curso python

cilsa

Estructuras de control iterativas

# Estructuras de control iterativas o bucles

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos

permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

* El bucle **while.**
* El bucle **for**.

## Bucle while

Este bucle, se encarga de ejecutar una misma acción “mientras que” una determinada condición se cumpla:

anio = 2020

while anio <= 2022:

print("Informes del Año", str(anio)) #Salida: los años desde el 2020 al 2022

anio += 1

Podrás notar que, en **cada iteración, incrementamos el valor de la variable que condiciona el bucle** (anio). Si no lo hiciéramos, esta variable siempre sería igual a 2020 y el bucle se ejecutará de forma infinita, ya que la condición (anio <= 2022) siempre se estaría cumpliendo.

### Bucle while con estructura condicional

Pero ¿Qué sucede si el valor que condiciona la iteración no es numérico y no puede incrementarse?

En ese caso, podremos utilizar una estructura de control condicional, anidada dentro del bucle, y frenar la ejecución cuando el condicional deje de cumplirse, con la palabra clave reservada break:

while True:

nombre = input("Indique su nombre: ")

if nombre:

break

El bucle anterior, incluye un condicional anidado que verifica si la variable nombre es verdadera (sólo será

verdadera si el usuario tipea un texto en pantalla cuando el nombre le es solicitado). Si es verdadera, el bucle

para (break). Sino, seguirá ejecutándose hasta que el usuario, ingrese un texto en pantalla.

## Bucle for

El bucle **for**, en Python, es aquel que nos permitirá iterar sobre una variable compleja, del tipo lista o tupla:

Por cada nombre en *mi\_lista*, imprimir *nombre*:

mi\_lista = ['Juan', 'Antonio', 'Pedro', 'Herminio']

for nombre in mi\_lista:

print(nombre) #Salida: Juan Antonio Pedro Herminio

Por cada color en *mi\_tupla*, imprimir *color*:

mi\_tupla = ('rosa', 'verde', 'celeste', 'amarillo')

for color in mi\_tupla:

print(color) #Salida: rosa verde celeste amarillo

En los ejemplos anteriores, **nombre** y **color**, son dos variables declaradas en tiempo de ejecución (es decir, se declaran dinámicamente durante el bucle), asumiendo como valor, el de cada elemento de la lista (o tupla) en cada iteración.

### Bucle for emulando while

Otra forma de iterar con el bucle for, puede emular a while:

Por cada año en el rango 2020 a 2022, imprimir la frase “Informes del Año año”

for anio in range(2020, 2022):

print("Informes del Año", str(anio)) #Salida años desde el 2020 al 2022

Como pueden ver, en el bucle for a diferencia del while, la condición de parada es un rango o una cantidad. Este se va a repetir hasta que esa cantidad se complete. En el caso de las listas, se ejecutará hasta pasar por todos los elementos de la misma (corta cuando la cantidad de elementos se acaba).

Es por ello que se dice que el bucle for se utiliza para “contar” elementos **finitos** (una cantidad determinada).

# Estructura de control Try Except

Por último, vamos a ver otra estructura, que no es iterativa. Es una estructura o bloque muy utilizada en programas de grandes magnitudes, el **Try Except**.

Esta estructura es la conocida como “capturadora de excepciones”. Su funcionalidad es la de si pasa

algo que detiene al flujo (un error) pueda capturar ese error, mostrarnos algún mensaje relacionado con el

mismo, pero no corta la ejecución del programa, sino que deja que continúe.

Esto sirve mucho cuando tenemos una parte del código que no es tan crítica, pero sí necesitamos que el

programa continúe con su ejecución, como puede ser una conexión a una base de datos, un error de un servicio, algún error de entrada, etc.

## Sintaxis del Try Except

Su forma de utilización es la siguiente:

try:

#codigo a ejecutar donde puede ocurrir un error

a = 5

b = 0

division = a/b

print(division)

except:

print(“hubo un problema”)

Aquí lo que sucede, es que como no se puede dividir por 0, nuestro programa automáticamente se frena y nos saltará un error por pantalla por defecto de Python. Lo que hacemos es decirle a Python, que está bien, sabemos que puede ocurrir un error, pero que solo lo vamos a capturar, dar el mensaje “hubo un problema” y seguiremos con nuestro programa.

## Tipos de errores

Lo que nos permite también el except, es que si conocemos de qué tipo es el error, podremos capturarlo y

renombrarlo para que nos dé mejor información de lo que está pasando:

print(x) #nos da un error de “NameError: name ‘x' is not defined” qué es que “x” no está definida

Para solucionar esto y que se entienda mejor el tipo de error, podríamos capturar este error:

try:

print(x)

except NameError:

print(“La variable que trata de mostrar por pantalla no está definida”)

Entonces ahora podremos entender mejor porque fue el error.

# Conclusión

Con estas estructuras podremos realizar verificaciones, automatizar procesos, y capturar errores.

Cada vez nos estamos acercando a crear un programa que pueda resolver cualquier problema que le

planteemos siempre y cuando esté a su alcance.

**RECUERDEN QUE LA MÁQUINA HACE LO QUE LE DECIMOS, ¡NO HACE MAGIA!**