Módulo 4: Condicionales e iteradores

Introducción

Luego de haber cubierto las estructuras de datos y los distintos tipos de variables que existen en Python, en esta lectura se abordarán conceptos que le otorgarán inteligencia a nuestro programa. Por un lado, se tratarán los condicionales para poder evaluar distintos escenarios y actuar en consecuencia; y por otro, los iteradores para poder repetir una secuencia de instrucciones una determinada cantidad de veces, o para recorrer estructuras de datos previamente estudiadas.

Video de inmersión

Unidad 1. Condicionales

Tema 1. Valores booleanos

Hasta el momento hemos explicado distintos tipos de variables, desde las simples a las más complejas, y las operaciones que se pueden realizar entre ellas. Para que un algoritmo posea cierta inteligencia y no solo se limite a hacer operaciones, se debe incorporar un concepto adicional: la lógica condicional. Como su nombre lo indica, esto permitirá evaluar condiciones y modificar el flujo del programa dependiendo del resultado. Es decir, si sucede una cosa el programa sigue un camino; mientras que, si sucede otra, las acciones a tomar serán diferentes.

Al momento de querer cumplir un objetivo en específico a través de código, un programador desarrollará un programa que hará preguntas y resolverá operaciones para proporcionar respuestas al momento de ser ejecutados. El tipo de preguntas que un

programa es capaz de resolver no son demasiado complejas: solo puede identificar si algo es cierto (*True*), o si algo no es cierto (*False*). Estos son los valores booleanos que ya definimos anteriormente.

Para poder realizar preguntas, Python hace uso de una serie de operadores. Se comenzará por mencionar el principal: el operador de igualdad (==).

El operador previamente mencionado responderá a la pregunta "¿Estos dos valores son iguales?" En base a ello, devolverá el resultado *True* o *False*.

Figura 1: Operador de igualdad

Fuente: elaboración propia.

Cabe destacar la diferencia entre el operador de igualdad (==) con el operador de asignación (=), que fue utilizado reiteradas veces en lecturas anteriores:

☰ Operador de igualdad (==)

realiza una comparación entre dos valores. Por ejemplo: 2==2, o a==b. En estos casos, estoy comparando si el valor o variable de la derecha al operador es igual al de la izquierda.

☰ Operación de asignación (=)

asigna el valor de la variable de la derecha, a la variable de la izquierda del operador. Por ejemplo: a = 2.

Para hacer uso de este operador en Python y evaluar una condición comparando dos variables, se hace uso de la **sentencia *if***. De su traducción del inglés, la forma en que esta sentencia opera es siguiendo la lógica de "Si sucede esto, tomaré las siguientes acciones… Si no sucede, tomaré estas otras acciones".

La sintaxis que corresponde a la instrucción *if* es la siguiente:

Figura 2. *If*

```
if 2==2:
    print("Es correcto")
```

Fuente: elaboración propia.

Como se observa, luego de *if* se escribe la condición a evaluar utilizando el operador de igualdad. En las líneas siguientes, se describen las acciones a tomar en caso de que el resultado de la comparativa sea *True*. Estas acciones deben escribirse respetando la indentación correspondiente para que Python sepa interpretar que corresponden a la sentencia *if*.

Como al ejecutar el código anterior la condición a evaluar da como resultado *True*, se muestra en terminal el mensaje “Es correcto”. Si el resultado fuera *False*, no se tomaría ninguna acción.

De la misma manera, se puede trabajar con variables:

Figura 3. Variables

```
a= 10
b= 10

if a==b:
   print("Es correcto")
```

Fuente: elaboración propia.

Al ejecutar este último *script*, el resultado es el mismo obtenido con el anterior.

1. Actividad de lectura: Verdadero o Falso.

El operador de igualdad y de asignación pueden usarse de manera intercambiable solo cuando se utiliza en una sentencia *if*.

