

➤ Estructuras de control de flujo

Estructuras de control - condicionales



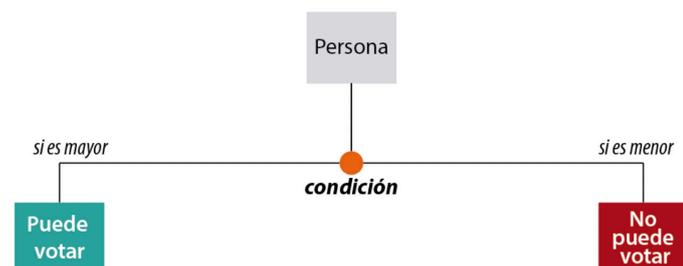
INTRODUCCIÓN

Como has visto hasta ahora, estuvimos ejecutando instrucciones de código de forma plana, esto quiere decir que, el código se estaba leyendo línea por línea una tras otra, de forma secuencial.

Pero, ¿qué pasa cuando queremos el programa "tome una decisión" para realizar una acción u otra en base a instrucciones/parámetros que le damos?

Por ejemplo: queremos saber si una persona es mayor o menor de cierta edad y que en base a esto pueda votar o no.

En un caso así, necesitamos que el programa haga una cosa si se cumple que la persona es mayor de una edad o, que realice otra cosa si la persona es menor de esa edad.



Como puedes ver, mediante una condición, la cual contiene una pregunta, el programa va a tomar una forma de avanzar u otra. Es decir, gracias a las estructuras de control de flujo, el programa va a saber como "actuar" ante una determinada situación, e incluso, repetir una tarea si es necesario.

Los tipos de estructuras de control que veremos son:

Las listas son una colección ordenada y mutable de elementos que pueden ser de diferentes tipos de datos.

CONDICIONALES

REPETITIVAS

Estructuras de control condicionales

Estas estructuras realizan la evaluación de una condición y, según el resultado, el programa va a realizar una determinada acción.

Para dar estas condiciones, utilizamos expresiones lógicas. La estructura que veremos es:

IF

```
1 if True:
2     print('Voy a hacer esta tarea.')
3     print('Se cumplió esta condición.')
4
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL**

Voy a hacer esta tarea.
Se cumplió esta condición.

Entonces vemos que para que la condición se cumpla, la expresión que se comprueba tiene que ser verdadera (True). Si la expresión resulta falsa (False), es decir, no se cumple, el programa sigue su recorrido sin ejecutar lo que se encuentra dentro del IF



Analicemos la estructura de if (que por si aún no te has dado cuenta, significa SI en inglés). Esta estructura consta de 3 partes puntualmente:



Veamos en código, el ejemplo de la persona que puede votar o no para que lo puedas comprender mejor:

```
1 edad = 19
2 if edad >= 16:
3     print('Usted puede votar')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL**

Usted puede votar

Bloque de código

Observamos que si el valor de la variable "edad" es mayor o igual a 16 (es decir, True, como en este caso), se ejecuta el bloque de código que hay dentro de if. Si esto no fuese verdad (es decir False), no se ejecutaría lo que hay dentro del if y el programa seguiría con las siguientes líneas de código, pasando por alto todo el bloque de código dentro del if. Además, si has notado, usamos un operador comparación para la condición.

Otra parte muy importante en la sintaxis para que el código se ejecute correctamente, es la indentación

Este espacio que hay entre el comienzo de la línea de código y el código que se escribe, se llama **indentación**

```
1 edad = 19
2 if edad >= 16:
3     print('Usted puede votar')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL**

Usted puede votar

La importancia de la indentación es fundamental para que el código dentro del bloque se pueda ejecutar. Si no se indenta el bloque de código dentro de la sentencia, dará un error y no se podrá continuar con el programa.

CORRECTO

```
1 edad = 19
2 if edad >= 16:
3     print('Usted puede votar.')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL**

Usted puede votar.

