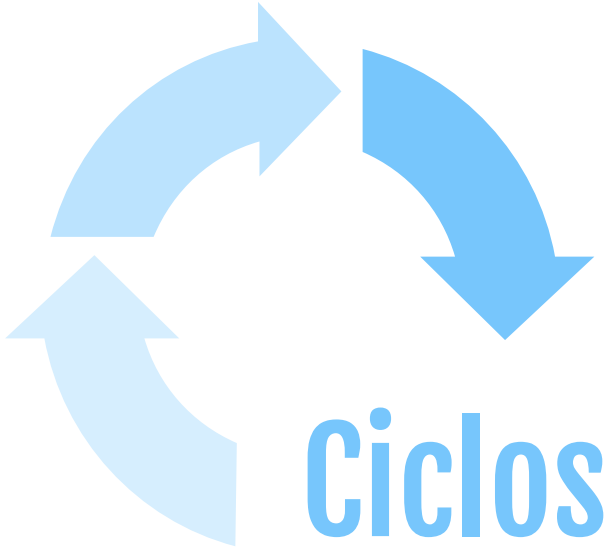


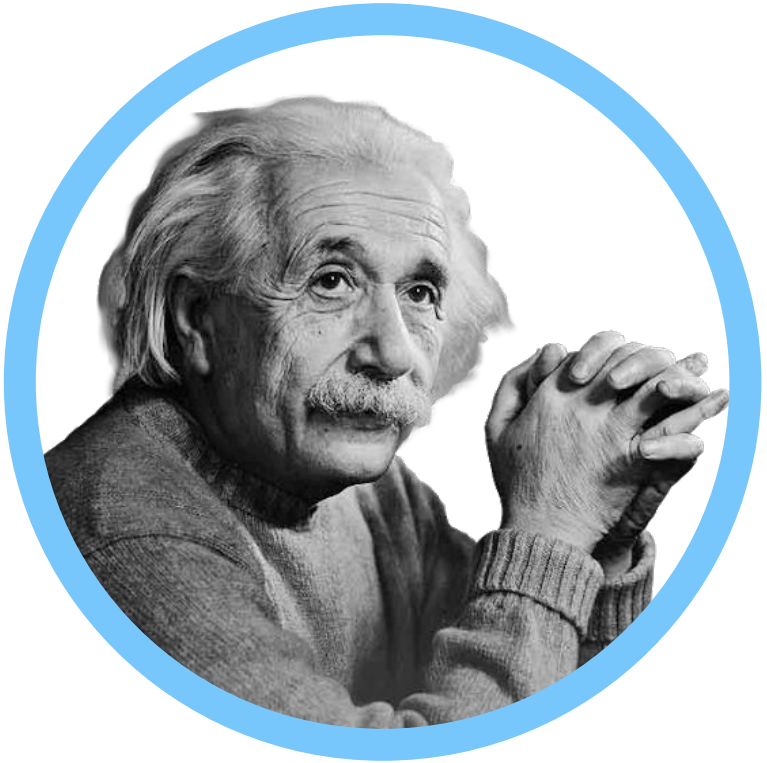
Ciclos

¿Repetir algo y obtener resultados distintos?

Joaquín Badillo

Locura es hacer lo mismo una y otra vez, esperando resultados diferentes.

— Albert Einstein (probablemente)



Joaquín Badillo

La idea principal

El día de hoy vamos a ver cómo hacer que una computadora pueda realizar un procedimiento de forma repetida.