Verdadero

Falso

Justificación

Tema 2. Operadores de comparación

Es de suponer que el operador de igualdad no es el único operador de comparación disponible. También se pueden hacer preguntas de comparación con los siguientes operadores:

- Desigualdad (!=):

De manera opuesta al operador de igualdad, el operador de desigualdad devuelve el valor *True* cuando las variables a ambos lados del operador son distintas.

Figura 4. *True*

Fuente: elaboración propia

- Mayor que (>):

Supongamos que se desea saber si la cantidad de tornillos producida fue mayor a la cantidad de clavos. Para un caso de uso como este, se usaría el operador “Mayor que”.

Figura 5. Mayor que

Fuente: elaboración propia.

- Mayor o igual que (>=):

Si quisiera desarrollar un código que determine si una persona es mayor de edad, un operador que sería útil para realizar esta comparación es el operador “Mayor o igual que”:

Figura 6. Mayor o igual que

Fuente: elaboración propia.

Al ejecutar el *script* anterior, dado que la condición da como resultado *True*, se mostrará en pantalla el texto “Es mayor de edad”.

- Menor que (<):

Este operador de comparación devolverá como resultado *True* cuando la variable a la izquierda del mismo sea menor que la variable de la derecha, de manera opuesta al operador “Mayor que”.

Figura 7. Mayor que

```
if cantTornillos < cantClavos:
  print("Hay menos tornillos que clavos")
```

Fuente: elaboración propia.

- Menor o igual que (<=):

Supongamos que deseamos saber en una lista de autos, cuáles deben ser multados por exceso de velocidad. En este caso, un operador de comparación que sería útil es el operador “Menor o igual que”.

Figura 8. Menor o igual que

```
if velocidad <= 100:
  print("No será multado")
```

Fuente: elaboración propia.

2. Actividad de lectura: *Multiple choice*

¿Qué operador de comparación es útil a la hora de evaluar si una variable es menor que determinado valor?

Menor que

Mayor que

Menor o igual que

Mayor o igual que

Justificación

Tema 3. Condiciones anidadas: *if*/*elif*/*else*

En los ejemplos anteriores se observó cómo, a través de la instrucción *if*, se podía evaluar una condición mediante operadores de comparación. No obstante, solo se estaba tomando una acción cuando el resultado de la operación era *True*. En esta sección se explicará cómo tomar otras acciones cuando la condición principal a evaluar da como resultado *False*.

Para que el código tome un camino alternativo cuando la instrucción *if* da como resultado *False*, se utiliza la instrucción *else*. La lógica es: “Si esta condición no se cumple (*if*), entonces hago esto otro”.

La sentencia es la siguiente:

Figura 9. Sentencia

```
if 1==2:
   print("Los números son iguales")
else:
   print("Los números NO son iguales")
```

Fuente: elaboración propia.

Si consideramos que se ejecutan las acciones cuando el resultado de la comparación es *True*, al ejecutar el código anterior, se observa en pantalla el mensaje “Los números **no** son iguales”. Esto es debido a que las acciones de la instrucción *if* no se ejecutan por dar como resultado *False*. Para todo el resto de resultados posibles se definió una instrucción *else*, por lo tanto, se ejecutarán las acciones especificadas en esa sección.

A continuación, se dará un ejemplo con un operador de comparación “Mayor que”:

Figura 10. Mayor que

```
a = 8
if a > 6:
   print("La variable es mayor a 6")
else:
   print("La variable es menor a 6")
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior se lee en la terminal el mensaje “La variable es mayor a 6”,

dado que el resultado de la condición principal (instrucción *if*) es *True* y, por lo tanto, se ejecutan esas acciones en lugar de lo especificado en la instrucción *else*.

Como se observa en los ejemplos anteriores, mediante la sentencia *if* se evalúa una sola condición y en caso de que no se cumpla, para todo el resto de los escenarios se ejecutan las acciones bajo la sentencia *else* (en caso la misma haya sido especificada). No obstante, hay una manera de incluir diversos escenarios a ser evaluados mediante otra instrucción denominada *elif*.

La sentencia *elif* es una contracción de *else if*, y la lógica detrás de la misma es dar alternativas adicionales a la condición principal. Por lo tanto, la sintaxis se leería como “Si sucede esto (*if*), ejecuto estas acciones. Si no sucede, pero sucede esta otra cosa (*elif*), ejecuto estas otras acciones. En el caso de que nada de lo anterior haya sucedido, ejecuto estas otras acciones (*else*)”.

A continuación, se observa un ejemplo de esto en el código:

Figura 11. Ejemplo