INCORRECTO

```
1 edad = 19
2 if edad >= 16:
3 print('Usted puede votar.')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL**

IndentationError: expected an indented block after 'if' statement on line 2
PS C:\Users\PC\Desktop\Info2023> & "C:/Program Files/Python311/python.exe" as_de_control.py
File "c:\Users\PC\Desktop\Info2023\estructuras_de_control.py", line 3

If encadenados

Otra forma en la que podemos usar if es encadenando varios. Esta opción es muy específica para ciertos casos, ya que hay otras funciones que ofrece if (las cuales veremos más adelante).

```

1  edad = 15
2  if edad >= 16:
3      print('Usted puede votar')
4  if edad < 16:
5      print('Usted NO puede votar')

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

Usted NO puede votar

Para un caso como este estamos “forzando” a que se ejecute uno de los bloques, según la condiciones que estamos dando. Como el valor de nuestra variable es 15, cuando el programa verifica la condición en el primer if donde obtiene un valor False, por lo que no ejecuta el bloque de código que se encuentra dentro y sigue con las siguientes líneas de código. Se encuentra con otro if y vuelve a verificar la condición. Al obtener un valor True, ejecuta el bloque de código que encuentra dentro de este segundo if.

if anidados

Otra opción para usar if es la de anidar. Esto quiere decir, que ponemos poner un if dentro de un bloque de código de otro if. Esto sucede cuando necesitamos hacer dos o más comparaciones para brindar mensajes distintos, pero a la vez relacionados, como en el ejemplo que veremos.

```

1  edad1 = 16
2  edad2 = 21
3  if edad1 >= 16:
4      print('Usted puede votar.')
5      if edad2 >= 21:
6          print('Usted puede viajar fuera del país.')

```

anidados

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

Usted puede votar.
Usted puede viajar fuera del país.

Para este caso, en ambos condicionales if obtenemos un valor True, y como tal, ambos mensajes se muestran. Si, en tal caso, en el if que está dentro del bloque de código del primer if (o podríamos decir, del if más a la izquierda según la indentación) obtuviésemos False, el segundo mensaje no se mostraría, y, si en el primer if obtuviésemos False, ninguno de los mensajes se mostraría.

¿Lo vas comprendiendo mejor? Copiá el código en tu editor y comenzá a cambiar los valores de las variables para practicar y entenderlo aún mejor.

Algo importante a destacar, es el uso de la indentación, ya que si no estuviese correctamente indentado, no funcionaría este bloque de código. ¡Practicando y, a prueba y error vas a comprenderlo cada vez mejor!

Uso de operadores lógicos con if

En el caso anterior, necesitábamos brindar mensajes distintos, según la condición, pero que estaban relacionados de todas formas. Sin embargo, hay casos en los que necesitamos hacer más de una comparación o necesitamos más de una condición para que se realice una acción, por eso vamos a tomar uno

de los ejemplos más cotidianos que se suelen ver:

```

1 usuario = 'Informatorio'
2 contraseña = 'Info2023'
3
4 if usuario == 'Informatorio' and contraseña == 'Info2023':
5     print('Acceso correcto. Bienvenido')
6
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
Acceso correcto. Bienvenido
    
```

En el ejemplo anterior has visto como funcionaría básicamente un acceso de usuario y contraseña. Usando un operador lógico, en este caso and, para verificar dos condiciones a la vez.

Esto no se limita a solo dos condiciones, pueden haber más, pero siempre hay que respetar la estructura que venimos mostrando en los ejemplos. También se puede usar el operador or o el operador not, o se pueden combinar todos.

Lo importante es pensar en la condición que queremos dar, pensar en lo que queremos que evalúe el programa según nuestra necesidad y en base a eso, armar las condiciones que correspondan. ¡Con práctica lo irás a comprendiendo mejor! Y para que sigas practicando te dejaremos las tablas de operadores lógicos y los resultados que se obtienen según las combinaciones.