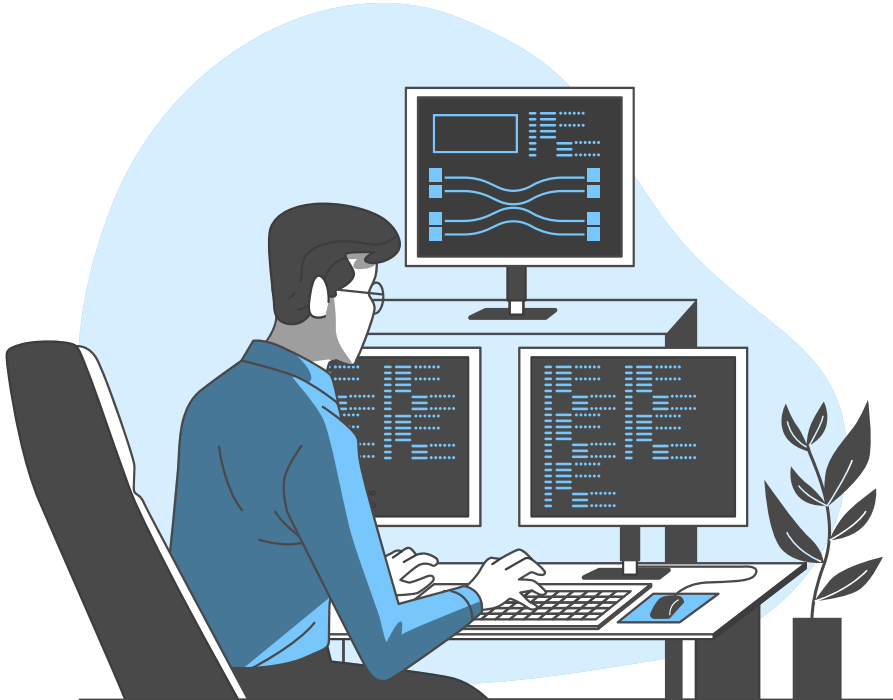
Pero repetir un *procedimiento* no es lo mismo que hacer exactamente la misma acción una cantidad arbitraria de veces.

Por ejemplo hacer las sumas:
1 + 1
1 + 1 + 1
1 + 1 + 1 + 1

Son todas distintas, pero estamos aplicando el mismo procedimiento (sumar 1).

La idea principal

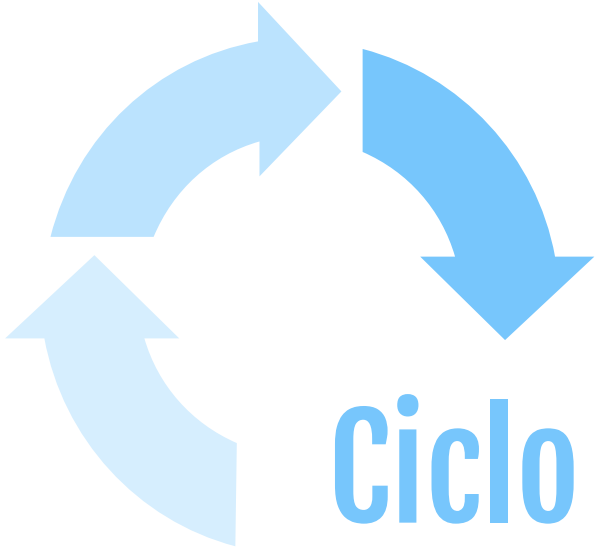
Al programar podemos apoyarnos de las *variables* para tener valores que cambian en cada *iteración*



Joaquín Badillo

También podemos repetir una acción hasta que se cumpla una condición (Por ejemplo pedir una contraseña hasta que sea la correcta)

Joaquín Badillo



Ciclo While

Joaquín Badillo

Ciclo While

Un ciclo *while* repite uno o muchos procedimientos hasta que una expresión lógica (es decir una condición) sea falsa.

En Python

```
while condicional:
        procedimiento1
        procedimiento2
              ⋮
        procedimientoN
```

En español

```
mientras se cumpla esta condición:
        realiza este procedimiento
        y este procedimiento
        etc.
```

Ejemplos en Python

```
itr = 0

while itr < 3:
    print(“Hello loops”)
    itr += 1
```

El resultado que verían sería que se imprime la frase *Hello loops* 3 veces en pantalla. Además se creó una variable, *iter*, cuyo valor al final de la ejecución es 3.

Ejemplos en Python

```
suma = 0

while suma < 10:
    suma += 1

print(suma)
```

En este código se crea una variable, *suma*, que se incrementa de 1 en 1 hasta llegar a 10, después se imprime

Ejemplos en Python

```
password = “secret!”
userPassword= input(“Enter the password: ”)

while userPassword != password:
    print(“Incorrect Password!”)
    userPassword= input(“Enter the password: ”)

print(“Welcome”)
```

Pide al usuario una contraseña y continúa pidiendo la contraseña hasta que estas coincidan. Una vez termina el ciclo se muestra un mensaje de bienvenida

Manos a la obra

Exponenciación con Números Positivos

Actividad:

Crea una función que realice la operación 2^n (para exponentes positivos y enteros), usando **únicamente** multiplicaciones. Esta función debe recibir el exponente como argumento y se llamará pow2(n).

Recuerda que:

$$2^n = \underbrace{2 \cdot 2 \cdot \ \ldots \ \cdot 2}_{n \text{ veces}}$$

Pregunta Detonante

Joaquín Badillo

Pregunta Detonante

Vamos a modificar uno de los ejemplos vistos anteriormente:

```
suma = 0
cond = suma < 10

while cond:
    suma += 1

print(suma)
```

La condición suma < 10 la almacenaré en una variable llamada *cond*.