```
if a > 6:
   print("La variable es mayor a 6")
elif a < 3:
   print("La variable es menor a 3")
else:
   print("La variable está entre 3 y 6")
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observarán distintos mensajes en pantalla dependiendo del valor de la variable “a”.

- Si a = 8: se observaría en pantalla el mensaje “La variable es mayor a 6”
- Si a = 5: se observaría en pantalla el mensaje “La variable está entre 3 y 6”
- Si a = 1: se observaría en pantalla el mensaje “La variable es menor a 3”

Pueden incluirse cuantas instrucciones *elif* se desee, en base a la cantidad de condiciones específicas que se quieran determinar en el código.

3. Actividad de lectura: Verdadero o falso

Siempre que se especifica una sentencia *if*, debe escribirse una instrucción *else* que la acompañe, aun así no se tome ninguna acción.

Verdadero

Falso

Justificación

Tema 4. Expresiones booleanas compuestas

En algunas ocasiones se desea evaluar como verdadera o falsa múltiples condiciones en una misma sentencia *if*. Por ejemplo, cuando se desea saber si una variable se encuentra entre dos valores. Esto podría realizarse, de manera poco práctica, con una sentencia *if* dentro de otra sentencia *if*:

Figura 12. *If*

Fuente: elaboración propia.

Sin embargo, Python provee la posibilidad de incluir múltiples condiciones bajo la misma instrucción a través de las palabras claves *and* (indicando que se deben cumplir todas las condiciones dadas para dar como resultado *True*) y *or* (se puede cumplir una condición o la otra, para dar como resultado *True*).

Repitiendo el último ejemplo tomando en cuenta lo anteriormente mencionado, el código quedaría de la siguiente manera:

Figura 13. Ejemplo

Fuente: elaboración propia.

Al ejecutar el *script* anterior, dado que se cumplen ambas condiciones (la variable a es mayor que 6 y, al mismo tiempo, es menor que 10), la instrucción da como resultado *True* y, por lo tanto, se muestra en pantalla el texto “La variable es mayor a 6 y menor que 10”.

Supongamos un escenario en que hay que determinar si se debe enviar el auto a realizar servicio técnico. Las condiciones dadas para que esto suceda es que el vehículo haya recorrido 30 mil kilómetros o que hayan pasado 3 años desde el último control (cualquiera de los dos indicadores informan la necesidad de un *service*). Esto, en un código, se vería de la siguiente manera:

Figura 14. Resultado

Fuente: elaboración propia.

Como se puede observar, si bien no se cumple la condición del kilometraje, sí se cumple la condición de la cantidad de años desde que se realizó el último servicio técnico. Por lo tanto, al ejecutar el *script* anterior se observa en pantalla el mensaje “Debe realizar *service*”.

4. Actividad de lectura: *Multiple choice*

¿Cuál de las siguientes palabras clave permite definir una expresión condicional compuesta?

