




# Control de flujo y ciclos


Sesión Conceptual 2



# Recordemos...



¿Qué es un algoritmo?



¿Qué herramientas previas al código existen para enfrentarse a la creación de un algoritmo?



¿Qué son las pruebas lógicas?



# Desarrollo

{desafío}  
latam\_



70 minutos

**\*/ Ciclo While \*/**

# 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

## Ciclo while

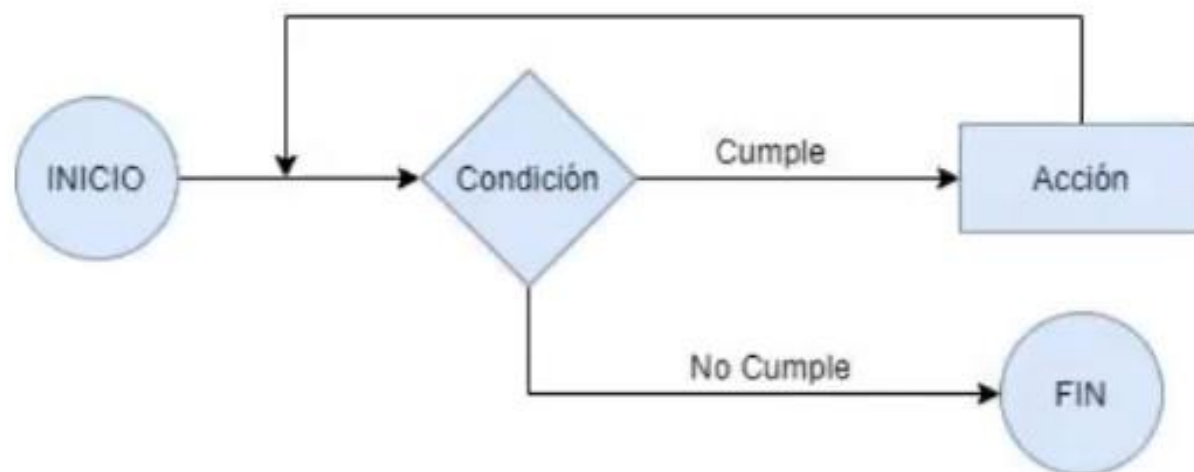
La instrucción `while` nos permite ejecutar una o más operaciones **mientras** se cumpla una condición, la cual es idéntica a las utilizadas en nuestras sentencias `if`.

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



## while paso a paso

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.





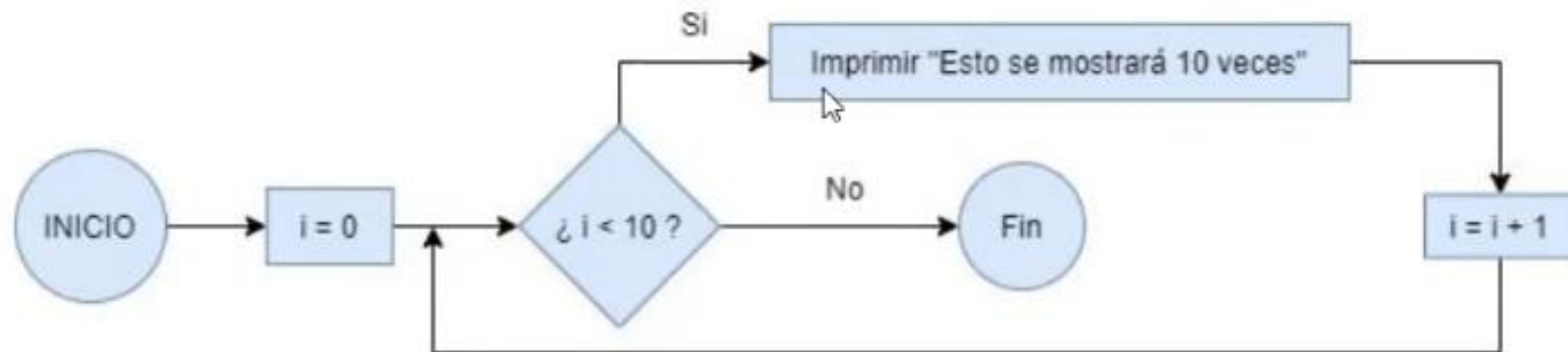


# Ejercicio Guiado

"Password"



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.