Ahora utilizaré la variable cond como condición en el ciclo

¿Este programa funciona igual?

Joaquín Badillo

Pregunta Detonante

Este programa de hecho es inútil

```
suma = 0
cond = suma < 10

while cond:
    suma += 1
    cond = suma < 10

print(suma)
```

¡La computadora no almacena la condición como tal, únicamente su valor!

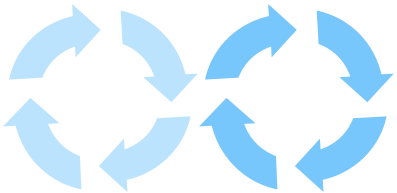
Cuando se creó la variable *cond* su valor era *Verdadero* y seguirá teniendo este valor a menos que dentro del ciclo volvamos a **evaluar** la expresión. Por ende este código crea un ciclo que nunca termina: un ciclo **infinito**.

Joaquín Badillo

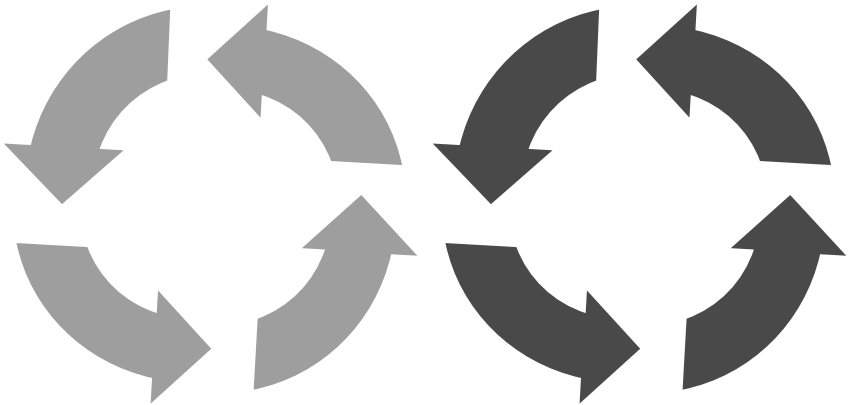
Lección con ciclos

Al trabajar con ciclos tenemos que tener cuidado de checar que la condición se modifique y eventualmente pueda ser falsa, de lo contrario vamos crear un ciclo que nunca acaba.

```
suma = 0
cond = suma < 10

while cond:
    suma += 1
```



¡No es tan obvio!



Joaquín Badillo

Problema Relevante

3n + 1 (también conocido como la Conjetura de Collatz)

Considera el siguiente procedimiento.
Me das un número entero y positivo:
- Si tu número es par, lo divido entre 2
- Si tu número es impar lo multiplico por 3 y le agrego 1.

Ahora, la pregunta interesante...

Imagina que sigo repitiendo este procedimiento con el resultado y termino hasta que llegue al 1. Por ejemplo si me das inicialmente el número 6 todos los números que obtendría hasta detenerme serían:
$6 \longrightarrow 3 \longrightarrow 10 \longrightarrow 5 \longrightarrow 16 \longrightarrow 8 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1$
(Llegué al 1 y por lo tanto me detuve)

¿Este procedimiento eventualmente terminará sin importar que número (entero y positivo) me diste inicialmente?

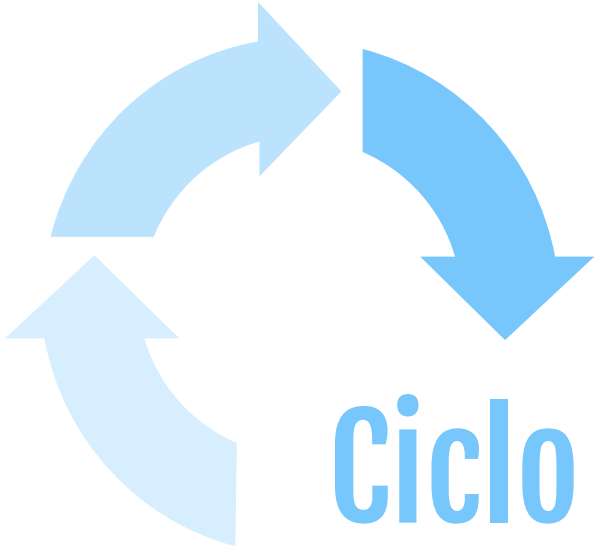
En Python:

```
def f(n):
    while n != 1:
        if n % 2 == 0:
            n /= 2
        else:
            n = 3*n + 1
```

¿Esta función siempre terminara para una entrada (n) entera y positiva?