and

also

or

in

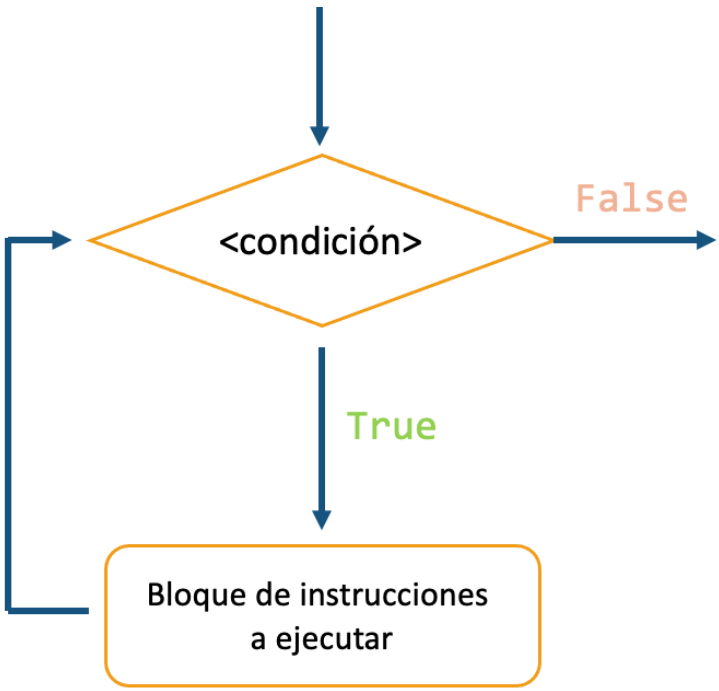
Justificación

Unidad 2. Iteradores

Tema 1. Iteración

Una iteración implica realizar una misma acción o serie de instrucciones reiteradas veces. Dada una condición, se repite un bloque de código una cantidad definida de veces o bien hasta que la condición a evaluar ya no sea verdadera. En programación, su propósito es variado y cuenta con infinidad de casos de uso: recorrer una lista en búsqueda de un valor, repetir una operación hasta que se logre cierta condición, ejecutar un programa en bucle de manera que nunca termine, entre otras.

Figura 15: Iteración



Fuente: elaboración propia.

En Python existen dos maneras de iterar sobre un bloque de código: el ciclo *While* y *For*.

Se utilizará una u otra dependiendo de la intención buscada por el programador. Por un lado, el ciclo *While* evaluará una condición y ejecutará una serie de instrucciones hasta que esa condición ya no sea verdadera. Por otro lado, el ciclo *For* no evalúa una condición, sino que determina de distintas maneras la cantidad específica de iteraciones que repetirá. Esto tendrá distintos casos de uso, ya sea para repetir una acción o para recorrer estructuras de datos previamente estudiadas.

A continuación, se cubrirá ambas instrucciones en mayor detalle.

5. Actividad de lectura: Verdadero o falso

Es posible definir que un ciclo iterativo se repita una cantidad definida de veces, o bien que se repita hasta que una condición ya no sea verdadera.

Verdadero

Falso

Justificación

Tema 2. Ciclo *While*

De su traducción en inglés, *while* significa “mientras”. Por lo tanto, como se puede inducir, un ciclo *While* evalúa una condición (al igual que una sentencia *if*) y, si la misma da como resultado *True*, se ejecuta el bloque de código (un conjunto de instrucciones) de manera iterativa hasta que la condición evaluada inicialmente deje de ser verdadera. Esto puede

darse porque el valor de una de las variables considerada en la condición de evaluación es modificado durante el ciclo *While*.

En cambio, si la condición a evaluar da como resultado *False* no se ejecuta ninguna de las instrucciones especificadas ni una sola vez.

La sintaxis del ciclo *While* es la siguiente:

Figura 16. Ciclo *While*

Fuente: elaboración propia.

Si se observa detenidamente, la sintaxis es exactamente igual a la de la sentencia *if* con la diferencia de que, mientras que con *if* se ejecutan las instrucciones una sola vez cuando se cumple la condición, en el ciclo *While* se repetirán un número indeterminado de veces mientras la condición sea evaluada como *True*.

A continuación, se dará un ejemplo de un ciclo *While* que evaluará la condición "¿La variable "iteraciones" es distinta de 0?". Dado que se define previamente un valor distinto de 0, se ejecutarán las acciones dentro del bucle. Las instrucciones a ejecutar constan de imprimir en pantalla el número de iteraciones restantes y de restar una unidad a la variable iteraciones. Por lo tanto, luego de varias iteraciones, eventualmente la variable valdrá 0 y el ciclo *While* terminará.

Figura 17. Resultado

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa el siguiente *output* en pantalla:

Figura 18. *Output*

Fuente: elaboración propia.

