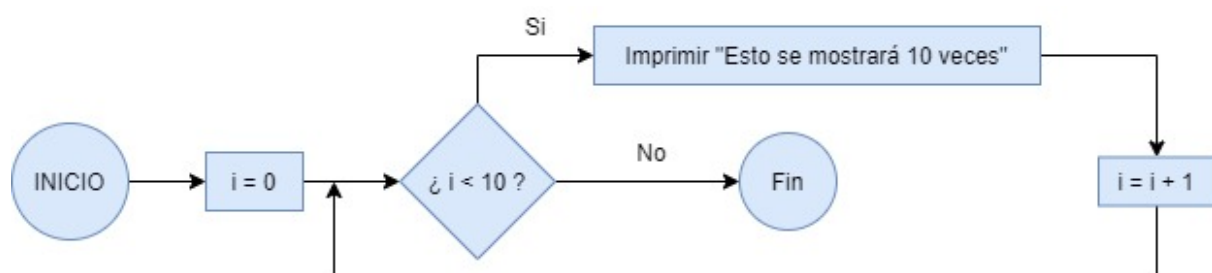


Imagen 7. Diagrama para mostrar un texto 10 veces.
Fuente: Desafío Latam

Contando con while

En el diagrama anterior, donde se buscaba imprimir un texto en pantalla 10 veces, la implementación del código se realiza de la siguiente forma:

```
i = 0
while i < 10:
    print("Esto se mostrará 10 veces") # está es la expresión a repetir
    i = i + 1 # IMPORTANTE
```

La instrucción `print("Esto se mostrará 10 veces")` se repetirá hasta que la variable `i` alcance el valor `10`. Una vez esto ocurra, la condición asociada al `while` se evaluará como `False` y terminará el ciclo.

Considera que en programación, es una convención utilizar una variable llamada `i` como variable de iteración para operar en un ciclo.

En este ciclo el iterador (`i`) en la primera vuelta valdrá 1, en la segunda iteración valdrá 2, en la tercera valdrá 3 y así sucesivamente hasta llegar a la condición declarada que sea menor que 10.



NOTA: Si no se aumenta el valor de la variable `i`, entonces el ciclo nunca alcanzará su condición de salida, por ende, el loop será infinito.

Operadores de asignación

Una de las ventajas que ofrece Python es su sintaxis, la cual es concisa y fácil de leer. Esto, porque nos permite reescribir algunos elementos que pueden ser difíciles de entender a primera vista, como por ejemplo:

```
# Este trozo de código incrementa el valor de i en 1.
# Al valor actual de i le suma 1 y lo vuelve a asignar a i.
i = i + 1

# Mismo resultado, pero más compacto
i += 1 # Esto es un operador de asignación
```

Los operadores de asignación permiten realizar una operación sobre una variable, pero a la vez sobrescribir esa misma variable, básicamente, son modificadores de esa variable.

En la siguiente tabla se muestra el comportamiento de algunos de los operadores de asignación:

Operador	Nombre	Ejemplo	Resultado
=	Asignación.	a = 2	a toma el valor 2.
+=	Incremento y asignación.	a += 2	a es incrementado en dos y asignado el valor resultante.
-=	Decremento y asignación.	a -= 2	a es reducido en dos y asignado el valor resultante.
*=	Multiplicación y asignación.	a *= 3	a es multiplicado por tres y asignado el valor resultante.
/=	División y asignación.	a /= 3	a es dividido por tres y asignado el valor resultante.

Tabla 1. Operadores de asignación.

Fuente: Desafío Latam

Ejercicio guiado 2: La bomba de tiempo

Para entender mejor el funcionamiento de los ciclos `while`, realizaremos un pequeño juego, el cual nos permitirá **combinar ciclos y operadores de asignación**. Para ello, crearemos un programa en el que ingresamos un tiempo determinado hasta que explote una bomba.