## 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
```

# Operadores 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.

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.



## Ejercicio guiado

"La bomba de tiempo"



# 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.`



# Quiz



{desafío}  
latam\_

29:06 - Sharing Started



**/\* Ciclo For \*/**

# El Ciclo For

La instrucción `for` en Python suele funcionar un poco distinto, ya que normalmente va a iterar en un rango de valores, pero en Python funciona más como un *"for each"*, es decir, que el `for` iterará en cada elemento de un objeto.

```
for variable in iterable:  
    # se ejecutará código para cada valor  
    del iterable.  
    # El código debe estar correctamente  
    indentado.
```

# Iterables

Los iterables son estructuras que tienen elementos en los cuales es posible desplazarse desde un elemento a otro en orden, y que corresponden a estructuras de datos, como por ejemplo, listas y diccionarios, etc.



## La función Range

Cuando se utiliza un solo valor, este corresponde al límite superior.

En el caso de utilizar dos valores, estos corresponderán al valor de inicio y al límite superior.

Al utilizar tres valores, estos corresponderán al valor inicial, al límite superior y al paso (step).

# Utilizando estructuras de datos en un ciclo For

Estructuras de datos:

Listas.

Strings.

Diccionarios.

# Otras funciones útiles al momento de iterar

## enumerate()

`enumerate()`

permite agregar un contador a la iteración, por lo tanto extrae elemento y contador.

## zip()

Permite unir varios iterables para utilizarlos dentro de la misma iteración.

## ¿Cómo salir de un ciclo for?



Para terminar un ciclo **for** a propósito, se debe escribir la palabra **break**, y esta instrucción hace que el ciclo termine y se continúe con la ejecución del resto del programa.