Como se mencionó anteriormente, un objetivo del programador puede ser que el ciclo sea infinito para que el programa nunca llegue al fin de su ejecución. Una de las maneras de lograr esto es a través de la siguiente condición:

Figura 19. Condición

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observará una secuencia infinita del mensaje “Soy una instrucción en un ciclo infinito”. Para interrumpir su ejecución, puede presionarse la combinación de teclas Ctrl+Z o Ctrl+C, dependiendo de la computadora.

6. Actividad de lectura: Opción correcta

Tanto la sentencia *if* como *while* requieren evaluar una condición para ejecutar las instrucciones definidas dentro de ellas. ¿Cuál de ellas ejecuta las instrucciones una sola vez, cuando la condición es cumplida?

If

While

Justificación

Tema 3. Ciclo *For*

Otra manera de iterar sobre una serie de instrucciones es a través del ciclo *For*. La diferencia con el ciclo *While* es que, en el ciclo *For* no se evalúa una condición, sino que existe previamente una cantidad definida de iteraciones a realizar. Si bien esto puede parecer una tarea que puede realizarse perfectamente con el ciclo *While* visto previamente, el objetivo de *For* apunta a algo más complejo: la capacidad de poder recorrer estructuras de datos (como listas o diccionarios), elemento por elemento. Más adelante se verá cómo realizar esta operación, pero en principio veamos cómo iterar sobre una cantidad definida de ciclos:

Figura 20. Resultado

```
for i in range(10):
  print(f"Iteracion nro {i}")
```

Fuente: elaboración propia.

En el *script* anterior se ven varios conceptos nuevos:

- El ciclo *For* se abre con la palabra reservada *for*.
- La variable que sigue a la palabra *for* (i) es una variable de control, cuenta las iteraciones del ciclo y toma el valor del elemento que se está recorriendo (se explorará esto más en detalle a continuación). Esta variable no debe estar definida previamente, sino que se le asigna un nombre directamente en la declaración del ciclo *For*, y será relevante durante la ejecución del mismo. Durante cada iteración, la misma tomará un valor diferente que puede ser utilizado o no en las instrucciones.
- La función *range*() se encarga de generar los valores que recorrerá el ciclo *For.* La misma genera una lista que contiene la cantidad de elementos definida en el argumento, comenzando por el número 0 y finalizando un número antes del especificado. En el caso del ejemplo, iría del número 0 al 9, siendo un total de 10 elementos.

Al ejecutar el *script* anterior, se observa el siguiente texto en pantalla:

Figura 21. Resultado

Fuente: elaboración propia.

Vamos a detallar la función *range*(): como se mostró en el ejemplo anterior, al pasarle el parámetro 10 y al utilizarla en conjunto con *for* se iteró sobre una instrucción un total de 10 veces. Sin embargo, al imprimir el resultado que tomó la variable en cada iteración, se observa que fue del 0 al 9. Si se desea que la función vaya del Numero_A al Numero_B, la sintaxis debería ser la siguiente:

Figura 22. Resultado

Fuente: elaboración propia.

Siguiendo el ejemplo anterior, si en lugar del 0 al 9 se requiere que el programa tome los valores del 1 al 5, entonces debería escribirse de la siguiente manera:

Figura 23. Resultado

Fuente: elaboración propia.

Cabe notar que en lugar de *“i”*, a modo de ejemplo se reemplazó el nombre de la variable por “ítem”. El nombre de esa variable puede ser cualquiera siempre que se utilice el mismo dentro de las instrucciones en el caso de que se desee utilizar su valor en las acciones a tomar (como en este último *script*).
Como se espera, al ejecutar el *script* anterior se obtiene el siguiente resultado en terminal:

Figura 24. Resultado en terminal

Fuente: elaboración propia.

La función *range*() no es más que una función que genera una lista de números en el intervalo especificado. Se podría lograr el mismo objetivo al realizar lo siguiente:

Figura 25. *range*()

```
lista_numeros = [1, 2, 3, 4, 5]
for i in lista_numeros:
   print(f"Iteracion nro {i}")
```

Fuente: elaboración propia.