1. Crear el archivo `bomba.py`.
2. En este caso podemos utilizar `sys.argv` para ingresar el número de segundos antes de explotar.

```
import sys
i = int(sys.argv[1]) # fijar valor inicial
```

3. Creamos un ciclo que nos permita generar decremento, desde el valor inicial hasta cero:

```
import sys
i = int(sys.argv[1]) # fijar valor inicial
# el ciclo terminará cuando i sea 0
while i > 0:
    print(i)
    i -= 1 # i irá decreciendo en 1 unidad.
```

4. Este programa se ejecuta muy rápido y no representa un decremento de un segundo. Para que efectivamente `i` represente un segundo, podemos utilizar la librería `time`, la que nos permitirá esperar un segundo entre cada decremento. La idea es que al terminar el ciclo, la bomba explote, de manera que el código completo se vea así:

```
import time
import sys
i = int(sys.argv[1]) # fijar valor inicial

while i > 0:
    print(i) # Muestra el valor de i
    time.sleep(1) # espera un segundo
    i -= 1 # Descuenta 1 al valor de i.

print("BOOM!") # al salir del ciclo la bomba explota!!
```

Si bien este ejercicio puede parecer simple, no es tan intuitivo entender el código. A continuación, se muestra cómo sería en caso de no utilizar un ciclo, a través de un ejemplo que demuestra el caso de 5 segundos:

```
import time

i = 5
print(i) # Muestra el valor 5
time.sleep(1) # espera un segundo

i = 4 # Descuenta 1 al valor de i.
print(i) # Muestra el valor 4
time.sleep(1) # espera un segundo

i = 3 # Descuenta 1 al valor de i.
print(i) # Muestra el valor 3
time.sleep(1) # espera un segundo

i = 2 # Descuenta 1 al valor de i.
```

```
print(i)    # Muestra el valor 2
time.sleep(1) # espera un segundo

i = 1 # Descuenta 1 al valor de i.
print(i)    # Muestra el valor de i
time.sleep(1) # espera un segundo

i = 0
# En este punto se evalúa el fin de ciclo ya que i es cero.
print("BOOM!") # al salir del ciclo la bomba explota!!
```



NOTA: El código fue reordenado para poder entender mejor qué ocurre. Como se puede observar, hay muchas líneas que se pueden ahorrar gracias al uso de `while`.

Se puede observar que incluso un ejercicio sencillo genera un código muy largo. En caso de requerir distintos tiempos antes de la explosión se necesitan distintos códigos, lo cual no lo hace un proyecto factible.

Podemos analizar entonces, si estamos repitiendo 2 o más veces una misma línea de código, entonces es muy probable que pueda utilizar un ciclo.

En la plataforma tendrás el código de este ejercicio con el nombre **Ejecución guiada - bomba de tiempo**.

Contadores y Acumuladores

Cuando se trabaja con ciclos, existen algunos elementos que nos permiten hacer cálculos para llevar a cabo una tarea. Estos son los contadores y acumuladores:

- **Contador:** Aumenta de 1 en 1.
 - `cont = cont + 1.`
 - `cont += 1.`
- **Acumulador:** Acumula, el valor anterior más un valor adicional.
 - `acu = acu + valor.`
 - `acu += valor.`

Es bueno aclarar que también es posible utilizar el operador de asignación `+=` para ir acumulando strings. Esto porque, como ya hemos visto, el operador `+` funciona como un concatenador al ser aplicado este tipo de datos.

```
saludo = "hola"  
saludo += " mundo"  
print(saludo) # hola mundo
```

```
saludo += "chao"  
print(saludo) # hola mundo chao
```

Debido a que el término de un ciclo `while` depende de una condición, es necesario tener especial cuidado ya que se podría caer en ciclos infinitos. Un ciclo infinito, es aquel que no termina nunca, ya que no se puede alcanzar la condición de salida.

Para comprenderlo mejor, presta atención al siguiente ejemplo:

```
i = 1  
while i < 10:  
    print(i)
```

Este es un caso muy típico de ciclo infinito, en donde no se agrega un contador a `i` por lo que este ciclo jamás terminará. `i` siempre tendrá el valor 1, por lo que siempre será menor a 10, por lo tanto, este ciclo nunca alcanzará su valor de salida y, por ende, nunca terminará.

¡Cuidado! con los ciclos infinitos dado que pueden colapsar tu computador, si estás en la terminal de comandos o consola puedes detener dicho ciclo presionando la combinación de teclas `ctrl + c`.

Para practicar un poco más lo aprendido en este capítulo, te dejamos un ejercicio para que puedas seguir ejercitando.