Nadie a podido responder esta pregunta todavía.
(Una respuesta requiere de una demostración de por qué siempre terminará o bien de un ejemplo en el que no termine)

Joaquín Badillo



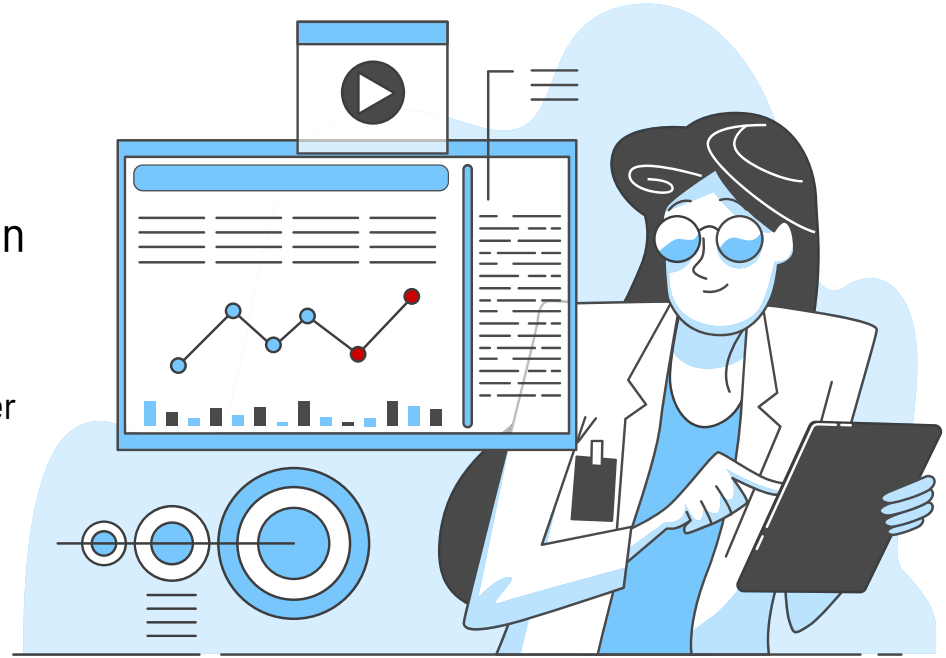
Ciclo For

Joaquín Badillo

Ciclo For

En *Python* un ciclo *for* repite uno o muchos procedimientos hasta terminar de *recorrer* un *iterable*.

¡Los ciclos *for* en otros lenguajes de programación suelen ser distintos!

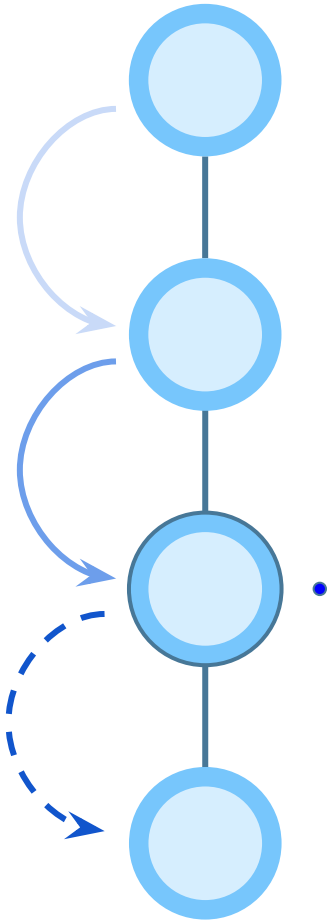


Glosario de Términos

Estructura de datos: Una manera de almacenar datos de forma organizada, con un conjunto de operaciones para utilizarla (acceder a los datos, agregar datos, eliminar...).

Recorrido: Pasar por cada uno de los elementos de una *estructura de datos* con algún tipo de orden.

Iterable: Estructura de datos que se puede recorrer. (listas, strings, diccionarios, tuplas, rangos, conjuntos...)



Joaquín Badillo

Joaquín Badillo

Ciclo For

Sintaxis de Python

for *variable* in *iterable*:
    procedimiento1
    procedimiento2
    ⋮
    procedimientoN

En Español

por cada elemento en el iterable:
    realiza este *procedimiento*
    y este otro *procedimiento*
    etc.