AND		
VARIABLES		RETORNO
a	b	y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OR		
VARIABLES		RETORNO
a	b	y
FALSE	FALSE	FALSE
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE

NOT	
VARIABLES	RETORNO
a	y
TRUE	FALSE
FALSE	TRUE

if - else - elif

El condicional alternativo o doble, es una solución muy útil la cual usarás mucho. Mira este ejemplo y trata de entenderlo... luego profundizamos.

```
1 usuario = 'Informatorio'
2 contraseña = 'contraseña_incorrecta'
3
4 if usuario == 'Informatorio' and contraseña == 'Info2023':
5     print('Acceso correcto. Bienvenido')
6 else:
7     print('Ingresaste mal tu usuario o contraseña. Reiniciá el programa y volvé a intentar.')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + - ☐ ☒

Ingresaste mal tu usuario o contraseña. Reiniciá el programa y volvé a intentar.

¿Y que pasó? Usando la sentencia else (que se lo puede traducir como sino en español), pudimos realizar una acción u otra acción, es decir, si no se ejecutara un bloque de código, se ejecutaría el otro bloque de código, y esto gracias a else.

Podemos ver que la indentación de else, está a la izquierda, a la altura de if, y el bloque de código, de else, también se encuentra a la altura del bloque de código del if.

De esta manera, el programa lee la sentencia if, y si tiene un retorno False ejecuta inmediatamente lo que se encuentra dentro del bloque de código de else. Entonces, podemos decir que, cuando utilizamos un condicional doble, uno de los dos bloques de código se va a ejecutar; ya sea el bloque de código de if (si se retorna un valor True de la condición), o el bloque de código de else (si se retorna un valor False de la condición if).

Algo para hacer notar, else no lleva condicional, ya que la condición se realiza en la primer parte, en el if.

¿Y qué pasa si necesitamos condiciones múltiples?

elif

Para responder a la pregunta, te mostraremos 3 ejemplos y volvemos a profundizar: (Ver pag. siguiente)

Ejemplo 1: se ingresan tres notas, se saca el promedio y se da un mensaje al usuario (para este caso Desaprobado).

```

1  nota1 = 1
2  nota2 = 3
3  nota3 = 5
4  promedio_notas = (nota1 + nota2 + nota3) / 3
5
6  if promedio_notas < 6:
7      print(f'Por lo visto no estás leyendo los apuntes ni practicando. Desaprobaste. Tu promedio es: {promedio_notas}')
8  elif promedio_notas < 8:
9      print(f'¡Aprobaste! ¡Pero con más práctica vas a lograr más! Tu promedio es: {promedio_notas}')
10 else:
11     print(f'¡Excelente! Se nota que estás leyendo los apuntes y practicando. Tu promedio es: {promedio_notas}')
12

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + -

Por lo visto no estás leyendo los apuntes ni practicando. Desaprobaste. Tu promedio es: 3.0

En este ejemplo podemos observar que el promedio de notas es de 3.0, por lo que según las condiciones que tenemos, el bloque de código que se ejecuta es el primero ya que el rango que tenemos como condición es entre 0 y 5.

Por lo que el programa verifica el primer condicional y, como retorna un valor True (ya que se cumple la condición del rango que pedimos) ejecuta el bloque de código que se encuentra dentro. El resto de los bloques no se ejecutarán porque se cumplió la primer condición.

Ejemplo 2: se ingresan tres notas, se saca el promedio y se da un mensaje al usuario (para este caso Aprobado pero...).

```

1  nota1 = 5
2  nota2 = 7
3  nota3 = 9
4  promedio_notas = (nota1 + nota2 + nota3) / 3
5
6  if promedio_notas < 6:
7      print(f'Por lo visto no estás leyendo los apuntes ni practicando. Desaprobaste. Tu promedio es: {promedio_notas}')
8  elif promedio_notas < 8:
9      print(f'¡Aprobaste! ¡Pero con más práctica vas a lograr más! Tu promedio es: {promedio_notas}')
10 else:
11     print(f'¡Excelente! Se nota que estás leyendo los apuntes y practicando. Tu promedio es: {promedio_notas}')

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + -

¡Aprobaste! ¡Pero con más práctica vas a lograr más! Tu promedio es: 7.0

En este segundo ejemplo vemos (que el alumno estudió más) que el promedio de notas es de 7.0, por lo que según las condiciones que tenemos, el bloque de código que se ejecuta es el segundo (donde ya tenemos la sentencia elif) ya que el rango que tenemos como condición en este caso es entre 6 y 7.99.

Por lo que el programa verifica el primer condicional y retorna un valor False, así que pasa a la siguiente condición y esta retorna un valor True, así que ejecuta ese bloque de código. Si te preguntas si se pueden seguir agregando más elif, la respuesta es sí!

Elif que sería la combinación entre **if** y **else** (interpretándolo del inglés sería como “sino si”) es una condición intermedia entre **if** y **else**. Se puede poner la cantidad que necesitemos, aunque siempre es conveniente mirar desde otra perspectiva el problema y, tratar de reducir y simplificar el código de la

forma más óptima posible, ya que mientras más código escrito, más complicado es leerlo y con una sintaxis deficiente (por uso excesivo de condicionales muchas veces), es aún peor.

Así que si, se puede usar la cantidad de **elif** que queramos, pero hay que ser precavidos.

Ejemplo 3: se ingresan tres notas, se saca el promedio y se da un mensaje al usuario (para este caso Excelente).

