INFORMATORIO

INTRODUCCIÓN

En este apartado, y para terminar con la parte de estructuras de control de flujo, vamos a mostrarte lo que son las estructuras repetitivas o también llamadas iterativas o cíclicas.

Estas estructuras de control se utilizan para ejecutar un bloque de código varias veces, lo cual es muy útil cuando se necesita realizar una tarea repetitiva.

Por el momento, podíamos dar una condición y el programa tomaba un camino a seguir, sin embargo, una vez finalizado el bloque de código ejecutado, el programa seguía leyendo las líneas de código que continuaban. Pero con las estructuras que vamos a ver, el programa va a poder repetir las veces que necesitemos el bloque de código, dentro de nuestra estructura.

Cada conjunto de instrucciones a ejecutar se denomina **BUCLE**
y cada repetición del bucle se llama **ITERACIÓN**.

Existen dos tipos de estructuras de control repetitivas en Python: el bucle "**while**" y el bucle "**for**".

while

El bucle "while" se utiliza cuando se desea ejecutar un bloque de código mientras se cumpla una determinada condición. La estructura básica de un bucle "while" es la siguiente:

```
1   while condicion: #con la misma estructura que if
2       #bloque de código
```

Y como vemos, funciona con la misma estructura que vimos **if.**

Primero la palabra reservada, en este caso **while**, la condición que vamos a darle para que el programa decida si entrar al bucle o no, luego los dos puntos y, para comenzar a escribir el código que queremos que se ejecute, siempre indentamos.

Ahora, como buen observador, te habrás dado cuenta que si lo pasamos a español, while sería "mientras", por lo que podemos leer como "mientras que esta condición se cumpla, se ejecutará el bloque de código". Así que esta estructura va a ser ejecutada siempre que se retorne un valor True, y

solamente terminará, cuando se retorne un valor False. Vamos con un ejemplo para que lo comprendas mejor.

```
1  i = 1
2  while i <= 5:
3      print(i)
4      i += 1
```

```
PROBLEMAS    SALIDA    CON

1
2
3
4
5
```

¿y qué pasó acá? Antes que nada, vamos a explicarte estas dos nuevas palabras que mencionamos antes (y de paso vamos incorporarlas a nuestro diccionario de programadores). Una palabra es ITERACIÓN y la otra es BUCLE. Cuando hablamos de **bucles**, hablamos de iterar, y nos referimos a repetición, es decir, el **bucle** va a **iterar** (repetirse) las veces necesarias mientras nuestra condición sea True.
Este BUCLE va a ser nuestro bloque de código con las instrucciones que vamos a ejecutar. Cuando la condición se vuelva False, el bucle finaliza, o la expresión correcta sería, "el **bucle while** completó todas las iteraciones".

Ahora ya podemos analizar este ejemplo. Primero que nada, estamos inicializando una variable con un valor entero de 1. Luego como condición, vamos a decirle al programa que mientras que el valor de nuestra variable sea menor o igual a 5 ejecute nuestro bucle.
Este bloque de código muestra por pantalla el valor que tiene nuestra variable por cada iteración que realice. Y, por último, para que nuestra variable aumente su valor con cada iteración le sumamos 1. De esta forma en cada iteración nuestra variable aumentará su valor de 1 en 1 y será verificada en la condición. Una vez que la condición retorna False, el bucle finaliza.

Pero antes de seguir, vamos a ver unos nuevos tipos de variables especiales que (como en el ejemplo que acabamos de ver), vamos a requerir al momento de generar nuestros bucles.

Contador

Como vimos en el ejemplo anterior, la variable i que usamos es un contador. Esta variable especial se utiliza para poder controlar, mediante el incremento de su valor, la cantidad de iteraciones que vamos a realizar.

Pensemos, ¿si no utilizáramos el contador en el ejemplo anterior, como hubiésemos finalizado el bucle? No lo podríamos haber finalizado y eso habría generado un *bucle infinito*.

> *Nos referimos a **bucle infinito** cuando este no puede finalizar ya que su ejecución no está controlada correctamente para finalizar en un momento dado.*

Entonces usamos contadores para llevar un registro de las iteraciones que hacemos o queremos hacer, ya sea para controlar nuestro bucle o para llevar registro de algo específico.

Vamos con otro ejemplo donde vamos a usar un contador para llevar un registro de la cantidad de veces que aparecen números pares en una lista, y otro contador para controlar nuestro bucle.