Teniendo en cuenta esto último, se observa que, a través de la instrucción *for*, recorremos elemento por elemento de la lista. Podríamos usar este mismo criterio para recorrer cualquier tipo de lista. Por ejemplo, a continuación, se especifica una lista de palabras:

Figura 26. *for*

```
lista_palabras = ["Esto", "es", "un", "ejemplo"]
for palabra in lista_palabras:
   print(palabra)
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa el siguiente resultado en pantalla:

Figura 27. Resultado

```
Esto
es
un
ejemplo
```

Fuente: elaboración propia.

De la misma manera, esta ejecución no solo funciona con listas, sino también con los elementos de un diccionario. Recordemos que los diccionarios contaban con los métodos *keys*() y *values*() para acceder a las llaves y los valores del mismo, respectivamente. Si recorremos el diccionario con un ciclo *For*, podríamos acceder a las llaves y los valores uno por uno, ya sea para mostrarlos en pantalla de una manera en particular o para realizar algún otro tipo de procesamiento (por ejemplo, guardarlos en una tabla de Excel). A continuación, se muestra un ejemplo de cómo recorrer las llaves de un diccionario elemento por elemento:

Figura 28. Ejemplo

```
persona_1 = {"nombre": "Juan", "apellido": "Lopez", "edad": 42}

for datos in persona_1.keys():
  print(datos)
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa lo siguiente en la terminal:

Figura 29. Ejemplo

```
nombre
apellido
edad
```

Fuente: elaboración propia.

De la misma manera, se podrían recorrer los valores y mostrarlos:

Figura 30. Ejemplo

```
for valores in persona_1.values():
  print(valores)
```

Fuente: elaboración propia.

En este último caso, al ejecutar el *script* se observa lo siguiente:

Figura 31. Ejemplo

```
Juan
Lopez
42
```

Fuente: elaboración propia.

7. Actividad de lectura: Verdadero o falso

Un ciclo *For* es utilizado únicamente para iterar una cantidad definida de veces a través de la función *range*().

Verdadero

Falso

Justificación

Tema 4. Declaración *break* y *continue*

Hasta el momento se vieron casos de uso de iteraciones en que se completaba el ciclo y se ejecutaban todas las acciones dentro del mismo, hasta que dejaba de cumplirse una condición (ciclo *While*) o bien cuando se completaba una cantidad de iteraciones definida (ciclo *For*).

No obstante, existe también la posibilidad de enfrentar otras situaciones:

- No es necesario continuar iterando a través del ciclo en su totalidad para optimizar la ejecución del código, o bien es necesario interrumpir el ciclo para evitar algún error. Por lo tanto, debe existir la posibilidad de salir del ciclo al enfrentar esa opción.
- Se debe continuar con el próximo ciclo, sin terminar de completar con las instrucciones del ciclo actual. Esto sucede, por ejemplo, cuando se encuentra un valor que se estaba buscando y no tiene sentido seguir iterando sobre una estructura de datos.

Las instrucciones para poder realizar las operaciones previamente mencionadas son las siguientes:

☰ Break

de su traducción en inglés, romper o cortar. La operación *break*, que es una palabra clave, termina con el ciclo completo sin importar el número de iteración en el que se encuentre, ni la falta de instrucciones por ejecutar en el ciclo de iteración actual. Simplemente sale del ciclo completo en ese instante y se continúa ejecutando las líneas de código que siguen luego de la definición del ciclo *For/While*.

☰ Continue

de su traducción en inglés, continuar. A diferencia de la operación anterior, la instrucción *continue* solo saltea la iteración actual interrumpiendo las instrucciones restantes dentro de la misma, pero continúa con las iteraciones restantes del ciclo.

A continuación, se dará un ejemplo de cada una de ellas:

Figura 32. Ejemplo

```
for value in range(1,11):
    print(f"Iteración nro {value}")
    if value == 3:
        print(f"Encontré el valor {value}. Terminando la operación")
        break
```

Fuente: elaboración propia.