```
1 nota1 = 10
2 nota2 = 10
3 nota3 = 10
4 promedio_notas = (nota1 + nota2 + nota3) / 3
5
6 if promedio_notas < 6:
7     print(f'Por lo visto no estás leyendo los apuntes ni practicando. Desaprobaste. Tu promedio es: {promedio_notas}')
8 elif promedio_notas < 8:
9     print(f'¡Aprobaste! ¡Pero con más práctica vas a lograr más! Tu promedio es: {promedio_notas}')
10 else:
11     print(f'¡Excelente! Se nota que estás leyendo los apuntes y practicando. Tu promedio es: {promedio_notas}')
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + v [] [X]

¡Excelente! Se nota que estás leyendo los apuntes y practicando. Tu promedio es: 10.0

En este último caso, las dos primeras condiciones retornan un valor False, por lo que “por defecto” la ejecución del programa va a ingresar al bloque de código de else y ejecuta lo que encuentra adentro.

Podemos destacar que elif sí lleva condición a diferencia de else, por lo que siempre que agregues elif, también debes agregar una condición de comparación, tal cual se venía haciendo con if.

Se puede ver en estos 3 ejemplos, que el programa va a ingresar al condicional **if**, donde si este retorna un valor False, va a ejecutar el bloque de código que se encuentra en **else**,

pero si alguna de las anteriores condiciones retorna un valor True, ejecutará el código dentro del bloque donde obtuvo el retorno True y saldrá del condicional **if**.

Ahora que conocés la estructura de control condicional te invitamos a practicar, hacer tus propios programas, e incluso, ya puedes hacer un programa que pida usuario y contraseña para dar un mensaje (mejorando el código de ejemplo que te dimos antes).

> estructuras de control de flujo - loops

Estructuras de control de flujo

for - while loops

SEMANA 3



INTRODUCCIÓN

En este apartado, y para terminar con la parte de estructuras de control de flujo, te mostraremos 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
```

Como se ha podido ver en el ejemplo anterior, funciona con la misma estructura que vimos en 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, habrás notado que si lo traducimos al 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 puedas comprender mejor.

```

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

```

PROBLEMAS	SALIDA	COMENTARIOS
1		
2		
3		
4		
5		

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.

Variables especiales

Además, para ayudarnos en este tipo de bucle vamos a tener unos tipos de variables especiales que (como en el ejemplo que acabamos de ver), vamos a requerir al momento de generar nuestros bucles:

Contador

Tal cual 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.

Veamos 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 deseamos, 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 tipo float, ya que podemos sumar o restar décimas, centésimas, milésimas, etc., siempre será según lo que queramos realizar.*

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

El contador de nombre "pares" lleva un registro de las veces que aparecen números pares en la lista.

El contador con nombre "i" lleva un 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.

```

1  lista = [1, 2, 3, 4, 5]
2  pares = 0
3  i = 0
4
5  while i < len(lista):
6      if lista[i] % 2 == 0:
7          pares += 1
8          i += 1
9
10 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

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 las primeras páginas del apunte) 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.