Como aclaración: un contador puede inicializarse en el número que deseemos, pero siempre tiene que ser inicializado (es decir, darle un valor inicial), y en general, usamos enteros como tipo de valor. Sin embargo, un

contador puede ser con float, ya que podemos sumar o restar décimas, centésimas, milésimas, etc., siempre según lo que queramos realizar.

```python
lista = [1, 2, 3, 4, 5]
pares = 0
i = 0

while i < len(lista):
    if lista[i] % 2 == 0:
        pares += 1
    i += 1

print(f'La cantidad de números pares hallados en la lista es de: {pares}')
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  Python

La cantidad de números pares hallados en la lista es de: 2
```

En este ejemplo arrancamos nuestro programa confeccionando una lista con valores enteros. Luego inicializamos los dos contadores que vamos a utilizar en 0 (cero).

Nuestro contador de nombre "pares" va a llevar un registro de las veces que aparezcan números pares en nuestra lista.

El contador con nombre "i" va a llevar el registro de iteraciones.

La condición que tiene while es que mientras que el contador "i" sea menor a la cantidad de elementos que tiene la lista (que son 5), tiene que realizar las iteraciones correspondientes.

El contador "i" entonces, irá incrementado su valor en 1 por cada iteración que se realice.

Cuando el contador "i" llegue a 5, while retornará un valor False, por lo que el bucle se detendrá.

Mientras que el contador con nombre "pares" irá sumando 1 si en las iteraciones se encuentra un número par (usando la estructura condicional if). Para hallar un número par, estamos usando el operador matemático (que vimos en los primeros apuntes) de módulo para verificar el resto de la división. Si este resto es igual a 0, entonces el número es par, por lo que se suma 1 a "pares" y con eso llevamos un registro de los números pares que encontramos en cada iteración.

Por último, al finalizar nuestro bucle while, mostramos por pantalla la cantidad de números pares hallados en nuestra lista.

¿Ya lo vas comprendiendo mejor? En este ejemplo que te mostramos para un contador, también usamos en conjunto con la estructura while, la estructura if, por lo que ya podés ir relacionando aún más las formas de usar una estructura con otra, para pulir tus programas.

Ahora vamos a ver otro tipo de variable especial.

Acumulador

La función de esta variable nos resulta muy útil cuando queremos almacenar el resultado de operaciones.

La principal diferencia con el contador es que el incremento o decremento es variable en lugar de constante.

En otras palabras, usamos acumuladores para acumular (o sumar, multiplicar, concatenar, etc) los valores de una lista, tupla, diccionario u otro iterable.

Te vamos a dar un ejemplo sumando los elementos de una lista mediante while:

```
1  lista = [1, 2, 3, 4, 5]
2  suma = 0
3  i = 0
4
5  while i < len(lista):
6      suma += lista[i]
7      i += 1
8
9  print(f'El acumulador tiene la suma de: {suma}')
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

El acumulador tiene la suma de: 15
```

Como podés ver en este ejemplo, usamos un acumulador, que es la variable “suma”, para sumar los valores que se registraron en cada iteración del bucle while, que fue controlado por el contador “i”, el cual se usa como índice para iterar sobre la lista.

Bandera

Y, por último, llegamos al apartado en el que vamos a usar valores booleanos para marcar si se ha cumplido una condición.

Vamos a ver un ejemplo para que lo entiendas mejor.

```python
lista = [1, 2, 3, 4, 5]
nro_buscado = 3
encontrado = False
i = 0

while i < len(lista):
    if lista[i] == nro_buscado:
        encontrado = True
        break
    i += 1

if encontrado:
    print('El número se encuentra en la lista.')
else:
    print('El número no se encuentra en la lista.')
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

El número se encuentra en la lista.
```

En el ejemplo usamos nuevamente la variable "i" como índice para iterar sobre la lista. La variable "encontrado" es la que hace de "bandera" para marcar si el número que estábamos buscando lo hemos encontrado, ya que la inicializamos esta variable con False y si el número se encuentra en la lista, este valor cambia a True. Luego se establece una función nueva que vamos a utilizar para terminar el bucle. Esta función es break.

Una vez que, en este caso, se cumple la condición, ya que encontramos el número, ya podemos finalizar el bucle y mostrar por pantalla que el número fue encontrado.

Si, en tal caso, no se cumpliese la condición que establecimos, en este caso la búsqueda de un número, lo que se mostraría por pantalla, sería que el número no se encontró.

While (mientras)

Una vez explicadas estas variables especiales, volvemos al bucle while.
Este bucle nos permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición retorne el valor True).
Mediante las instrucciones que le demos a este bucle para controlarlo, vamos a poder decidir cuándo se detendrá (como hicimos en los ejemplos con la variable "i" o las variables especiales).

Características del bucle while:

- No se conoce la cantidad de veces a iterar o repetir el conjunto de acciones
- El final del bucle está controlado con una condición.
- El conjunto de acciones se ejecutan mientras la evaluación de la condición devuelva un resultado verdadero (True)
- El ciclo se puede ejecutar 0 o más veces.