Ejemplos en python

```
nums = [“Tres”, “Dos”, “Uno”]

for value in nums:
    print(value)
```

Se tiene una lista llamada *nums* que contiene los *strings* “Tres”, “Dos” y “Uno”. Con un ciclo for se imprime cada uno de los valores contenidos en la lista, por lo que en pantalla se ve una cuenta regresiva.

Ejemplos en python

```
suma = 0
values = [1, 2, 3, 4, 5]

for value in values:
    suma += value

print(suma)
```

Se tiene una lista llamada *values* que contiene los números naturales del 1 al 5. Además se tiene una variable *sum* cuyo valor es inicialmente 0. Usando un ciclo for se agrega cada uno de estos valores a *sum* y se imprime el resultado. En pocas palabras sumamos los números naturales hasta el 5.

Ejemplos en python

```
suma = 0

for value in range(6):
    suma += value

print(suma)
```

Para no tener que crear listas de números a mano, *Python* trae una función llamada *range*. Si a esta función le das un número entero positivo (llamémoslo *n*) te dará una especie de lista (en realidad es una estructura nueva llamada rango), que empieza en 0 y termina en n-1.

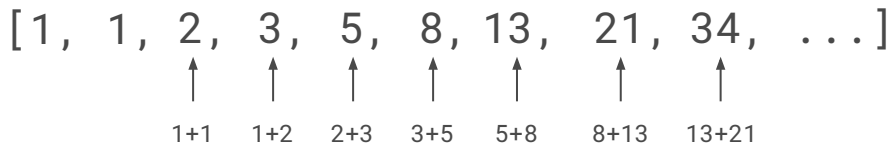
Ejemplos en python

```
def fib(n):
    prev = 1
    current = 0

    for i in range(n):
        temp = current
        current += prev
        prev = temp

    return current
```

Pronto volveremos a encontrarnos esta función…

Esta es una función para obtener el *n*-ésimo número de Fibonacci.

Los números de Fibonacci son una sucesión (una “lista”) de números que se obtiene empezando con los números 1 y 1, tal que el siguiente número se obtiene sumando los 2 anteriores:



Manos a la obra

Actividad: *Lista de Tareas*

Joaquín Badillo

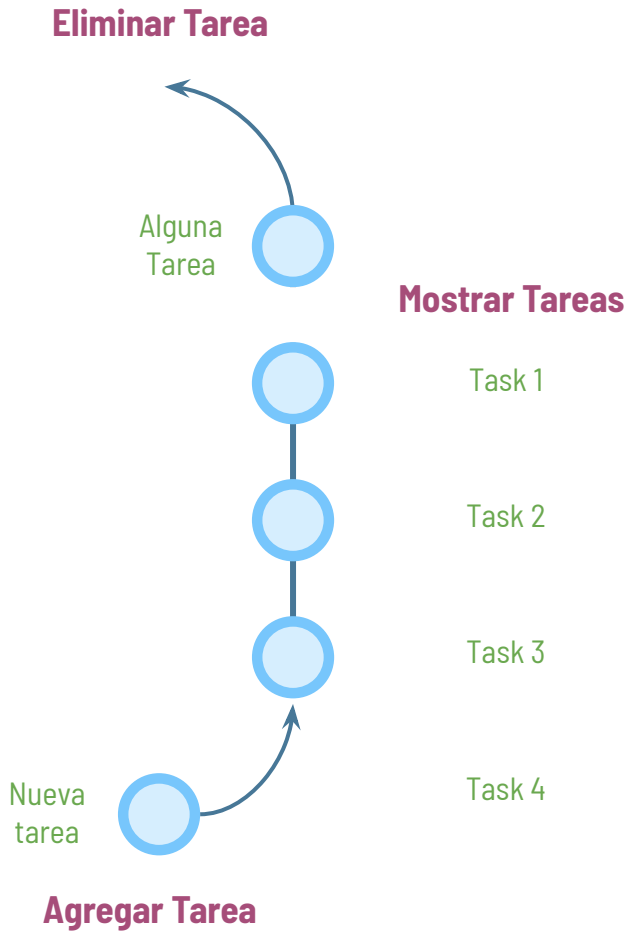
Hoy vamos a crear una lista de tareas muy sencilla. Primero debemos observar que de forma general cualquier lista de tareas soporta 3 funciones básicas:

Agregar Tarea **Eliminar Tarea** **Mostrar Tareas**

addTask(tasks, task)
La función de agregar tarea toma una lista (la lista de tareas) y un string (la tarea) y agrega la tarea a la lista.

removeTask(tasks, task)
La función de eliminar tarea toma una lista (la lista de tareas) y un string (la tarea) y elimina la tarea de la lista.

printTasks(tasks)
La función de mostrar tareas toma una lista (la lista de tareas) y muestra por pantalla todas las tareas.