# Ejercicio guiado

"Búsqueda"



## Ciclos anidados

Un ciclo anidado no es más que un ciclo dentro de otro ciclo, y no existe un límite explícito sobre cuántos ciclos pueden haber anidados dentro de un código, aunque por cada uno aumentará la complejidad.





## Quiz

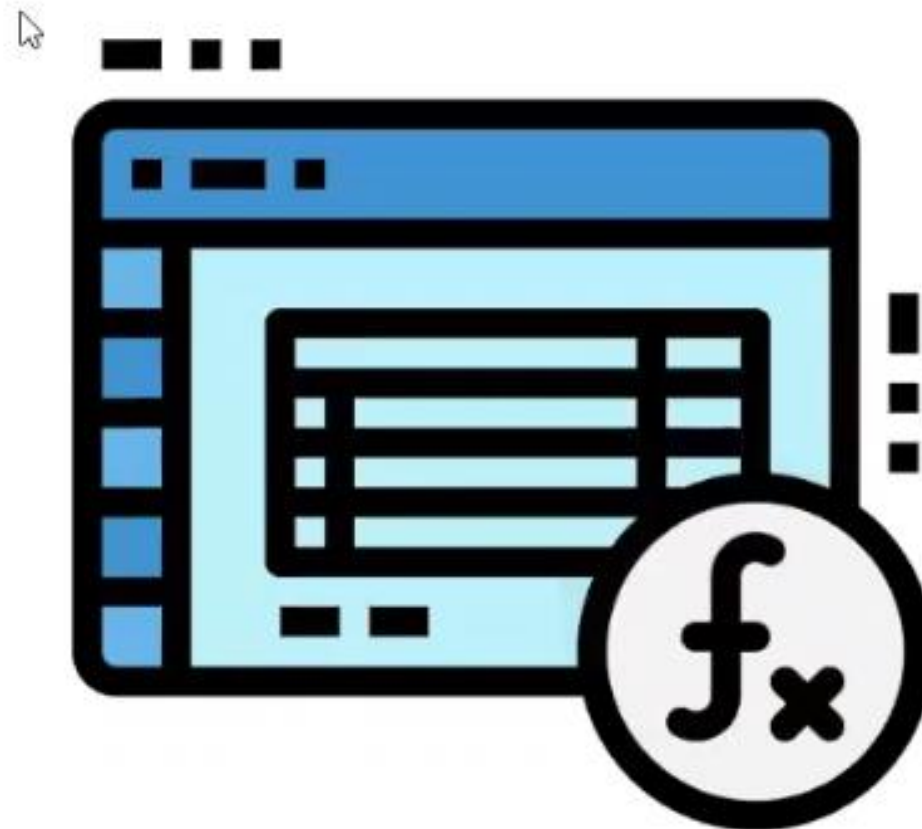




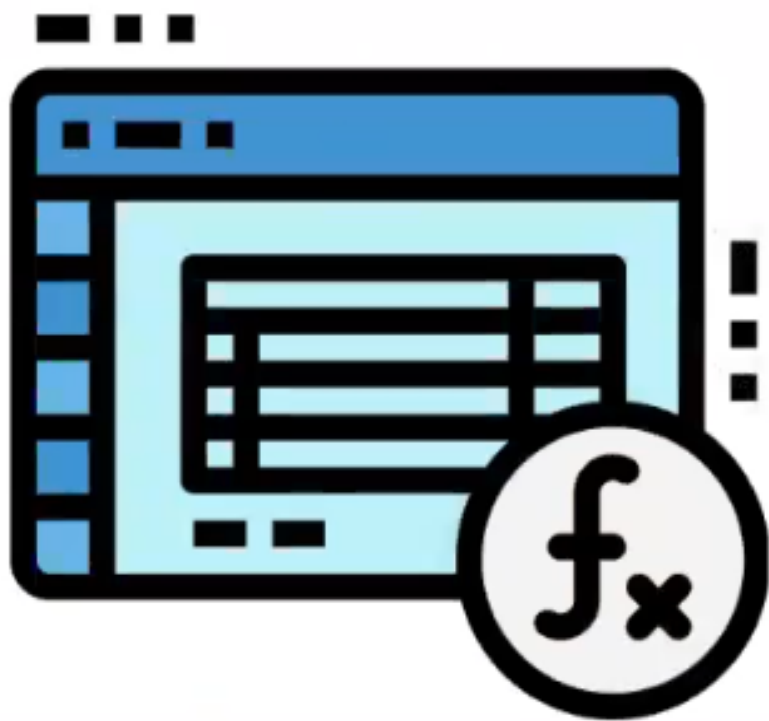
# **/\* Python Comprehensions (Comprensiones) \*/**

# Transformando un ciclo for en un Comprehension

La sintaxis de un List Comprehension es: `[fórmula for variable in iterable]`



# Condicionales con List Comprehensions



Existen otras versiones de un List Comprehension que permiten combinar Control de Flujo condicional con los ciclos `for`.

```
[expresión1 if condición1 else  
expresión2 for variable in  
iterable]
```

## Operaciones de filtrado

Finalmente, los List Comprehensions pueden ser utilizados para filtrar datos de una lista. Por ejemplo, contemos la cantidad de datos de esta lista que son string:

```
lista = ['Lechugas', 'Tomates', 5, 10,  
         True, False, True, 'Papas',  
         5.1, 45.2, 1, 2, 0]
```

```
count_str = 0  
for elemento in lista:  
    if type(elemento) is str:  
        count_str += 1
```

```
print(count_str)
```

3



# Dictionary Comprehensions

Los Dictionary Comprehensions son la equivalencia en Diccionarios de los Python Comprehensions. Recuerda que los diccionarios son estructuras clave-valor por lo que se debe considerar al momento de iterar.





# Quiz



{desafío}  
latam\_



Cierre





¿Qué son los ciclos y por qué son importantes al momento de Programar?



¿Qué son los Iteradores y Acumuladores? ¿Para qué se utilizan?



¿Cual es la diferencia entre un ciclo For y un Ciclo While?



¿Qué son los Python Comprehensions? ¿Para qué sirven y qué tipos hay?

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

[www.desafiolatam.com](http://www.desafiolatam.com)