Vamos a ver otro ejemplo, integrando ejemplos anteriores, donde vamos a mejorar el login de usuario que vimos antes donde habíamos pedido al usuario que ingrese su usuario y contraseña. Sin embargo, en ese ejemplo si el usuario ingresaba mal alguno de los dos, debía reiniciar el programa. Con while vamos a poder mejorarlo de la siguiente manera.

```
1  usuario = input('Ingresá el usuario: ')
2  contrasena = input('Ingresá la contraseña: ')
3
4  while usuario != 'Informatorio' and contrasena != 'Info 2023':
5      print('El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.')
6      usuario = input('Ingresá el usuario: ')
7      contrasena = input('Ingresá la contraseña: ')
8  else:
9          print('Ingresaste correctamente. Bienvenido.')
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

Ingresá el usuario: sdfs
Ingresá la contraseña: sdfsdf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfds
Ingresá la contraseña: sdfsdf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfsdf
Ingresá la contraseña: sdfsdf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfefefes
Ingresá la contraseña: rgfgfdfg
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: Informatorio
Ingresá la contraseña: Info2023
Ingresaste correctamente. Bienvenido.
```

Ya estamos mejorando nuestro programa de login que vimos antes, y esto gracias a nuestro bucle while. Además, te mostramos que while acepta **else**, por lo que podemos, como en estos casos, realizar una comprobación las veces que sean necesarias mientras se retorne un valor True mostrando un mensaje en particular, pero mostrar otro mensaje cuando se retorne un valor False.

Con esto podés ver lo poderoso que puede ser while cuando no sabemos la cantidad de veces que necesitamos iterar una condición. Además, con el uso de else en while, podemos dar otro tipo de mensajes, o realizar acciones cuando finaliza el bucle.

Te recordamos que hay que ser cuidadosos al momento de usar while, ya que, si las condiciones que damos no están bien estipuladas, se puede crear un bucle infinito, debido a que el bucle no va a saber cuándo detenerse.

for

El ciclo "for" se utiliza para iterar sobre una secuencia de valores, como una lista, una tupla, un diccionario o un conjunto. En cada iteración, el ciclo asigna el valor actual de la secuencia a una variable y ejecuta un conjunto de instrucciones que se encuentran dentro del bloque de código del ciclo for.

La sintaxis básica del ciclo for en Python es la siguiente:

```
1   for variable in secuencia:
2       # Bloque de código a ejecutar en cada iteración
```

En esta sintaxis, variable es la variable que se utiliza para almacenar el **valor actual** de la secuencia en cada iteración, y secuencia es la **secuencia que se va a iterar**. El bloque de código que sigue al ciclo for se ejecutará en cada iteración del ciclo, utilizando el valor actual de variable.

Vamos a ver algunos ejemplos para que comiences a entenderlo mejor:

INFORMATORIO

Iterando sobre una lista

Supongamos que tenemos una lista de números enteros y queremos imprimir cada número en la lista. Podemos utilizar un ciclo for para iterar sobre la lista de la siguiente manera:

```
1   numeros = [1, 2, 3, 4, 5]
2
3   for numero in numeros:
4       print(numero)
```

```
PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN

1
2
3
4
5
```

En este ejemplo, "numeros" es la lista que queremos iterar y "numero" es la variable que utilizamos para almacenar el valor actual de la lista en cada iteración. En cada iteración del ciclo for, el valor actual de numero se imprimirá en la consola.

Así vemos que la principal diferencia con while es que en for tenemos definido, desde un primer momento, la cantidad de veces que vamos a iterar, y, por eso no usamos el método len() como fue en el caso de while para hacer una estructura similar.

Vamos a ver algunas de sus características:

- El ciclo for se utiliza para iterar sobre una secuencia de valores, como una lista, tupla, diccionario, conjunto, cadena de texto o rango de valores.
- En cada iteración del ciclo, el valor actual de la secuencia se asigna a una variable que se especifica después de la palabra clave for.
- El bloque de código dentro del ciclo for se ejecuta una vez para cada valor en la secuencia.
- La secuencia sobre la que se itera puede ser modificada dentro del ciclo for, pero esto puede tener efectos inesperados y se debe hacer con precaución.
- Podemos utilizar la palabra clave *break* para detener la ejecución del ciclo for en cualquier momento.
- Podemos utilizar la palabra clave *continue* para saltar la iteración actual del ciclo y pasar a la siguiente.
- Podemos utilizar la función range() para crear un rango de valores que se puede iterar en un ciclo for.
- Podemos anidar ciclos for para iterar sobre múltiples secuencias al mismo tiempo.
- Podemos utilizar el operador *enumerate*() para iterar sobre una secuencia y obtener tanto el valor como el índice de cada elemento.