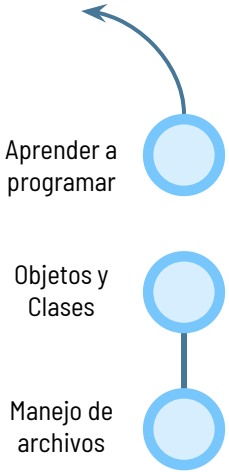
Nuestra lista de tareas

Propongo la siguiente lista de tareas para el curso:

1. Aprender a programar
    - 1.1. Aprender cómo darle instrucciones a una computadora
    - 1.2. Aprender un poco acerca de eficiencia

2. Aprender acerca de programación orientada a objetos
    - 2.1. Crear una clase que represente la tarea
    - 2.2. Crear una clase que represente la lista de tareas
        - 2.2.1. ¿Podemos agregar subtareas como estoy haciendo aquí?

3. Aprender a guardar información de forma permanente
    - 3.1. Manejo de Archivos
    - 3.2. Guardar las tareas en un archivo



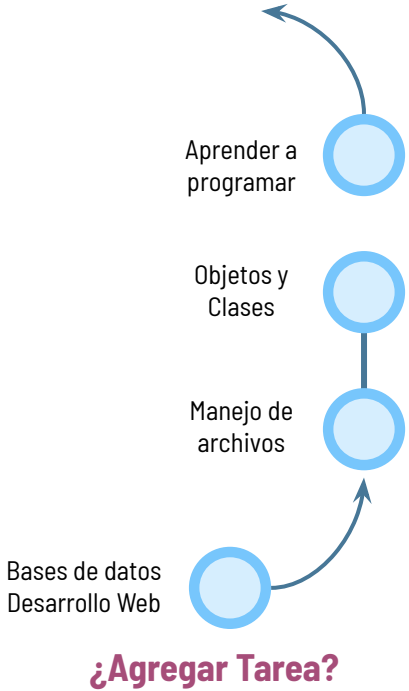
Nuestra lista de tareas

Terminando el curso, espero que puedan investigar acerca de otras herramientas y quizá continuar con este proyecto.

Por ejemplo, podrían investigar acerca de bases de datos para no usar un archivo de texto o csv normal.

También pueden aprender acerca de Desarrollo Web y hacer que su lista de tareas tenga un *frontend*.

Si quieren combinan ambos, una base de datos y una página web, pueden aprender acerca de backend y la creación de APIs que permiten mandar información entre programas.



Joaquín Badillo

Pregunta detonante:

¿Se puede tener un ciclo for infinito en Python?

Joaquín Badillo

Lista Circular

Una lista circular es una lista que cuando llega al “final” regresa al primer elemento.

```
from itertools import cycle

circularList = cycle([1, 2, 3])

for num in circularList:
    print(num)
```

Estoy agregando la función *cycle* que otras personas hicieron. Esta función implementa una lista circular (formalmente es un iterador)

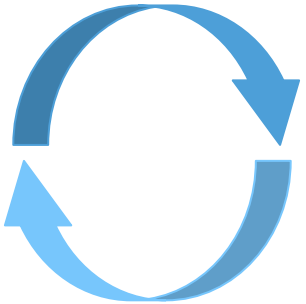
Los ciclos infinitos también son posibles en los *ciclos for* de Python, pero es muy raro encontrar este problema de forma inesperada.

Joaquín Badillo

¿Quién usaría una lista circular?

Imagina una lista de canciones (una *playlist*).

Ahora supongamos que haces una aplicación que reproduce la música en esta playlist. (Cómo hacer que un programa pueda reproducir la música esta un poco fuera del enfoque, pero lo pueden buscar, seguro hay librerías)



Podrías hacer que la aplicación deje de reproducir canciones una vez que pasaron todas. O bien podrías hacer que **cicle**. Una lista circular implementa esta segunda funcionalidad!

Las listas circulares también pueden ser útiles para algo conocido como *Scheduling*, que permite que muchos procesos se lleven a cabo secuencialmente, esto es importante en *Sistemas Operativos*

Atribución

Ilustraciones de *Stories*
Plantilla inspirada en *Technology Consulting* de *Slidesgo*