¿Lo vas comprendiendo mejor?

Entonces, podemos resumir que en el ejemplo mostrado usamos contadores, también usamos en conjunto con la estructura while, la estructura if.

Con esto ya puedes ir relacionando aún más las formas de usar una estructura con otra, para pulir tus programas.

A continuación veremos otro tipo de variable especial:

Acumulador

La funcionalidad 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 daremos 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}')
10
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

El acumulador tiene la suma de: 15

Como puedes 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.

Veamos un ejemplo para que lo puedas comprender mejor:

```

1 lista = [1, 2, 3, 4, 5]
2 nro_buscado = 3
3 encontrado = False
4 i = 0
5
6 while i < len(lista):
7     if lista[i] == nro_buscado:
8         encontrado = True
9         break
10    i += 1
11
12 if encontrado:
13     print('El número se encuentra en la lista.')
14 else:
15     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, podemos finalizar el bucle y mostrar por pantalla que el número fue encontrado.

Si, en tal caso, no se cumpliera la condición que teníamos establecida, como es en este caso la búsqueda de un número, lo que se mostraría por pantalla, sería que "el número no se encontró".

WHILE

Ahora que te hemos explicado estas variables especiales, podemos volver 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.

Veamos otro ejemplo, integrando ejemplos anteriores, donde mejoraremos el login de usuario que vimos antes, donde habíamos pedido al usuario que ingrese su usuario y contraseña. Sin embargo, en ese ejemplo (pág. 31) si el usuario ingresaba mal alguno de los dos, debía reiniciar el programa. Con while vamos a poder mejorarlo de la siguiente manera (lo vemos en la próxima página).

Veamos un ejemplo para que lo puedas comprender mejor

```
1 usuario = input('Ingresá el usuario: ')
2 contraseña = input('Ingresá la contraseña: ')
3
4 while usuario != 'Informatorio' and contraseña != 'Info 2023':
5     print('El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.')
6     usuario = input('Ingresá el usuario: ')
7     contraseña = input('Ingresá la contraseña: ')
8 else:
9     print('Ingresaste correctamente. Bienvenido.')
10
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
Ingresá el usuario: sdfsf
Ingresá la contraseña: sdfsf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfsf
Ingresá la contraseña: sdfsf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfsf
Ingresá la contraseña: sdfsf
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: sdfefefes
Ingresá la contraseña: rgfgfdg
El usuario o contraseña ingresado es incorrecto. Intentá nuevamente.
Ingresá el usuario: Informatorio
Ingresá la contraseña: Info2023
Ingresaste correctamente. Bienvenido.
```

Hemos mejorado el programa de login que vimos antes, y esto gracias al bucle while. Además, como has visto, 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 puedes ver lo útil que es 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.

Veremos algunos ejemplos para que puedas entenderlo mejor:

Iterando sobre una lista

Supongamos que tienes una lista de números enteros y quieres imprimir cada número en la lista. Podemos utilizar un ciclo for para iterar sobre la lista de la siguiente manera (lo vemos en la siguiente página):

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

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.

Características del ciclo for

- 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.

Iterando sobre una cadena de texto

Veamos 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. 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 veremos 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

```

En este ejemplo "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.

Iterando sobre un rango de valores

Otra funcionalidad 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 DE

```

1
2
3
4
5

```

Como puedes ver, la función range genera un rango de valores desde 1 hasta 5 inclusive, para este caso (siempre se tomará un valor antes del último número que le pasemos).

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 - 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)
7

```

PROBLEMAS SALIDA CONSOLA DE DEPUR

```

1
2

```

Para este ejemplo 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)
7
```

PROBLEMAS SALIDA CONSOLA DE DEPURACION

```
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; habrás visto lo poderosas que son estas herramientas de estructuras de control de flujo y el potencial que tienen para realizar nuestras aplicaciones. Las utilizaremos siempre como programadores, por lo que lo que te recomendamos... practicar, practicar y practicar.

Puedes ir mejorando los ejemplos de código que te hemos mostrado anteriormente, implementando estas nuevas herramientas.