INFORMATORIO

Iterando sobre una cadena de texto

Vamos con otro ejemplo donde podemos utilizar un ciclo for para iterar sobre una cadena de texto. En este caso, el ciclo iterará sobre cada carácter de la cadena.

```
1   texto = "Hola mundo"
2
3   for letra in texto:
4       print(letra)
```

```
PROBLEMAS     SALIDA     CONSOLA DE DEPURACIÓN

H
o
l
a

m
u
n
d
o
```

En este ejemplo, "**texto**" es la cadena de texto que queremos **iterar** y "**letra**" es la variable que utilizamos para almacenar el valor actual de la cadena en cada iteración. En cada iteración del ciclo for, el valor actual de letra se imprimirá en la consola

Iterando sobre un diccionario

También podemos realizar iteraciones sobre diccionarios como vamos a ver a continuación:

```
1   estudiantes = {
2       "Juan": 18,
3       "María": 20,
4       "Pedro": 19
5   }
6
7   for estudiante in estudiantes:
8       print(estudiante)
```

```
PROBLEMAS     SALIDA     CONSOLA DE DEPURACIÓN

Juan
María
Pedro
Juan
María
Pedro
```

Como vemos acá "**estudiantes**" es el diccionario que queremos **iterar** y "**estudiante**" es la variable que utilizamos para **almacenar la clave actual** del diccionario en cada iteración.
En cada iteración del ciclo for, el valor actual de estudiante se imprimirá en la consola.

INFORMATORIO

Iterando sobre un rango de valores

Otra función es la de utilizar un ciclo for para iterar sobre un rango de valores. En este caso, el ciclo iterará sobre todos los valores en el rango especificado. Y con esto vamos a utilizar una nueva función que es range().

```
1   for i in range(1, 6):
2       print(i)

PROBLEMAS     SALIDA     CONSOLA DE DEPURACIÓN

1
2
3
4
5
```

Como podés ver la función range genera un rango de valores desde 1 hasta 5 inclusive (recordá que toma un valor antes del último número que le pasamos). El ciclo for iterará sobre cada uno de los valores en este rango, y en cada iteración, el valor actual se almacenará en la variable i, que se imprimirá en la consola.

Una las ventajas de utilizar tipos range() es que el argumento del tipo range() controla el número de veces que se ejecuta el bucle.

Sirve para generar una lista de números que podemos recorrer fácilmente, pero no ocupa memoria porque se interpreta sobre la marcha.

Control de bucles

Utilizando break y continue

Dentro del bloque de código del ciclo for, podemos utilizar las palabras clave **break** y **continue** para controlar el flujo de ejecución del ciclo.

La palabra clave **break** se utiliza para detener la ejecución del ciclo for en el punto en que se encuentra. Por ejemplo:

```
1   numeros = [1, 2, 3, 4, 5]
2
3   for numero in numeros:
4       if numero == 3:
5           break
6       print(numero)

PROBLEMAS     SALIDA     CONSOLA DE DEPURA

1
2
```

Acá utilizamos un condicional para comprobar si el valor actual de "numero" es igual a 3. Si es así, utilizamos la palabra clave break para detener la ejecución del ciclo. Si no, el valor actual de "numero" se imprimirá en la consola.

Entonces sentencia *break* es usada también para terminar un ciclo aun cuando la evaluación de la condición no devuelva *False*.

La palabra clave **continue** se utiliza para saltar la iteración actual del ciclo y pasar a la siguiente. Por ejemplo:

```
1   numeros = [1, 2, 3, 4, 5]
2
3   for numero in numeros:
4       if numero == 3:
5           continue
6       print(numero)
```

```
PROBLEMAS    SALIDA    CONSOLA DE DEPURACI

1
2
4
5
```

Utilizamos un condicional para comprobar si el valor actual de "numero" es igual a 3. Si es así, utilizamos la palabra clave continue para saltar la iteración actual del ciclo y pasar a la siguiente. Si no, el valor actual de "numero" se imprimirá en la consola.

Entonces la sentencia *continue* regresa al comienzo del bucle, ignorando todos los elementos que quedan en la iteración actual del bucle e inicia la siguiente iteración.

Para simplificarlo, al encontrar el valor buscado, *continue* salta ese valor y sigue iterando por el siguiente.

Para finalizar, te habrás dado cuenta lo poderosas que son estas herramientas de estructuras de control de flujo y el potencial que tienen para realizar nuestras aplicaciones. Vamos a utilizarlas siempre como programadores, por lo que lo que te recomendamos siempre… practicar, practicar y practicar.

Podés ir mejorando los ejemplos de código que veníamos haciendo, implementando estas nuevas herramientas.