Como se observa, según la función *range*() definida, el programa debería iterar 10 veces desde el valor 1 al 10. No obstante, parte de las acciones a tomar es evaluar si el valor de la variable *value* es igual a 3. Si esta condición se cumple, se imprime un mensaje en pantalla y se interrumpe el ciclo *For.*

Al ejecutar el *script* anterior se observa el siguiente resultado en pantalla:

Figura 33. Resultado

```
Iteración nro 1
Iteración nro 2
Iteración nro 3
Encontré el valor 3. Terminando la operación
```

Fuente: elaboración propia.

En el siguiente ejemplo se utilizará la función *continue*. Se sigue con el mismo ejemplo de iterar 10 veces desde el valor 1 al 10. La diferencia en este caso es que no se interrumpirá el ciclo completo, sino que una vez encontrado el valor 3, se imprimirá un mensaje y se salteará la instrucción siguiente continuando con el resto de las iteraciones.

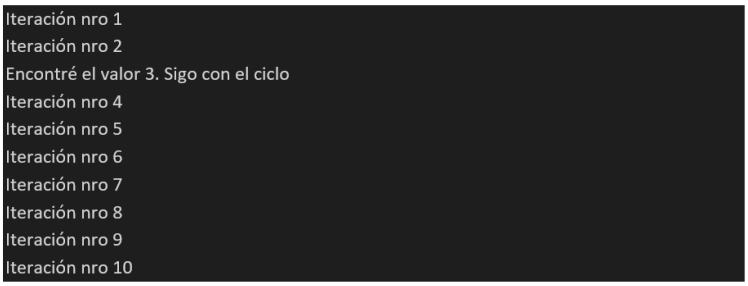
Figura 34. *continue*

```
for value in range(1,11):
    if value == 3:
        print(f"Encontré el valor {value}. Sigo con el ciclo")
        continue
    print(f"Iteración nro {value}")
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa el siguiente resultado en pantalla:

Figura 35. Resultado



Fuente: elaboración propia.

Se observa que, tal como se esperaba, se ejecuta una operación diferente al encontrar el valor 3 y se continúa luego de eso con el resto de las iteraciones.

8. Actividad de lectura: Opción correcta

¿Cuál de las siguientes instrucciones permite concluir un ciclo iterativo?

break

continue

Justificación

Video de habilidades

Preguntas de habilidades

¿Qué operador es posible utilizar para evaluar si un número es divisible por otro?

*

==

$

%

+

Justificación

La sentencia *if* permite evaluar hasta 3 condiciones al mismo tiempo.

Verdadero

Falso

Justificación

Para evaluar dos condiciones en la misma línea en una sentencia *if*, se utiliza el operador "*for*".

Verdadero

Falso

Justificación

¿Cuál es la palabra clave utilizada en una sentencia *if* para ejecutar una serie de instrucciones cuando ninguna de las condiciones evaluadas anteriormente se cumple?

End

Elif

Else if

Else

Or

Justificación

El ciclo *for* itera a través de una serie de valores hasta que la condición evaluada ya no sea verdadera.

Verdadero

Falso

Justificación

Cierre

En esta lectura se completan los fundamentos de Python, habiendo cubierto estructuras de datos complejas como ser las listas y diccionarios, cómo definirlas e interactuar con ellas de distintas maneras. Posteriormente, se explicaron los condicionales con el objetivo de poder actuar en consecuencia de distintos resultados obtenidos y modificar el flujo de un programa. Por último, se introdujo el concepto de iteradores, distintos tipos disponibles y cómo utilizarlos en conjunto con los conocimientos obtenidos previamente. Con las prácticas y conocimientos adquiridos, es posible realizar tanto desde programas muy simples hasta complejos. Las herramientas obtenidas a través de esta clase le permitirán a un desarrollador realizar diversos trabajos.

Glosario


Glosario interactivo
Buscar...
Clústeres
API
Big Data
C
Comparar variables
Código fuente
Data Science
Diccionario